

Package ‘np’

March 6, 2012

Version 0.40-13

Date 2012-03-05

Depends boot, cubature

Suggests quantreg, MASS

Title Nonparametric kernel smoothing methods for mixed data types

Author Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

Maintainer Jeffrey S. Racine <racinej@mcmaster.ca>

Description This package provides a variety of nonparametric (and semiparametric) kernel methods that seamlessly handle a mix of continuous, unordered, and ordered factor data types. We would like to gratefully acknowledge support from the Natural Sciences and Engineering Research Council of Canada (NSERC:www.nserc.ca), the Social Sciences and Humanities Research Council of Canada (SSHRC:www.sshrc.ca), and the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca).

License GPL

URL <https://github.com/JeffreyRacine/R-Package-np/>

Repository CRAN

Date/Publication 2012-03-06 06:40:16

R topics documented:

b.star	2
cps71	4
Engel95	5
gradients	7
Italy	8
np	9

npcdens	12
npcdensbw	20
npcmstest	27
npconmode	31
npdeneqtest	36
npdeptest	38
npindex	41
npindexbw	50
npksum	56
npplot	64
npplreg	76
npplregbw	82
npqcmstest	89
npqreg	92
npreg	96
npregbw	106
npregiv	113
npregivderiv	118
npscoef	123
npscoefbw	127
npsdeptest	133
npseed	136
npsigtest	137
npsymtest	141
npudens	144
npudensbw	151
npunitest	160
oecdpanel	164
se	165
uocquantile	166
wage1	167

Index**169**

b.star

*Compute Optimal Block Length for Stationary and Circular Bootstrap***Description**

b.star is a function which computes the optimal block length for the continuous variable data using the method described in Patton, Politis and White (2009).

Usage

```
b.star(data,  
       Kn = NULL,  
       mmax= NULL,  
       Bmax = NULL,  
       c = NULL,  
       round = FALSE)
```

Arguments

data	data, an $n \times k$ matrix, each column being a data series.
Kn	See footnote c, page 59, Politis and White (2004). Defaults to $\text{ceiling}(\log_{10}(n))$.
mmax	See Politis and White (2004). Defaults to $\text{ceiling}(\sqrt{n})+Kn$.
Bmax	See Politis and White (2004). Defaults to $\text{ceiling}(\min(3*\sqrt{n}, n/3))$.
c	See Politis and White (2004). Defaults to $qnorm(0.975)$.
round	whether to round the result or not. Defaults to FALSE.

Details

b.star is a function which computes optimal block lengths for the stationary and circular bootstraps. This allows the use of tsboot from the `boot` package to be fully automatic by using the output from b.star as an input to the argument `l =` in tsboot. See below for an example.

Value

A $k \times 2$ matrix of optimal bootstrap block lengths computed from data for the stationary bootstrap and circular bootstrap (column 1 is for the stationary bootstrap, column 2 the circular).

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Patton, A. and D.N. Politis and H. White (2009), "CORRECTION TO "Automatic block-length selection for the dependent bootstrap" by D. Politis and H. White", *Econometric Reviews* 28(4), 372-375.
- Politis, D.N. and J.P. Romano (1994), "Limit theorems for weakly dependent Hilbert space valued random variables with applications to the stationary bootstrap", *Statistica Sinica* 4, 461-476.
- Politis, D.N. and H. White (2004), "Automatic block-length selection for the dependent bootstrap", *Econometric Reviews* 23(1), 53-70.

Examples

```
set.seed(12345)

# Function to generate an AR(1) series

ar.series <- function(phi,epsilon) {
  n <- length(epsilon)
  series <- numeric(n)
  series[1] <- epsilon[1]/(1-phi)
  for(i in 2:n) {
    series[i] <- phi*series[i-1] + epsilon[i]
  }
  return(series)
}

yt <- ar.series(0.1,rnorm(10000))
b.star(yt,round=TRUE)

yt <- ar.series(0.9,rnorm(10000))
b.star(yt,round=TRUE)
```

cps71

Canadian High School Graduate Earnings

Description

Canadian cross-section wage data consisting of a random sample taken from the 1971 Canadian Census Public Use Tapes for male individuals having common education (grade 13). There are 205 observations in total.

Usage

```
data("cps71")
```

Format

A data frame with 2 columns, and 205 rows.

logwage the first column, of type numeric

age the second column, of type integer

Source

Aman Ullah

References

Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.

Examples

```
data("cps71")
attach(cps71)

plot(age, logwage, xlab="Age", ylab="log(wage)")

detach(cps71)
```

Engel95

1995 British Family Expenditure Survey

Description

British cross-section data consisting of a random sample taken from the British Family Expenditure Survey for 1995. The households consist of married couples with an employed head-of-household between the ages of 25 and 55 years. There are 1655 household-level observations in total.

Usage

```
data("Engel95")
```

Format

A data frame with 10 columns, and 1655 rows.

food expenditure share on food, of type numeric
catering expenditure share on catering, of type numeric
alcohol expenditure share on alcohol, of type numeric
fuel expenditure share on fuel, of type numeric
motor expenditure share on motor, of type numeric
fares expenditure share on fares, of type numeric
leisure expenditure share on leisure, of type numeric
logexp logarithm of total expenditure, of type numeric
logwages logarithm of total earnings, of type numeric
nkids number of children, of type numeric

Source

Richard Blundell and Dennis Kristensen

References

Blundell, R. and X. Chen and D. Kristensen (2007), "Semi-Nonparametric IV Estimation of Shape-Invariant Engel Curves," *Econometrica*, 75, 1613-1669.
Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Examples

```

## Not run:
## Example - compute nonparametric instrumental regression using
## Landweber-Fridman iteration of Fredholm integral equations of the
## first kind.

## We consider an equation with an endogenous regressor ('z') and an
## instrument ('w'). Let  $y = \phi(z) + u$  where  $\phi(z)$  is the function of
## interest. Here  $E(u|z)$  is not zero hence the conditional mean  $E(y|z)$ 
## does not coincide with the function of interest, but if there exists
## an instrument  $w$  such that  $E(u|w) = 0$ , then we can recover the
## function of interest by solving an ill-posed inverse problem.

data(Engel95)

## Sort on logexp (the endogenous regressor) for plotting purposes

Engel95 <- Engel95[order(Engel95$logexp),]

attach(Engel95)

model.iv <- npregiv(y=food,z=logexp,w=logwages,method="Landweber-Fridman")
phihat <- model.iv$phihat

## Compute the non-IV regression (i.e. regress y on z)

ghat <- npreg(food~logexp,regtype="ll")

## For the plots, restrict focal attention to the bulk of the data
## (i.e. for the plotting area trim out 1/4 of one percent from each
## tail of y and z)

trim <- 0.0025

plot(logexp,food,
     ylab="Food Budget Share",
     xlab="log(Total Expenditure)",
     xlim=quantile(logexp,c(trim,1-trim)),
     ylim=quantile(food,c(trim,1-trim)),
     main="Nonparametric Instrumental Kernel Regression",
     type="p",
     cex=.5,
     col="lightgrey")

lines(logexp,phihat,col="blue",lwd=2,lty=2)

lines(logexp,fitted(ghat),col="red",lwd=2,lty=4)

legend(quantile(logexp,trim),quantile(food,1-trim),
      c(expression(paste("Nonparametric IV: ",hat(varphi)(logexp))),
        "Nonparametric Regression: E(food | logexp)"),
      lty=c(2,4),

```

```
col=c("blue","red"),
lwd=c(2,2))

## End(Not run)
```

gradients

Extract Gradients

Description

gradients is a generic function which extracts gradients from objects.

Usage

```
gradients(x, ...)
```

S3 method for class 'condensity'

```
gradients(x, errors = FALSE, ...)
```

S3 method for class 'condistribution'

```
gradients(x, errors = FALSE, ...)
```

S3 method for class 'npregression'

```
gradients(x, errors = FALSE, ...)
```

S3 method for class 'qregression'

```
gradients(x, errors = FALSE, ...)
```

S3 method for class 'singleindex'

```
gradients(x, errors = FALSE, ...)
```

Arguments

x	an object for which the extraction of gradients is meaningful.
...	other arguments.
errors	a logical value specifying whether or not standard errors of gradients are desired. Defaults to FALSE.

Details

This function provides a generic interface for extraction of gradients from objects.

Value

Gradients extracted from the model object x.

Note

This method currently only supports objects from the `np` library.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

See the references for the method being interrogated via `gradients` in the appropriate help file. For example, for the particulars of the gradients for nonparametric regression see the references in `npreg`

See Also

`fitted`, `residuals`, `coef`, and `se`, for related methods; `np` for supported objects.

Examples

```
x <- runif(10)
y <- x + rnorm(10, sd = 0.1)
gradients(npreg(y~x, gradients=TRUE))
```

Italy

Italian GDP Panel

Description

Italian GDP growth panel for 21 regions covering the period 1951-1998 (millions of Lire, 1990=base). There are 1008 observations in total.

Usage

```
data("Italy")
```

Format

A data frame with 2 columns, and 1008 rows.

year the first column, of type ordered

gdp the second column, of type numeric: millions of Lire, 1990=base

Source

Giovanni Baiocchi

References

Baiocchi, G. (2006), “Economic Applications of Nonparametric Methods,” Ph.D. Thesis, University of York.

Examples

```
data("Italy")
attach(Italy)

plot(ordered(year), gdp, xlab="Year (ordered factor)",
     ylab="GDP (millions of Lire, 1990=base)")

detach(Italy)
```

 np

Nonparametric Kernel Smoothing Methods for Mixed Data Types

Description

This package provides a variety of nonparametric and semiparametric kernel methods that seamlessly handle a mix of continuous, unordered, and ordered factor data types (unordered and ordered factors are often referred to as ‘nominal’ and ‘ordinal’ categorical variables respectively). A vignette containing many of the examples found in the help files accompanying the **np** package that is intended to serve as a gentle introduction to this package can be accessed via `vignette("np", package="np")`.

For a listing of all routines in the **np** package type: `library(help="np")`.

Bandwidth selection is a key aspect of sound nonparametric and semiparametric kernel estimation. **np** is designed from the ground up to make bandwidth selection the focus of attention. To this end, one typically begins by creating a ‘bandwidth object’ which embodies all aspects of the method, including specific kernel functions, data names, data types, and the like. One then passes these bandwidth objects to other functions, and those functions can grab the specifics from the bandwidth object thereby removing potential inconsistencies and unnecessary repetition. Furthermore, many functions such as `plot` (which automatically calls `npplot`) can work with the bandwidth object directly without having to do the subsequent companion function evaluation.

As of **np** version 0.20-0, we allow the user to combine these steps. When using **np** versions 0.20-0 and higher, if the first step (bandwidth selection) is not performed explicitly then the second step will automatically call the omitted first step bandwidth selector using defaults unless otherwise specified, and the bandwidth object could then be retrieved retroactively if so desired via `objectname$bw`s. Furthermore, options for bandwidth selection will be passed directly to the bandwidth selector function. Note that the combined approach would not be a wise choice for certain applications such as when bootstrapping (as it would involve unnecessary computation since the bandwidths would properly be those for the original sample and not the bootstrap resamples) or when conducting quantile regression (as it would involve unnecessary computation when different quantiles are computed from the same conditional cumulative distribution estimate).

There are two ways in which you can interact with functions in **np**, either i) using data frames, or ii) using a formula interface, where appropriate.

To some, it may be natural to use the data frame interface. The R `data.frame` function preserves a variable's type once it has been cast (unlike `cbind`, which we avoid for this reason). If you find this most natural for your project, you first create a data frame casting data according to their type (i.e., one of continuous (default, `numeric`), `factor`, `ordered`). Then you would simply pass this data frame to the appropriate np function, for example `npudensbw(dat=data)`.

To others, however, it may be natural to use the formula interface that is used for the regression examples, among others. For nonparametric regression functions such as `npreg`, you would proceed as you would using `lm` (e.g., `bw <- npregbw(y~factor(x1)+x2)`) except that you would of course not need to specify, e.g., polynomials in variables, interaction terms, or create a number of dummy variables for a factor. Every function in np supports both interfaces, where appropriate.

Note that if your factor is in fact a character string such as, say, X being either "MALE" or "FEMALE", np will handle this directly, i.e., there is no need to map the string values into unique integers such as (0,1). Once the user casts a variable as a particular data type (i.e., `factor`, `ordered`, or continuous (default, `numeric`)), all subsequent methods automatically detect the type and use the appropriate kernel function and method where appropriate.

All estimation methods are fully multivariate, i.e., there are no limitations on the number of variables one can model (or number of observations for that matter). Execution time for most routines is, however, exponentially increasing in the number of observations and increases with the number of variables involved.

Nonparametric methods include unconditional density (distribution), conditional density (distribution), regression, mode, and quantile estimators along with gradients where appropriate, while semiparametric methods include single index, partially linear, and smooth (i.e., varying) coefficient models.

A number of tests are included such as consistent specification tests for parametric regression and quantile regression models along with tests of significance for nonparametric regression.

A variety of bootstrap methods for computing standard errors, nonparametric confidence bounds, and bias-corrected bounds are implemented.

A variety of bandwidth methods are implemented including fixed, nearest-neighbor, and adaptive nearest-neighbor.

A variety of data-driven methods of bandwidth selection are implemented, while the user can specify their own bandwidths should they so choose (either a raw bandwidth or scaling factor).

A flexible plotting utility, `npplot` (which is automatically invoked by `plot`), facilitates graphing of multivariate objects. An example for creating postscript graphs using the `npplot` utility and pulling this into a LaTeX document is provided.

The function `npksum` allows users to create or implement their own kernel estimators or tests should they so desire.

The underlying functions are written in C for computational efficiency. Despite this, due to their nature, data-driven bandwidth selection methods involving multivariate numerical search can be time-consuming, particularly for large datasets. A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

To cite the np package, type `citation("np")` from within R for details.

Details

The kernel methods in np employ the so-called ‘generalized product kernels’ found in Hall, Racine, and Li (2004), Li and Racine (2003), Li and Racine (2004), Li and Racine (2007), Ouyang, Li, and Racine (2006), and Racine and Li (2004), among others. For details on a particular method, kindly refer to the original references listed above.

We briefly describe the particulars of various univariate kernels used to generate the generalized product kernels that underlie the kernel estimators implemented in the np package. In a nutshell, the generalized kernel functions that underlie the kernel estimators in np are formed by taking the product of univariate kernels such as those listed below. When you cast your data as a particular type (continuous, factor, or ordered factor) in a data frame or formula, the routines will automatically recognize the type of variable being modelled and use the appropriate kernel type for each variable in the resulting estimator.

Second Order Gaussian (*x* is continuous) $k(z) = \exp(-z^2/2)/\sqrt{2\pi}$, where $z = (x_i - x)/h$, and $h > 0$.

Second Order Epanechnikov (*x* is continuous) $k(z) = 3(1 - z^2/5)/(4\sqrt{5})$ if $z^2 < 5$, 0 otherwise, where $z = (x_i - x)/h$, and $h > 0$.

Uniform (*x* is continuous) $k(z) = 1/2$ if $|z| < 1$, 0 otherwise, where $z = (x_i - x)/h$, and $h > 0$.

Aitchison and Aitken (*x* is a (discrete) factor) $l(x_i, x, \lambda) = 1 - \lambda$ if $x_i = x$, and $\lambda/(c - 1)$ if $x_i \neq x$, where c is the number of (discrete) outcomes assumed by the factor x . Note that λ must lie between 0 and $(c - 1)/c$.

Wang and van Ryzin (*x* is a (discrete) ordered factor) $l(x_i, x, \lambda) = 1 - \lambda$ if $|x_i - x| = 0$, and $(1 - \lambda)\lambda^{|x_i - x|}/2$ if $|x_i - x| \geq 1$. Note that λ must lie between 0 and 1.

Li and Racine (*x* is a (discrete) factor) $l(x_i, x, \lambda) = 1$ if $x_i = x$, and λ if $x_i \neq x$. Note that λ must lie between 0 and 1.

Li and Racine (*x* is a (discrete) ordered factor) $l(x_i, x, \lambda) = 1$ if $|x_i - x| = 0$, and $\lambda^{|x_i - x|}$ if $|x_i - x| \geq 1$. Note that λ must lie between 0 and 1.

So, if you had two variables, x_{i1} and x_{i2} , and x_{i1} was continuous while x_{i2} was, say, binary (0/1), and you created a data frame of the form `X <- data.frame(x1, factor(x2))`, then the kernel function used by np would be $K(\cdot) = k(\cdot) \times l(\cdot)$ where the particular kernel functions $k(\cdot)$ and $l(\cdot)$ would be, say, the second order Gaussian (`ckertype="gaussian"`) and Aitchison and Aitken (`ukertype="aitchisonaitken"`) kernels by default, respectively.

Note that higher order continuous kernels (i.e., fourth, sixth, and eighth order) are derived from the second order kernels given above (see Li and Racine (2007) for details).

For particulars on any given method, kindly see the references listed for the method in question.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

Maintainer: Jeffrey S. Racine <racinej@mcmaster.ca>

We are grateful to John Fox and Achim Zeileis for their valuable input and encouragement. We would like to gratefully acknowledge support from the Natural Sciences and Engineering Research Council of Canada (NSERC:www.nserc.ca), the Social Sciences and Humanities Research Council of Canada (SSHRC:www.sshrc.ca), and the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca)

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, 99, 1015-1026.
- Li, Q. and J.S. Racine (2003), "Nonparametric estimation of distributions with categorical and continuous data," *Journal of Multivariate Analysis*, 86, 266-292.
- Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," *Statistica Sinica*, 14, 485-512.
- Ouyang, D. and Q. Li and J.S. Racine (2006), "Cross-validation and the estimation of probability distributions with categorical data," *Journal of Nonparametric Statistics*, 18, 69-100.
- Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," *Journal of Econometrics*, 119, 99-130.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.
- Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

 npcdens

Kernel Conditional Density and Distribution Estimates with Mixed Data Types

Description

npcdens computes kernel conditional density estimates on $p + q$ -variate evaluation data, given a set of training data (both explanatory and dependent) and a bandwidth specification (a conbandwidth object or a bandwidth vector, bandwidth type, and kernel type) using the method of Hall, Racine, and Li (2004). Similarly npcdist computes kernel conditional cumulative distribution estimates. The data may be continuous, discrete (unordered and ordered factors), or some combination thereof.

Usage

```
npcdens(bws, ...)
```

```
## S3 method for class 'formula'
```

```
npcdens(bws, data = NULL, newdata = NULL, ...)
```

```
## S3 method for class 'call'
```

```
npcdens(bws, ...)
```

```
## S3 method for class 'conbandwidth'
npcdens(bws,
        txdat = stop("invoked without training data 'txdat'"),
        tydat = stop("invoked without training data 'tydat'"),
        exdat,
        eydat,
        gradients = FALSE,
        ...)

## Default S3 method:
npcdens(bws, txdat, tydat, ...)
```

Arguments

bws	a bandwidth specification. This can be set as a conbandwidth object returned from a previous invocation of <code>npcdensbw</code> , or as a $p + q$ -vector of bandwidths, with each element i up to $i = p$ corresponding to the bandwidth for column i in <code>txdat</code> , and each element i from $i = p + 1$ to $i = p + q$ corresponding to the bandwidth for column $i - p$ in <code>tydat</code> . If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, training data, and so on.
gradients	a logical value specifying whether to return estimates of the gradients at the evaluation points. Defaults to FALSE.
...	additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is necessary if you specify <code>bws</code> as a $p+q$ -vector and not a conbandwidth object, and you do not desire the default behaviours. To do this, you may specify any of <code>bwmethod</code> , <code>bwscaling</code> , <code>bwtype</code> , <code>cxkertype</code> , <code>cxkerorder</code> , <code>cykertype</code> , <code>cykerorder</code> , <code>uxkertype</code> , <code>uykertype</code> , <code>oxkertype</code> , <code>oykertype</code> , as described in <code>npcdensbw</code> .
data	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npcdensbw</code> was called.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
txdat	a p -variate data frame of sample realizations of explanatory data (training data). Defaults to the training data used to compute the bandwidth object.
tydat	a q -variate data frame of sample realizations of dependent data (training data). Defaults to the training data used to compute the bandwidth object.
exdat	a p -variate data frame of explanatory data on which conditional densities will be evaluated. By default, evaluation takes place on the data provided by <code>txdat</code> .
eydat	a q -variate data frame of dependent data on which conditional densities will be evaluated. By default, evaluation takes place on the data provided by <code>tydat</code> .

Details

npcdens and `npclist` implement a variety of methods for estimating multivariate conditional distributions ($p+q$ -variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2004) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

Training and evaluation input data may be a mix of continuous (default), unordered discrete (to be specified in the data frames using `factor`), and ordered discrete (to be specified in the data frames using `ordered`). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `np` for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npcdens returns a condensity object, similarly `npclist` returns a condistribution object. The generic accessor functions `fitted`, `se`, and `gradients`, extract estimated values, asymptotic standard errors on estimates, and gradients, respectively, from the returned object. Furthermore, the functions `predict`, `summary` and `plot` support objects of both classes. The returned objects have the following components:

<code>xbw</code>	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, <code>txdat</code>
<code>ybw</code>	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, <code>tydat</code>
<code>xeval</code>	the evaluation points of the explanatory data
<code>yeval</code>	the evaluation points of the dependent data
<code>condens</code> or <code>condist</code>	estimates of the conditional density (cumulative distribution) at the evaluation points
<code>conderr</code>	standard errors of the conditional density (cumulative distribution) estimates
<code>congrad</code>	if invoked with <code>gradients = TRUE</code> , estimates of the gradients at the evaluation points
<code>congerr</code>	if invoked with <code>gradients = TRUE</code> , standard errors of the gradients at the evaluation points
<code>log_likelihood</code>	log likelihood of the conditional density estimate

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, 99, 1015-1026.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.
- Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also

[npudens](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we load Giovanni
# Baiocchi's Italian GDP panel (see Italy for details), and compute the
# likelihood cross-validated bandwidths (default) using a second-order
# Gaussian kernel (default). Note - this may take a minute or two
# depending on the speed of your computer.

data("Italy")
attach(Italy)

# First, compute the bandwidths... note that this may take a minute or
# two depending on the speed of your computer.

bw <- npcdensbw(formula=gdp~ordered(year))

# Next, compute the condensity object...

fhat <- npcdens(bws=bw)

# The object fhat now contains results such as the estimated conditional
# density function (fhat$condens) and so on...

summary(fhat)

## Not run:
```

```
# Call the plot() function to visualize the results (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems).

plot(bw)

# To plot the conditional distribution we use cdf=TRUE in plot
# (<ctrl>-C will interrupt on *NIX systems, <esc> will interrupt on MS
# Windows systems)

plot(bw, cdf=TRUE)

detach(Italy)

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we load Giovanni
# Baiocchi's Italian GDP panel (see Italy for details), and compute the
# likelihood cross-validated bandwidths (default) using a second-order
# Gaussian kernel (default). Note - this may take a minute or two
# depending on the speed of your computer.

data("Italy")
attach(Italy)

# First, compute the bandwidths... note that this may take a minute or
# two depending on the speed of your computer.

# Note - we cast 'X' and 'y' as data frames so that plot() can
# automatically grab names (this looks like overkill, but in
# multivariate settings you would do this anyway, so may as well get in
# the habit).

X <- data.frame(year=ordered(year))
y <- data.frame(gdp)

bw <- npcdensbw(xdat=X, ydat=y)

# Next, compute the condensity object...

fhat <- npcdens(bws=bw)

# The object fhat now contains results such as the estimated conditional
# density function (fhat$condens) and so on...

summary(fhat)

# Call the plot() function to visualize the results (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows systems).

plot(bw)

# To plot the conditional distribution we use cdf=TRUE in plot
# (<ctrl>-C will interrupt on *NIX systems, <esc> will interrupt on MS
# Windows systems)
```

```
plot(bw, cdf=TRUE)

detach(Italy)

# EXAMPLE 2 (INTERFACE=FORMULA): For this example, we load the old
# faithful geyser data from the R 'datasets' library and compute the
# conditional density and conditional distribution functions.

library("datasets")
data("faithful")
attach(faithful)

# Note - this may take a few minutes depending on the speed of your
# computer...

bw <- npcdensbw(formula=eruptions~waiting)

summary(bw)

# Plot the density function (<ctrl>-C will interrupt on *NIX systems,
# <esc> will interrupt on MS Windows systems).

plot(bw)

# Plot the distribution function (cdf=TRUE) (<ctrl>-C will interrupt on
# *NIX systems, <esc> will interrupt on MS Windows systems)

plot(bw, cdf=TRUE)

detach(faithful)

# EXAMPLE 2 (INTERFACE=DATA FRAME): For this example, we load the old
# faithful geyser data from the R 'datasets' library and compute the
# conditional density and conditional distribution functions.

library("datasets")
data("faithful")
attach(faithful)

# Note - this may take a few minutes depending on the speed of your
# computer...

# Note - we cast 'X' and 'y' as data frames so that plot() can
# automatically grab names (this looks like overkill, but in
# multivariate settings you would do this anyway, so may as well get in
# the habit).

X <- data.frame(waiting)
y <- data.frame(eruptions)

bw <- npcdensbw(xdat=X, ydat=y)
```

```

summary(bw)

# Plot the density function (<ctrl>-C will interrupt on *NIX systems,
# <esc> will interrupt on MS Windows systems)

plot(bw)

# Plot the distribution function (cdf=TRUE) (<ctrl>-C will interrupt on
# *NIX systems, <esc> will interrupt on MS Windows systems)

plot(bw, cdf=TRUE)

detach(faithful)

# EXAMPLE 3 (INTERFACE=FORMULA): Replicate the DGP of Klein & Spady
# (1993) (see their description on page 405, pay careful attention to
# footnote 6 on page 405).

set.seed(123)

n <- 1000

# x1 is chi-squared having 3 df truncated at 6 standardized by
# subtracting 2.348 and dividing by 1.511

x <- rchisq(n, df=3)
x1 <- (ifelse(x < 6, x, 6) - 2.348)/1.511

# x2 is normal (0, 1) truncated at +- 2 divided by 0.8796

x <- rnorm(n)
x2 <- ifelse(abs(x) < 2, x, 2) / 0.8796

# y is 1 if y* > 0, 0 otherwise.

y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

# Generate data-driven bandwidths (likelihood cross-validation). Note -
# this may take a few minutes depending on the speed of your computer...

bw <- npcdensbw(formula=factor(y)~x1+x2)

# Next, create the evaluation data in order to generate a perspective
# plot

x1.seq <- seq(min(x1), max(x1), length=50)
x2.seq <- seq(min(x2), max(x2), length=50)
X.eval <- expand.grid(x1=x1.seq, x2=x2.seq)

data.eval <- data.frame(y=factor(rep(1, nrow(X.eval))), x1=X.eval[,1], x2=X.eval[,2])

# Now evaluate the conditional probability for y=1 and for the
# evaluation Xs

```

```
fit <- fitted(npcdens(bws=bw,newdata=data.eval))

# Finally, coerce the data into a matrix for plotting with persp()

fit.mat <- matrix(fit, 50, 50)

# Generate a perspective plot similar to Figure 2 b of Klein and Spady
# (1993)

persp(x1.seq,
      x2.seq,
      fit.mat,
      col="white",
      ticktype="detailed",
      expand=0.5,
      axes=FALSE,
      box=FALSE,
      main="Estimated Nonparametric Probability Perspective",
      theta=310,
      phi=25)

# EXAMPLE 3 (INTERFACE=DATA FRAME): Replicate the DGP of Klein & Spady
# (1993) (see their description on page 405, pay careful attention to
# footnote 6 on page 405).

set.seed(123)

n <- 1000

# x1 is chi-squared having 3 df truncated at 6 standardized by
# subtracting 2.348 and dividing by 1.511

x <- rchisq(n, df=3)
x1 <- (ifelse(x < 6, x, 6) - 2.348)/1.511

# x2 is normal (0, 1) truncated at +- 2 divided by 0.8796

x <- rnorm(n)
x2 <- ifelse(abs(x) < 2, x, 2) / 0.8796

# y is 1 if y* > 0, 0 otherwise.

y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

# Create the X matrix

X <- cbind(x1, x2)

# Generate data-driven bandwidths (likelihood cross-validation). Note -
# this may take a few minutes depending on the speed of your computer...

bw <- npcdensbw(xdat=X, ydat=factor(y))
```

```

# Next, create the evaluation data in order to generate a perspective
# plot

x1.seq <- seq(min(x1), max(x1), length=50)
x2.seq <- seq(min(x2), max(x2), length=50)
X.eval <- expand.grid(x1=x1.seq,x2=x2.seq)

# Now evaluate the conditional probability for y=1 and for the
# evaluation Xs

fit <- fitted(npcdens(exdat=X.eval,
                     eydat=factor(rep(1, nrow(X.eval))),
                     bws=bw))

# Finally, coerce the data into a matrix for plotting with persp()

fit.mat <- matrix(fit, 50, 50)

# Generate a perspective plot similar to Figure 2 b of Klein and Spady
# (1993)

persp(x1.seq,
      x2.seq,
      fit.mat,
      col="white",
      ticktype="detailed",
      expand=0.5,
      axes=FALSE,
      box=FALSE,
      main="Estimated Nonparametric Probability Perspective",
      theta=310,
      phi=25)

## End(Not run)

```

 npcdensbw

Kernel Conditional Density Bandwidth Selection with Mixed Data Types

Description

npcdensbw computes a conbandwidth object for estimating the conditional density of a $p + q$ -variate kernel density estimator defined over mixed continuous and discrete (unordered, ordered) data using either the normal-reference rule-of-thumb, likelihood cross-validation, or least-squares cross validation using the method of Hall, Racine, and Li (2004).

Usage

```
npcdensbw(...)
```

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

## S3 method for class 'NULL'
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws, ...)

## S3 method for class 'conbandwidth'
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws,
          bandwidth.compute = TRUE,
          fast = FALSE,
          auto = TRUE,
          nmulti,
          remin = TRUE,
          itmax = 10000,
          ftol = 1.19209e-07,
          tol = 1.49012e-08,
          small = 2.22045e-16,
          ...)

## Default S3 method:
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws,
          bandwidth.compute = TRUE,
          fast,
          auto,
          nmulti,
          remin,
          itmax,
          ftol,
          tol,
          small,
          bwmethod,
          bwscaling,
          bwtype,
          cxkertype,
          cxkerorder,
          cykertype,
          cykerorder,
          uxkertype,
          uykertype,
          oxkertype,
          oykertype,
```

...)

Arguments

formula	a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the <code>na.action</code> setting of options, and is <code>na.fail</code> if that is unset. The (recommended) default is <code>na.omit</code> .
call	the original function call. This is passed internally by <code>np</code> when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.
xdat	a p -variate data frame of explanatory data on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
ydat	a q -variate data frame of dependent data which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
bws	a bandwidth specification. This can be set as a <code>conbandwidth</code> object returned from a previous invocation, or as a $p+q$ -vector of bandwidths, with each element i up to $i = p$ corresponding to the bandwidth for column i in <code>xdat</code> , and each element i from $i = p + 1$ to $i = p + q$ corresponding to the bandwidth for column $i - p$ in <code>ydat</code> . In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.
...	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
fast	a logical value specifying whether a significantly faster, memory intensive procedure should be used when doing least-squares cross-validation. Recommended for datasets of moderate size. Use of this on datasets of size larger than ~ 1000 observations will result in decreased performance. For more information see the <code>auto</code> argument. Defaults to <code>FALSE</code> .
auto	a logical value specifying whether to allow the code to attempt to automatically select the fastest routine, using a heuristic, to compute the least-squares cross-validation function. Defaults to <code>TRUE</code> .
bwmethod	which method to use to select bandwidths. <code>cv.ml</code> specifies likelihood cross-validation, <code>cv.ls</code> specifies least-squares cross-validation, and <code>normal-reference</code>

	just computes the ‘rule-of-thumb’ bandwidth h_j using the standard formula $h_j = 1.06\sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of the j th continuous variable defined as $\min(\text{standard deviation, interquartile range}/1.349)$, n the number of observations, P the order of the kernel, and l the number of continuous variables. Note that when there exist factors and the normal-reference rule is used, there is zero smoothing of the factors. Defaults to <code>cv.ml</code> .
<code>bwscaling</code>	a logical value that when set to <code>TRUE</code> the supplied bandwidths are interpreted as ‘scale factors’ (c_j), otherwise when the value is <code>FALSE</code> they are interpreted as ‘raw bandwidths’ (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j \sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of continuous variable j defined as $\min(\text{standard deviation, interquartile range}/1.349)$, n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j . Defaults to <code>FALSE</code> .
<code>bwtype</code>	character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the <code>conbandwidth</code> object. Defaults to <code>fixed</code> . Option summary: <code>fixed</code> : compute fixed bandwidths <code>generalized_nn</code> : compute generalized nearest neighbors <code>adaptive_nn</code> : compute adaptive nearest neighbors
<code>bandwidth.compute</code>	a logical value which specifies whether to do a numerical search for bandwidths or not. If set to <code>FALSE</code> , a <code>conbandwidth</code> object will be returned with bandwidths set to those specified in <code>bws</code> . Defaults to <code>TRUE</code> .
<code>cxkertype</code>	character string used to specify the continuous kernel type for <code>xdat</code> . Can be set as <code>gaussian</code> , <code>epanechnikov</code> , or <code>uniform</code> . Defaults to <code>gaussian</code> .
<code>cxkerorder</code>	numeric value specifying kernel order for <code>xdat</code> (one of (2, 4, 6, 8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.
<code>cykertype</code>	character string used to specify the continuous kernel type for <code>ydat</code> . Can be set as <code>gaussian</code> , <code>epanechnikov</code> , or <code>uniform</code> . Defaults to <code>gaussian</code> .
<code>cykerorder</code>	numeric value specifying kernel order for <code>ydat</code> (one of (2, 4, 6, 8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.
<code>uxkertype</code>	character string used to specify the unordered categorical kernel type. Can be set as <code>aitchisonaitken</code> or <code>liracine</code> . Defaults to <code>aitchisonaitken</code> .
<code>uykertype</code>	character string used to specify the unordered categorical kernel type. Can be set as <code>aitchisonaitken</code> .
<code>oxkertype</code>	character string used to specify the ordered categorical kernel type. Can be set as <code>wangvanryzin</code> or <code>liracine</code> . Defaults to <code>wangvanryzin</code> .
<code>oykertype</code>	character string used to specify the ordered categorical kernel type. Can be set as <code>wangvanryzin</code> .
<code>nmulti</code>	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points

remin	a logical value which when set as TRUE the search routine restarts from located minima for a minor gain in accuracy. Defaults to TRUE. Defaults to TRUE.
itmax	integer number of iterations before failure in the numerical optimization routine. Defaults to 10000.
ftol	tolerance on the value of the cross-validation function evaluated at located minima. Defaults to $1.19e-07$ (FLT_EPSILON).
tol	tolerance on the position of located minima of the cross-validation function. Defaults to $1.49e-08$ ($\sqrt{\text{DBL_EPSILON}}$).
small	a small number, at about the precision of the data type used. Defaults to $2.22e-16$ (DBL_EPSILON).

Details

npcdensbw implements a variety of methods for choosing bandwidths for multivariate distributions ($p + q$ -variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2004) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms (direction set (Powell’s) methods in multidimensions).

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

npcdensbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the `xdat` and `ydat` parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frames `xdat` and `ydat` may be a mix of continuous (default), unordered discrete (to be specified in the data frames using `factor`), and ordered discrete (to be specified in the data frames using `ordered`). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `np` for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form `dependent data ~ explanatory data`, where `dependent data` and `explanatory data` are both series of variables specified by name, separated by the separation character ‘+’. For example, `y1 + y2 ~ x1 + x2` specifies that the bandwidths for the joint distribution of variables `y1` and `y2` conditioned on `x1` and `x2` are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npcdensbw returns a conbandwidth object, with the following components:

xbw	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, xdat
ybw	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, ydat
fval	objective function value at minimum

if bwtype is set to fixed, an object containing bandwidths (or scale factors if bwscaling = TRUE) is returned. If it is set to generalized_nn or adaptive_nn, then instead the k th nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables.

The functions [predict](#), [summary](#) and [plot](#) support objects of type conbandwidth.

Usage Issues

If you are using data of mixed types, then it is advisable to use the [data.frame](#) function to construct your input data and not [cbind](#), since [cbind](#) will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the i th observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(5, ncol(xdat,ydat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set bws=bw on subsequent calls to this routine where bw is the initial bandwidth object). A version of this package using the Rmpi wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, 99, 1015-1026.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.

Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also

[bw.nrd](#), [bw.SJ](#), [hist](#), [npudens](#), [npudist](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we compute the
# likelihood cross-validated bandwidths (default) using a second-order
# Gaussian kernel (default). Note - this may take a minute or two
# depending on the speed of your computer.

data("Italy")
attach(Italy)

bw <- npcdensbw(formula=gdp~ordered(year))

# The object bw can be used for further estimation using
# npcdens(), plotting using plot() etc. Entering the name of
# the object provides useful summary information, and names() will also
# provide useful information.

summary(bw)

# Note - see the example for npudensbw() for multiple illustrations
# of how to change the kernel function, kernel order, and so forth.

detach(Italy)

## Not run:

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we compute the
# likelihood cross-validated bandwidths (default) using a second-order
# Gaussian kernel (default). Note - this may take a minute or two
# depending on the speed of your computer.

data("Italy")
attach(Italy)

bw <- npcdensbw(xdat=ordered(year), ydat=gdp)

# The object bw can be used for further estimation using
# npcdens(), plotting using plot() etc. Entering the name of
# the object provides useful summary information, and names() will also
# provide useful information.

summary(bw)
```

```
# Note - see the example for npudensbw() for multiple illustrations
# of how to change the kernel function, kernel order, and so forth.

detach(Italy)

## End(Not run)
```

 npcmstest

Kernel Consistent Model Specification Test with Mixed Data Types

Description

npcmstest implements a consistent test for correct specification of parametric regression models (linear or nonlinear) as described in Hsiao, Li, and Racine (2007).

Usage

```
npcmstest(formula,
          data = NULL,
          subset,
          xdat,
          ydat,
          model = stop(paste(sQuote("model"), " has not been provided")),
          distribution = c("bootstrap", "asymptotic"),
          boot.method = c("iid", "wild", "wild-rademacher"),
          boot.num = 399,
          pivot = TRUE,
          density.weighted = TRUE,
          random.seed = 42,
          ...)
```

Arguments

formula	a symbolic description of variables on which the test is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
subset	an optional vector specifying a subset of observations to be used.
model	a model object obtained from a call to lm (or glm). Important: the call to either glm or lm must have the arguments x=TRUE and y=TRUE or npcmstest will not work. Also, the test is based on residual bootstrapping hence the outcome must be continuous (which rules out Logit, Probit, and Count models).
xdat	a p -variate data frame of explanatory data (training data) used to calculate the regression estimators.

<code>ydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>xdat</code> .
<code>distribution</code>	a character string used to specify the method of estimating the distribution of the statistic to be calculated. <code>bootstrap</code> will conduct bootstrapping. <code>asymptotic</code> will use the normal distribution. Defaults to <code>bootstrap</code> .
<code>boot.method</code>	a character string used to specify the bootstrap method. <code>iid</code> will generate independent identically distributed draws. <code>wild</code> will use a wild bootstrap. <code>wild-rademacher</code> will use a wild bootstrap with Rademacher variables. Defaults to <code>iid</code> .
<code>boot.num</code>	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
<code>pivot</code>	a logical value specifying whether the statistic should be normalised such that it approaches $N(0, 1)$ in distribution. Defaults to <code>TRUE</code> .
<code>density.weighted</code>	a logical value specifying whether the statistic should be weighted by the density of <code>xdat</code> . Defaults to <code>TRUE</code> .
<code>random.seed</code>	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
<code>...</code>	additional arguments supplied to control bandwidth selection on the residuals. One can specify the bandwidth type, kernel types, and so on. To do this, you may specify any of <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> , as described in npregbw . This is necessary if you specify <code>bws</code> as a p -vector and not a bandwidth object, and you do not desire the default behaviours.

Value

`npcmstest` returns an object of type `cmstest` with the following components, components will contain information related to J_n or I_n depending on the value of `pivot`:

<code>Jn</code>	the statistic J_n
<code>In</code>	the statistic I_n
<code>Omega.hat</code>	as described in Hsiao, C. and Q. Li and J.S. Racine.
<code>q.*</code>	the various quantiles of the statistic J_n (or I_n if <code>pivot=FALSE</code>) are in components <code>q.90</code> , <code>q.95</code> , <code>q.99</code> (one-sided 1%, 5%, 10% critical values)
<code>P</code>	the P-value of the statistic
<code>Jn.bootstrap</code>	if <code>pivot=TRUE</code> contains the bootstrap replications of J_n
<code>In.bootstrap</code>	if <code>pivot=FALSE</code> contains the bootstrap replications of I_n

[summary](#) supports object of type `cmstest`.

Usage Issues

npcmstest supports regression objects generated by `lm` and uses features specific to objects of type `lm` hence if you attempt to pass objects of a different type the function cannot be expected to work.

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.

Hsiao, C. and Q. Li and J.S. Racine (2007), "A consistent model specification test with mixed categorical and continuous data," *Journal of Econometrics*, 140, 802-826.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Maasoumi, E. and J.S. Racine and T. Stengos (2007), "Growth and convergence: a profile of distribution dynamics and mobility," *Journal of Econometrics*, 136, 483-508.

Murphy, K. M. and F. Welch (1990), "Empirical age-earnings profiles," *Journal of Labor Economics*, 8, 202-229.

Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

Examples

```
# EXAMPLE 1: For this example, we conduct a consistent model
# specification test for a parametric wage regression model that is
# quadratic in age. The work of Murphy and Welch (1990) would suggest
# that this parametric regression model is misspecified.
```

```
data("cps71")
attach(cps71)
```

```
model <- lm(logwage~age+I(age^2), x=TRUE, y=TRUE)
```

```
plot(age, logwage)
lines(age, fitted(model))
```

```
# Note - this may take a few minutes depending on the speed of your
# computer...
```

```
npcmstest(model = model, xdat = age, ydat = logwage)
```

```
## Not run:
```

```
# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next try Murphy & Welch's (1990) suggested quintic specification.

model <- lm(logwage~age+I(age^2)+I(age^3)+I(age^4)+I(age^5), x=TRUE, y=TRUE)

plot(age, logwage)
lines(age, fitted(model))

X <- data.frame(age)

# Note - this may take a few minutes depending on the speed of your
# computer...

npcmstest(model = model, xdat = age, ydat = logwage)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Note - you can pass in multiple arguments to this function. For
# instance, to use local linear rather than local constant regression,
# you would use npcmtstest(model, X, regtype="ll"), while you could also
# change the kernel type (default is second order Gaussian), numerical
# search tolerance, or feed in your own vector of bandwidths and so
# forth.

detach(cps71)

# EXAMPLE 2: For this example, we replicate the application in Maasoumi,
# Racine, and Stengos (2007) (see oecdpanel for details). We
# estimate a parametric model that is used in the literature, then
# subject it to the model specification test.

data("oecdpanel")
attach(oecdpanel)

model <- lm(growth ~ oecd +
            factor(year) +
            initgdp +
            I(initgdp^2) +
            I(initgdp^3) +
            I(initgdp^4) +
            popgro +
            inv +
            humancap +
            I(humancap^2) +
            I(humancap^3) - 1,
            x=TRUE,
            y=TRUE)
```

```

X <- data.frame(factor(oecd), factor(year), initgdp, popgro, inv, humancap)

npcmstest(model = model, xdat = X, ydat = growth)

detach(oecdpanel)

## End(Not run)

```

npconmode

Kernel Modal Regression with Mixed Data Types

Description

npconmode performs kernel modal regression on mixed data, and finds the conditional mode given a set of training data, consisting of explanatory data and dependent data, and possibly evaluation data. Automatically computes various in sample and out of sample measures of accuracy.

Usage

```

npconmode(bws, ...)

## S3 method for class 'formula'
npconmode(bws, data = NULL, newdata = NULL, ...)

## S3 method for class 'call'
npconmode(bws, ...)

## Default S3 method:
npconmode(bws, txdat, tydat, ...)

## S3 method for class 'conbandwidth'
npconmode(bws,
          txdat = stop("invoked without training data 'txdat'"),
          tydat = stop("invoked without training data 'tydat'"),
          exdat,
          eydat,
          ...)

```

Arguments

bws	a bandwidth specification. This can be set as a conbandwidth object returned from an invocation of npcdensbw
...	additional arguments supplied to specify the bandwidth type, kernel types, and so on, detailed below. This is necessary if you specify bws as a $p + q$ -vector and not a conbandwidth object, and you do not desire the default behaviours.

<code>data</code>	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npcdensbw</code> was called.
<code>newdata</code>	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
<code>txdat</code>	a p -variate data frame of explanatory data (conditioning data) used to calculate the regression estimators. Defaults to the training data used to compute the bandwidth object.
<code>tydat</code>	a one (1) dimensional vector of unordered or ordered factors, containing the dependent data. Defaults to the training data used to compute the bandwidth object.
<code>exdat</code>	a p -variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>txdat</code> .
<code>eydat</code>	a one (1) dimensional numeric or integer vector of the true values (outcomes) of the dependent variable. By default, evaluation takes place on the data provided by <code>tydat</code> .

Value

`npconmode` returns a `conmode` object with the following components:

<code>conmode</code>	a vector of type factor (or ordered factor) containing the conditional mode at each evaluation point
<code>condens</code>	a vector of numeric type containing the density estimates at each evaluation point
<code>xeval</code>	a data frame of evaluation points
<code>yeval</code>	a vector of type factor (or ordered factor) containing the actual outcomes, or NA if not provided
<code>confusion.matrix</code>	the confusion matrix or NA if outcomes are not available
<code>CCR.overall</code>	the overall correct classification ratio, or NA if outcomes are not available
<code>CCR.byoutcome</code>	a numeric vector containing the correct classification ratio by outcome, or NA if outcomes are not available
<code>fit.mcfadden</code>	the McFadden-Puig-Kerschner performance measure or NA if outcomes are not available

The functions [mode](#), and [fitted](#) may be used to extract the conditional mode estimates, and the conditional density estimates at the conditional mode, respectively, from the resulting object. Also, [summary](#) supports `conmode` objects.

Usage Issues

If you are using data of mixed types, then it is advisable to use the [data.frame](#) function to construct your input data and not [cbind](#), since [cbind](#) will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, 99, 1015-1026.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- McFadden, D. and C. Puig and D. Kerschner (1977), "Determinants of the long-run demand for electricity," *Proceedings of the American Statistical Association (Business and Economics Section)*, 109-117.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.
- Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we use the
# birthweight data taken from the MASS library, and compute a parametric
# logit model and a nonparametric conditional mode model. We then
# compare their confusion matrices and summary measures of
# classification ability.

library("MASS")
data("birthwt")
attach(birthwt)

# Fit a parametric logit model with low (0/1) as the dependent
# variable and age, lwt, and smoke (0/1) as the covariates

# From ?birthwt
# 'low' indicator of birth weight less than 2.5kg
# 'smoke' smoking status during pregnancy
# 'race' mother's race ('1' = white, '2' = black, '3' = other)
# 'ht' history of hypertension
# 'ui' presence of uterine irritability
# 'ftv' number of physician visits during the first trimester
# 'age' mother's age in years
# 'lwt' mother's weight in pounds at last menstrual period

model.logit <- glm(low~factor(smoke)+
                    factor(race)+
```

```

        factor(ht)+
        factor(ui)+
        ordered(ftv)+
        age+
        lwt,
        family=binomial(link=logit))

# Generate the confusion matrix and correct classification ratio

cm <- table(low, ifelse(fitted(model.logit)>0.5, 1, 0))
ccr <- sum(diag(cm))/sum(cm)

# Now do the same with a nonparametric model. Note - this may take a
# few minutes depending on the speed of your computer...

bw <- npcdensbw(formula=factor(low)~factor(smoke)+
                factor(race)+
                factor(ht)+
                factor(ui)+
                ordered(ftv)+
                age+
                lwt)

model.np <- npconmode(bws=bw)

# Compare confusion matrices from the logit and nonparametric model

# Logit

cm
ccr

# Nonparametric
summary(model.np)

detach(birthwt)

## Not run:

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we use the
# birthweight data taken from the MASS library, and compute a parametric
# logit model and a nonparametric conditional mode model. We then
# compare their confusion matrices and summary measures of
# classification ability.

library("MASS")
data("birthwt")
attach(birthwt)

# Fit a parametric logit model with low (0/1) as the dependent
# variable and age, lwt, and smoke (0/1) as the covariates

# From ?birthwt

```

```
# 'low' indicator of birth weight less than 2.5kg
# 'smoke' smoking status during pregnancy
# 'race' mother's race ('1' = white, '2' = black, '3' = other)
# 'ht' history of hypertension
# 'ui' presence of uterine irritability
# 'ftv' number of physician visits during the first trimester
# 'age' mother's age in years
# 'lwt' mother's weight in pounds at last menstrual period

model.logit <- glm(low~factor(smoke)+
                  factor(race)+
                  factor(ht)+
                  factor(ui)+
                  ordered(ftv)+
                  age+
                  lwt,
                  family=binomial(link=logit))

# Generate the confusion matrix and correct classification ratio

cm <- table(low, ifelse(fitted(model.logit)>0.5, 1, 0))
ccr <- sum(diag(cm))/sum(cm)

# Now do the same with a nonparametric model...

X <- data.frame(factor(smoke),
               factor(race),
               factor(ht),
               factor(ui),
               ordered(ftv),
               age,
               lwt)

y <- factor(low)

# Note - this may take a few minutes depending on the speed of your
# computer...

bw <- npcdensbw(xdat=X, ydat=y)

model.np <- npconmode(bws=bw)

# Compare confusion matrices from the logit and nonparametric model

# Logit

cm
ccr

# Nonparametric
summary(model.np)

detach(birthwt)
```

```
## End(Not run)
```

 npdeneqtest

Kernel Consistent Density Equality Test with Mixed Data Types

Description

npdeneqtest implements a consistent integrated squared difference test for equality of densities as described in Li, Maasoumi, and Racine (2009).

Usage

```
npdeneqtest(x = NULL,
            y = NULL,
            bw.x = NULL,
            bw.y = NULL,
            boot.num = 399,
            random.seed = 42,
            ...)
```

Arguments

<code>x, y</code>	data frames for the two samples for which one wishes to test equality of densities. The variables in each data frame must be the same (i.e. have identical names).
<code>bw.x, bw.y</code>	optional bandwidth objects for <code>x, y</code>
<code>boot.num</code>	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
<code>random.seed</code>	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
<code>...</code>	additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is used if you do not pass in bandwidth objects and you do not desire the default behaviours. To do this, you may specify any of <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> .

Details

npdeneqtest computes the integrated squared density difference between the estimated densities/probabilities of two samples having identical variables/datatypes. See Li, Maasoumi, and Racine (2009) for details.

Value

npdeneqtest returns an object of type `deneqtest` with the following components

<code>Tn</code>	the (standardized) statistic <code>Tn</code>
<code>In</code>	the (unstandardized) statistic <code>In</code>
<code>Tn.bootstrap</code>	contains the bootstrap replications of <code>Tn</code>
<code>In.bootstrap</code>	contains the bootstrap replications of <code>In</code>
<code>Tn.P</code>	the P-value of the <code>Tn</code> statistic
<code>In.P</code>	the P-value of the <code>In</code> statistic
<code>boot.num</code>	number of bootstrap replications

[summary](#) supports object of type `deneqtest`.

Usage Issues

If you are using data of mixed types, then it is advisable to use the [data.frame](#) function to construct your input data and not [cbind](#), since [cbind](#) will typically not work as intended on mixed data types and will coerce the data to the same type.

It is crucial that both data frames have the same variable names.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Li, Q. and E. Maasoumi and J.S. Racine (2009), "A Nonparametric Test for Equality of Distributions with Mixed Categorical and Continuous Data," *Journal of Econometrics*, 148, pp 186-200.

See Also

[npdeptest](#), [npsdeptest](#), [npsymtest](#), [npunitest](#)

Examples

```
set.seed(1234)

## Distributions are equal

n <- 250

sample.A <- data.frame(x=rnorm(n))
sample.B <- data.frame(x=rnorm(n))

npdeneqtest(sample.A, sample.B, boot.num=99)

Sys.sleep(5)
```

```
## Distributions are unequal

sample.A <- data.frame(x=rnorm(n))
sample.B <- data.frame(x=rchisq(n,df=5))

npdeneqtest(sample.A,sample.B,boot.num=99)

## Not run:
## Mixed datatypes, distributions are equal

sample.A <- data.frame(a=rnorm(n),b=factor(rbinom(n,2,.5)))
sample.B <- data.frame(a=rnorm(n),b=factor(rbinom(n,2,.5)))

npdeneqtest(sample.A,sample.B,boot.num=99)

Sys.sleep(5)

## Mixed datatypes, distributions are unequal

sample.A <- data.frame(a=rnorm(n),b=factor(rbinom(n,2,.5)))
sample.B <- data.frame(a=rnorm(n,sd=10),b=factor(rbinom(n,2,.25)))

npdeneqtest(sample.A,sample.B,boot.num=99)

## End(Not run)
```

npdeptest

Kernel Consistent Pairwise Nonlinear Dependence Test for Univariate Processes

Description

npdeptest implements the consistent metric entropy test of pairwise independence as described in Maasoumi and Racine (2002).

Usage

```
npdeptest(data.x = NULL,
          data.y = NULL,
          method = c("integration","summation"),
          bootstrap = TRUE,
          boot.num = 399,
          random.seed = 42)
```

Arguments

data.x, data.y two univariate vectors containing two variables that are of type `numeric`.

method	a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as <code>integration</code> or <code>summation</code> (see below for details). Defaults to <code>integration</code> .
bootstrap	a logical value which specifies whether to conduct the bootstrap test or not. If set to <code>FALSE</code> , only the statistic will be computed. Defaults to <code>TRUE</code> .
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Details

`npsdeptest` computes the nonparametric metric entropy (normalized Hellinger of Granger, Maasoumi and Racine (2004)) for testing pairwise nonlinear dependence between the densities of two data series. See Maasoumi and Racine (2002) for details. Default bandwidths are of the Kullback-Leibler variety obtained via likelihood cross-validation. The null distribution is obtained via bootstrap resampling under the null of pairwise independence.

`npdeptest` computes the distance between the joint distribution and the product of marginals (i.e. the joint distribution under the null), $D[f(y, \hat{y}), f(y) \times f(\hat{y})]$. Examples include, (a) a measure/test of “fit”, for in-sample values of a variable y and its fitted values, \hat{y} , and (b) a measure of “predictability” for a variable y and its predicted values \hat{y} (from a user implemented model).

The summation version of this statistic will be numerically unstable when `data.x` and `data.y` lack common support or are sparse (the summation version involves division of densities while the integration version involves differences). Warning messages are produced should this occur (‘integration recommended’) and should be heeded.

Value

`npdeptest` returns an object of type `deptest` with the following components

<code>Srho</code>	the statistic <code>Srho</code>
<code>Srho.bootstrap.vec</code>	contains the bootstrap replications of <code>Srho</code>
<code>P</code>	the P-value of the <code>Srho</code> statistic
<code>bootstrap</code>	a logical value indicating whether bootstrapping was performed
<code>boot.num</code>	number of bootstrap replications
<code>bw.data.x</code>	the numeric bandwidth for <code>data.x</code> marginal density
<code>bw.data.y</code>	the numeric bandwidth for <code>data.y</code> marginal density
<code>bw.joint</code>	the numeric matrix of bandwidths for data and lagged data joint density at lag <code>num.lag</code>

`summary` supports object of type `deptest`.

Usage Issues

The integration version of the statistic uses multidimensional numerical methods from the [cubature](#) package. See [adaptIntegrate](#) for details. The integration version of the statistic will be substantially slower than the summation version, however, it will likely be both more accurate and powerful.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), “A dependence metric for possibly non-linear processes”, *Journal of Time Series Analysis*, 25, 649-669.

Maasoumi, E. and J.S. Racine (2002), “Entropy and Predictability of Stock Market Returns,” *Journal of Econometrics*, 107, 2, pp 291-312.

See Also

[npdeneqtest](#), [npsdeptest](#), [npsymtest](#), [npunitest](#)

Examples

```
set.seed(1234)

## Test/measure lack of fit between y and its fitted value from a
## regression model when x is relevant using the 'summation' version.

n <- 100
x <- rnorm(n)
y <- 1 + x + rnorm(n)
model <- lm(y~x)
y.fit <- fitted(model)

npdeptest(y,y.fit,boot.num=99,method="summation")

Sys.sleep(5)

## Test/measure lack of fit between y and its fitted value from a
## regression model when x is irrelevant using the 'summation' version.

n <- 100
x <- runif(n,-2,2)
y <- 1 + rnorm(n)
model <- lm(y~x)
y.fit <- fitted(model)

npdeptest(x,y,boot.num=99,method="summation")

## Not run:
## Test/measure lack of fit between y and its fitted value from a
```

```

## regression model when x is relevant using the 'integration'
## version (default, slower than summation version).

n <- 100
x <- rnorm(n)
y <- 1 + x + rnorm(n)
model <- lm(y~x)
y.fit <- fitted(model)

npdeptest(y,y.fit,boot.num=99)

Sys.sleep(5)

## Test/measure lack of fit between y and its fitted value from a
## regression model when x is irrelevant using the 'integration'
## version (default, slower than summation version).

n <- 100
x <- runif(n,-2,2)
y <- 1 + rnorm(n)
model <- lm(y~x)
y.fit <- fitted(model)

npdeptest(x,y,boot.num=99)

## End(Not run)

```

npindex

Semiparametric Single Index Model

Description

npindex computes a semiparametric single index model for a dependent variable and p -variate explanatory data using the model $Y = G(X\beta) + \epsilon$, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a npindexbw bandwidth specification. Note that for this semiparametric estimator, the bandwidth object contains parameters for the single index model and the (scalar) bandwidth for the index function.

Usage

```

npindex(bws, ...)

## S3 method for class 'formula'
npindex(bws,
        data = NULL,
        newdata = NULL,
        ...)

## S3 method for class 'call'

```

```

npindex(bws,
        ...)

## Default S3 method:
npindex(bws,
        txdat,
        tydat,
        ...)

## S3 method for class 'sibandwidth'
npindex(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        exdat,
        eydat,
        gradients = FALSE,
        residuals = FALSE,
        errors = FALSE,
        boot.num = 399,
        ...)

```

Arguments

<code>bws</code>	a bandwidth specification. This can be set as a <code>sibandwidth</code> object returned from an invocation of <code>npindexbw</code> , or as a vector of parameters (beta) with each element i corresponding to the coefficient for column i in <code>txdat</code> where the first element is normalized to 1, and a scalar bandwidth (h).
<code>gradients</code>	a logical value indicating that you want gradients and the asymptotic covariance matrix for beta computed and returned in the resulting <code>singleindex</code> object. Defaults to <code>FALSE</code> .
<code>residuals</code>	a logical value indicating that you want residuals computed and returned in the resulting <code>singleindex</code> object. Defaults to <code>FALSE</code> .
<code>errors</code>	a logical value indicating that you want (bootstrapped) standard errors for the conditional mean, gradients (when <code>gradients=TRUE</code> is set), and average gradients (when <code>gradients=TRUE</code> is set), computed and returned in the resulting <code>singleindex</code> object. Defaults to <code>FALSE</code> .
<code>boot.num</code>	an integer specifying the number of bootstrap replications to use when performing standard error calculations. Defaults to 399.
<code>...</code>	additional arguments supplied to specify the parameters to the <code>sibandwidth</code> S3 method, which is called during estimation.
<code>data</code>	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npindexbw</code> was called.
<code>newdata</code>	An optional data frame in which to look for evaluation data. If omitted, the training data are used.

txdat	a p -variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the training data used to compute the bandwidth object.
tydat	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of txdat. Defaults to the training data used to compute the bandwidth object.
exdat	a p -variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by txdat.
eydat	a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors.

Details

A matrix of gradients along with average derivatives are computed and returned if `gradients=TRUE` is used.

Value

npindex returns a `npsingleindex` object. The generic functions `fitted`, `residuals`, `coef`, `vcov`, `se`, `predict`, and `gradients`, extract (or generate) estimated values, residuals, coefficients, variance-covariance matrix, bootstrapped standard errors on estimates, predictions, and gradients, respectively, from the returned object. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

eval	evaluation points
mean	estimates of the regression function (conditional mean) at the evaluation points
beta	the model coefficients
betavcov	the asymptotic covariance matrix for the model coefficients
merr	standard errors of the regression function estimates
grad	estimates of the gradients at each evaluation point
gerr	standard errors of the gradient estimates
mean.grad	mean (average) gradient over the evaluation points
mean.gerr	bootstrapped standard error of the mean gradient estimates
R2	if method="ichimura", coefficient of determination
MSE	if method="ichimura", mean squared error
MAE	if method="ichimura", mean absolute error
MAPE	if method="ichimura", mean absolute percentage error
CORR	if method="ichimura", absolute value of Pearson's correlation coefficient
SIGN	if method="ichimura", fraction of observations where fitted and observed values agree in sign
confusion.matrix	if method="kleinspady", the confusion matrix or NA if outcomes are not available

CCR.overall	if method="kleinspady", the overall correct classification ratio, or NA if outcomes are not available
CCR.byoutcome	if method="kleinspady", a numeric vector containing the correct classification ratio by outcome, or NA if outcomes are not available
fit.mcfadden	if method="kleinspady", the McFadden-Puig-Kerschner performance measure or NA if outcomes are not available

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

`vcov` requires that `gradients=TRUE` be set.

Author(s)

Tristen Hayfield <hayfield@phys.ethz.ch>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.

Ichimura, H., (1993), "Semiparametric least squares (SLS) and weighted SLS estimation of single-index models," *Journal of Econometrics*, 58, 71-120.

Klein, R. W. and R. H. Spady (1993), "An efficient semiparametric estimator for binary response models," *Econometrica*, 61, 387-421.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

McFadden, D. and C. Puig and D. Kerschner (1977), "Determinants of the long-run demand for electricity," *Proceedings of the American Statistical Association (Business and Economics Section)*, 109-117.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): Generate a simple linear model then
# estimate it using a semiparametric single index specification and
# Ichimura's nonlinear least squares coefficients and bandwidth
# (default). Also compute the matrix of gradients and average derivative
# estimates.
```

```
set.seed(12345)
```

```
n <- 100
```

```
x1 <- runif(n, min=-1, max=1)
```

```
x2 <- runif(n, min=-1, max=1)
```

```
y <- x1 - x2 + rnorm(n)

# Note - this may take a minute or two depending on the speed of your
# computer. Note also that the first element of the vector beta is
# normalized to one for identification purposes, and that X must contain
# at least one continuous variable.

bw <- npindexbw(formula=y~x1+x2)

summary(bw)

model <- npindex(bws=bw, gradients=TRUE)

summary(model)

## Not run:

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Or you can visualize the input with plot.

plot(bw)

Sys.sleep(5)

# EXAMPLE 1 (INTERFACE=DATA FRAME): Generate a simple linear model then
# estimate it using a semiparametric single index specification and
# Ichimura's nonlinear least squares coefficients and bandwidth
# (default). Also compute the matrix of gradients and average derivative
# estimates.

set.seed(12345)

n <- 100

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)

y <- x1 - x2 + rnorm(n)

X <- cbind(x1, x2)

# Note - this may take a minute or two depending on the speed of your
# computer. Note also that the first element of the vector beta is
# normalized to one for identification purposes, and that X must contain
# at least one continuous variable.

bw <- npindexbw(xdat=X, ydat=y)

summary(bw)
```

```

model <- npindex(bws=bw, gradients=TRUE)

summary(model)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Or you can visualize the input with plot.

plot(bw)

Sys.sleep(5)

# EXAMPLE 2 (INTERFACE=FORMULA): Generate a simple binary outcome linear
# model then estimate it using a semiparametric single index
# specification and Klein and Spady's likelihood-based coefficients and
# bandwidth (default). Also compute the matrix of gradients and average
# derivative estimates.

n <- 100

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)

y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

# Note that the first element of the vector beta is normalized to one
# for identification purposes, and that X must contain at least one
# continuous variable.

bw <- npindexbw(formula=y~x1+x2, method="kleinspady")

summary(bw)

model <- npindex(bws=bw, gradients=TRUE)

# Note that, since the outcome is binary, we can assess model
# performance using methods appropriate for binary outcomes. We look at
# the confusion matrix, various classification ratios, and McFadden et
# al's measure of predictive performance.

summary(model)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 2 (INTERFACE=DATA FRAME): Generate a simple binary outcome
# linear model then estimate it using a semiparametric single index
# specification and Klein and Spady's likelihood-based coefficients and
# bandwidth (default). Also compute the matrix of gradients and average

```

```
# derivative estimates.

n <- 100

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)

y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

X <- cbind(x1, x2)

# Note that the first element of the vector beta is normalized to one
# for identification purposes, and that X must contain at least one
# continuous variable.

bw <- npindexbw(xdat=X, ydat=y, method="kleinspady")

summary(bw)

model <- npindex(bws=bw, gradients=TRUE)

# Note that, since the outcome is binary, we can assess model
# performance using methods appropriate for binary outcomes. We look at
# the confusion matrix, various classification ratios, and McFadden et
# al's measure of predictive performance.

summary(model)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 3 (INTERFACE=FORMULA): Replicate the DGP of Klein & Spady
# (1993) (see their description on page 405, pay careful attention to
# footnote 6 on page 405).

set.seed(123)

n <- 1000

# x1 is chi-squared having 3 df truncated at 6 standardized by
# subtracting 2.348 and dividing by 1.511

x <- rchisq(n, df=3)
x1 <- (ifelse(x < 6, x, 6) - 2.348)/1.511

# x2 is normal (0, 1) truncated at +- 2 divided by 0.8796

x <- rnorm(n)
x2 <- ifelse(abs(x) < 2, x, 2) / 0.8796

# y is 1 if y* > 0, 0 otherwise.
```

```

y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

# Compute the parameter vector and bandwidth. Note that the first
# element of the vector beta is normalized to one for identification
# purposes, and that X must contain at least one continuous variable.

bw <- npindexbw(formula=y~x1+x2, method="kleinshady")

# Next, create the evaluation data in order to generate a perspective
# plot

# Create an evaluation data matrix

x1.seq <- seq(min(x1), max(x1), length=50)
x2.seq <- seq(min(x2), max(x2), length=50)
X.eval <- expand.grid(x1=x1.seq, x2=x2.seq)

# Now evaluate the single index model on the evaluation data

fit <- fitted(npindex(exdat=X.eval,
                     eydat=rep(1, nrow(X.eval)),
                     bws=bw))

# Finally, coerce the fitted model into a matrix suitable for 3D
# plotting via persp()

fit.mat <- matrix(fit, 50, 50)

# Generate a perspective plot similar to Figure 2 b of Klein and Spady
# (1993)

persp(x1.seq,
      x2.seq,
      fit.mat,
      col="white",
      ticktype="detailed",
      expand=0.5,
      axes=FALSE,
      box=FALSE,
      main="Estimated Semiparametric Probability Perspective",
      theta=310,
      phi=25)

# EXAMPLE 3 (INTERFACE=DATA FRAME): Replicate the DGP of Klein & Spady
# (1993) (see their description on page 405, pay careful attention to
# footnote 6 on page 405).

set.seed(123)

n <- 1000

# x1 is chi-squared having 3 df truncated at 6 standardized by

```

```
# subtracting 2.348 and dividing by 1.511
x <- rchisq(n, df=3)
x1 <- (ifelse(x < 6, x, 6) - 2.348)/1.511

# x2 is normal (0, 1) truncated at +- 2 divided by 0.8796
x <- rnorm(n)
x2 <- ifelse(abs(x) < 2, x, 2) / 0.8796

# y is 1 if y* > 0, 0 otherwise.
y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

# Create the X matrix
X <- cbind(x1, x2)

# Compute the parameter vector and bandwidth. Note that the first
# element of the vector beta is normalized to one for identification
# purposes, and that X must contain at least one continuous variable.

bw <- npindexbw(xdat=X, ydat=y, method="kleinspady")

# Next, create the evaluation data in order to generate a perspective
# plot

# Create an evaluation data matrix
x1.seq <- seq(min(x1), max(x1), length=50)
x2.seq <- seq(min(x2), max(x2), length=50)
X.eval <- expand.grid(x1=x1.seq, x2=x2.seq)

# Now evaluate the single index model on the evaluation data
fit <- fitted(npindex(exdat=X.eval,
                     eydat=rep(1, nrow(X.eval)),
                     bws=bw))

# Finally, coerce the fitted model into a matrix suitable for 3D
# plotting via persp()

fit.mat <- matrix(fit, 50, 50)

# Generate a perspective plot similar to Figure 2 b of Klein and Spady
# (1993)

persp(x1.seq,
      x2.seq,
      fit.mat,
      col="white",
      ticktype="detailed",
```

```

    expand=0.5,
    axes=FALSE,
    box=FALSE,
    main="Estimated Semiparametric Probability Perspective",
    theta=310,
    phi=25)

## End(Not run)

```

npindexbw

Semiparametric Single Index Model Parameter and Bandwidth Selection

Description

npindexbw computes a npindexbw bandwidth specification using the model $Y = G(X\beta) + \epsilon$. The bandwidth object contains parameters for the single index model and the (scalar) bandwidth for the index function.

Usage

```

npindexbw(...)

## S3 method for class 'formula'
npindexbw(formula, data, subset, na.action, call, ...)

## S3 method for class 'NULL'
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          ...)

## Default S3 method:
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          random.seed,
          optim.method,
          optim.maxattempts,
          optim.reltol,
          optim.abstol,
          optim.maxit,
          only.optimize.beta,
          ...)

## S3 method for class 'sibandwidth'

```

```

npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          random.seed = 42,
          optim.method = c("Nelder-Mead", "BFGS", "CG"),
          optim.maxattempts = 10,
          optim.reltol = sqrt(.Machine$double.eps),
          optim.abstol = .Machine$double.eps,
          optim.maxit = 500,
          only.optimize.beta = FALSE,
          ...)

```

Arguments

formula	a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the <code>na.action</code> setting of options, and is <code>na.fail</code> if that is unset. The (recommended) default is <code>na.omit</code> .
call	the original function call. This is passed internally by <code>np</code> when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.
xdat	a p -variate data frame of explanatory data (training data) used to calculate the regression estimators.
ydat	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>xdat</code> .
bws	a bandwidth specification. This can be set as a <code>singleindexbandwidth</code> object returned from an invocation of <code>npindexbw</code> , or as a vector of parameters (beta) with each element i corresponding to the coefficient for column i in <code>xdat</code> where the first element is normalized to 1, and a scalar bandwidth (h). If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on.
method	the single index model method, one of either "ichimura" (default) (Ichimura (1993)) or "kleinspady" (Klein and Spady (1993)). Defaults to <code>ichimura</code> .
nmulti	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to <code>min(5, ncol(xdat))</code> .

<code>random.seed</code>	an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.
<code>bandwidth.compute</code>	a logical value which specifies whether to do a numerical search for bandwidths or not. If set to FALSE, a bandwidth object will be returned with bandwidths set to those specified in <code>bws</code> . Defaults to TRUE.
<code>optim.method</code>	<p>method used by <code>optim</code> for minimization of the objective function. See <code>?optim</code> for references. Defaults to "Nelder-Mead".</p> <p>the default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions.</p> <p>method "BFGS" is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized.</p> <p>method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak-Ribiere or Beale-Sorenson updates). Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.</p>
<code>optim.maxattempts</code>	maximum number of attempts taken trying to achieve successful convergence in <code>optim</code> . Defaults to 100.
<code>optim.abstol</code>	the absolute convergence tolerance used by <code>optim</code> . Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to <code>.Machine\$double.eps</code> .
<code>optim.reltol</code>	relative convergence tolerance used by <code>optim</code> . The algorithm stops if it is unable to reduce the value by a factor of <code>'reltol * (abs(val) + reltol)'</code> at a step. Defaults to <code>sqrt(.Machine\$double.eps)</code> , typically about <code>1e-8</code> .
<code>optim.maxit</code>	maximum number of iterations used by <code>optim</code> . Defaults to 500.
<code>only.optimize.beta</code>	signals the routine to only minimize the objective function with respect to beta
<code>...</code>	additional arguments supplied to specify the parameters to the <code>sibandwidth S3</code> method, which is called during the numerical search.

Details

We implement Ichimura's (1993) method via joint estimation of the bandwidth and coefficient vector using leave-one-out nonlinear least squares. We implement Klein and Spady's (1993) method maximizing the leave-one-out log likelihood function jointly with respect to the bandwidth and coefficient vector. Note that Klein and Spady's (1993) method is for *binary outcomes only*, while Ichimura's (1993) method can be applied for any outcome data type (i.e., continuous or discrete).

We impose the identification condition that the first element of the coefficient vector beta is equal to one, while identification also requires that the explanatory variables contain *at least one* continuous variable.

npindexbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the `xdat` and `ydat` parameters. Use of these two interfaces is **mutually exclusive**.

Note that, unlike most other bandwidth methods in the `np` package, this implementation uses the R `optim` nonlinear minimization routines and `npksum`. We have implemented multistarting and strongly encourage its use in practice. For exploratory purposes, you may wish to override the default search tolerances, say, setting `optim.reltol=.1` and conduct multistarting (the default is to restart `min(5, ncol(xdat))` times) as is done for a number of examples.

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form `dependent data ~ explanatory data`, where `dependent data` is a univariate response, and `explanatory data` is a series of variables specified by name, separated by the separation character `'+'`. For example `y1 ~ x1 + x2` specifies that the bandwidth object for the regression of response `y1` and semiparametric regressors `x1` and `x2` are to be estimated. See below for further examples.

Value

npindexbw returns a `sibandwidth` object, with the following components:

<code>bw</code>	bandwidth(s), scale factor(s) or nearest neighbours for the data, <code>xdat</code>
<code>beta</code>	coefficients of the model
<code>fval</code>	objective function value at minimum

If `bwtype` is set to `fixed`, an object containing a scalar bandwidth for the function $G(X\beta)$ and an estimate of the parameter vector β is returned.

If `bwtype` is set to `generalized_nn` or `adaptive_nn`, then instead the scalar k th nearest neighbor is returned.

The functions `coef`, `predict`, `summary`, and `plot` support objects of this class.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the i th observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting `optim.reltol=.1` and conduct multistarting (the default is to restart `min(5, ncol(xdat))` times). Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set `bws=bw` on subsequent calls to this routine where `bw` is the initial bandwidth object). A version of this package using the `Rmpi` wrapper is under development that

allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Ichimura, H., (1993), "Semiparametric least squares (SLS) and weighted SLS estimation of single-index models," *Journal of Econometrics*, 58, 71-120.
- Klein, R. W. and R. H. Spady (1993), "An efficient semiparametric estimator for binary response models," *Econometrica*, 61, 387-421.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): Generate a simple linear model then
# compute coefficients and the bandwidth using Ichimura's nonlinear
# least squares approach.

set.seed(12345)

n <- 100

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)

y <- x1 - x2 + rnorm(n)

# Note - this may take a minute or two depending on the speed of your
# computer. Note also that the first element of the vector beta is
# normalized to one for identification purposes, and that X must contain
# at least one continuous variable.

bw <- npindexbw(formula=y~x1+x2, method="ichimura")

summary(bw)

## Not run:

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)
```

```
# EXAMPLE 1 (INTERFACE=DATA FRAME): Generate a simple linear model then
# compute coefficients and the bandwidth using Ichimura's nonlinear
# least squares approach.

set.seed(12345)

n <- 100

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)

y <- x1 - x2 + rnorm(n)

X <- cbind(x1, x2)

# Note - this may take a minute or two depending on the speed of your
# computer. Note also that the first element of the vector beta is
# normalized to one for identification purposes, and that X must contain
# at least one continuous variable.

bw <- npindexbw(xdat=X, ydat=y, method="ichimura")

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 2 (INTERFACE=DATA FRAME): Generate a simple binary outcome
# model then compute coefficients and the bandwidth using Klein and
# Spady's likelihood-based approach.

n <- 100

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)

y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

# Note that the first element of the vector beta is normalized to one
# for identification purposes, and that X must contain at least one
# continuous variable.

bw <- npindexbw(formula=y~x1+x2, method="kleinspady")

summary(bw)

# EXAMPLE 2 (INTERFACE=DATA FRAME): Generate a simple binary outcome
# model then compute coefficients and the bandwidth using Klein and
# Spady's likelihood-based approach.

n <- 100
```

```

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)

y <- ifelse(x1 + x2 + rnorm(n) > 0, 1, 0)

X <- cbind(x1, x2)

# Note that the first element of the vector beta is normalized to one
# for identification purposes, and that X must contain at least one
# continuous variable.

bw <- npindexbw(xdat=X, ydat=y, method="kleinspady")

summary(bw)

## End(Not run)

```

npksum

Kernel Sums with Mixed Data Types

Description

npksum computes kernel sums on evaluation data, given a set of training data, data to be weighted (optional), and a bandwidth specification (any bandwidth object).

Usage

```

npksum(...)

## S3 method for class 'formula'
npksum(formula, data, newdata, subset, na.action, ...)

## Default S3 method:
npksum(bws,
       txdat = stop("training data 'txdat' missing"),
       tydat = NULL,
       exdat = NULL,
       weights = NULL,
       leave.one.out = FALSE,
       kernel.pow = 1.0,
       bandwidth.divide = FALSE,
       operator = c("normal", "convolution", "derivative", "integral"),
       smooth.coefficient = FALSE,
       ...)

## S3 method for class 'numeric'
npksum(bws,
       txdat = stop("training data 'txdat' missing"),

```

```

tydat,
exdat,
weights,
leave.one.out,
kernel.pow,
bandwidth.divide,
operator,
smooth.coefficient,
...)
```

Arguments

formula	a symbolic description of variables on which the sum is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
newdata	An optional data frame in which to look for evaluation data. If omitted, data is used.
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. The default is set by the <code>na.action</code> setting of options, and is <code>na.fail</code> if that is unset. The (recommended) default is <code>na.omit</code> .
...	additional arguments supplied to specify the parameters to the default S3 method, which is called during estimation.
txdat	a p -variate data frame of sample realizations (training data) used to compute the sum.
tydat	a numeric vector of data to be weighted. The i th kernel weight is applied to the i th element. Defaults to 1.
exdat	a p -variate data frame of sum evaluation points (if omitted, defaults to the training data itself).
bws	a bandwidth specification. This can be set as any suitable bandwidth object returned from a bandwidth-generating function, or a numeric vector.
weights	a n by q matrix of weights which can optionally be applied to <code>tydat</code> in the sum. See details.
leave.one.out	a logical value to specify whether or not to compute the leave one out sums. Will not work if <code>exdat</code> is specified. Defaults to FALSE.
kernel.pow	an integer specifying the power to which the kernels will be raised in the sum. Defaults to 1.
bandwidth.divide	a logical specifying whether or not to divide continuous kernel weights by their bandwidths. Use this with nearest-neighbor methods. Defaults to FALSE.
operator	a string specifying whether the normal, convolution, derivative, or integral kernels are to be used. Defaults to normal.

`smooth.coefficient`

a logical specifying whether or not to use certain optimisations if the smooth coefficient estimator is being computed. Currently does nothing. Defaults to FALSE.

Details

`npksum` exists so that you can create your own kernel objects with or without a variable to be weighted (default $Y = 1$). With the options available, you could create new nonparametric tests or even new kernel estimators. The convolution kernel option would allow you to create, say, the least squares cross-validation function for kernel density estimation.

`npksum` uses highly-optimized C code that strives to minimize its ‘memory footprint’, while there is low overhead involved when using repeated calls to this function (see, by way of illustration, the example below that conducts leave-one-out cross-validation for a local constant regression estimator via calls to the R function `nlm`, and compares this to the `npregbw` function).

`npksum` implements a variety of methods for computing multivariate kernel sums (p -variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the kernel sum at the point x . Generalized nearest-neighbor bandwidths change with the point at which the sum is computed, x . Fixed bandwidths are constant over the support of x .

`npksum` computes $\sum_{j=1}^n W_j' Y_j K(X_j)$, where A_j represents a row vector extracted from A . That is, it computes the kernel weighted sum of the outer product of the rows of W and Y . In the examples, the uses of such sums are illustrated.

`npksum` may be invoked *either* with a formula-like symbolic description of variables on which the sum is to be performed *or* through a simpler interface whereby data is passed directly to the function via the `txdat` and `tydat` parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame `txdat` (and also `exdat`) may be a mix of continuous (default), unordered discrete (to be specified in the data frame `txdat` using the `factor` command), and ordered discrete (to be specified in the data frame `txdat` using the `ordered` command). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `np` for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form `dependent data ~ explanatory data`, where `dependent data` and `explanatory data` are both series of variables specified by name, separated by the separation character `'+'`. For example, `y1 ~ x1 + x2` specifies that `y1` is to be kernel-weighted by `x1` and `x2` throughout the sum. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel (see `np` for details).

Value

npksum returns a npkernelsum object with the following components:

eval	the evaluation points
ksum	the sum at the evaluation points

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), “Multivariate binary discrimination by the kernel method,” *Biometrika*, 63, 413-420.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2003), “Nonparametric estimation of distributions with categorical and continuous data,” *Journal of Multivariate Analysis*, 86, 266-292.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.
- Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.
- Wang, M.C. and J. van Ryzin (1981), “A class of smooth estimators for discrete distributions,” *Biometrika*, 68, 301-309.

Examples

```
# EXAMPLE 1: For this example, we generate 100,000 observations from a
# N(0, 1) distribution, then evaluate the kernel density on a grid of 50
# equally spaced points using the npksum() function, then compare
# results with the (identical) npudens() function output

n <- 100000
x <- rnorm(n)
x.eval <- seq(-4, 4, length=50)

# Compute the bandwidth with the normal-reference rule-of-thumb

bw <- npudensbw(dat=x, bwmethod="normal-reference")

# Compute the univariate kernel density estimate using the 100,000
# training data points evaluated on the 50 evaluation data points,
```

```

# i.e., 1/nh times the sum of the kernel function evaluated at each of
# the 50 points

den.ksum <- npksum(txdat=x, exdat=x.eval, bws=bw$bw)$ksum/(n*bw$bw[1])

# Note that, alternatively (easier perhaps), you could use the
# bandwidth.divide=TRUE argument and drop the *bw$bw[1] term in the
# denominator, as in
# npksum(txdat=x, exdat=x.eval, bws=bw$bw, bandwidth.divide=TRUE)$ksum/n

# Compute the density directly with the npudens() function...

den <- fitted(npudens(tdat=x, edat=x.eval, bws=bw$bw))

# Plot the true DGP, the npksum()/(nh) estimate and (identical)
# npudens() estimate

plot(x.eval, dnorm(x.eval), xlab="X", ylab="Density", type="l")
points(x.eval, den.ksum, col="blue")
points(x.eval, den, col="red", cex=0.2)
legend(1, .4,
       c("DGP", "npksum()",
         "npudens()"),
       col=c("black", "blue", "red"),
       lty=c(1, 1, 1))

## Not run:

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 2: For this example, we first obtain residuals from a parametric
# regression model, then compute a vector of leave-one-out kernel
# weighted sums of squared residuals where the kernel function is raised
# to the power 2. We use a normal-reference rule-of-thumb obtained by
# using the scaling factor 1.06 and bwscaling=TRUE. Note that this
# returns a vector of kernel weighted sums, one for each element of the
# error vector. Note also that you can specify the bandwidth type,
# kernel function, kernel order etc.

data("cps71")
attach(cps71)

error <- residuals(lm(logwage~age+I(age^2)))

ksum <- npksum(txdat=age,
               tydat=error^2,
               bws=1.06,
               leave.one.out=TRUE,
               bwscaling=TRUE,
               kernel.pow=2)

```

```
ksum

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Note - npksum also can inherit properties from a bandwidth
# object. To again specify a value for the bandwidth, you might do so in
# the bandwidth object itself then pass this into npksum (e.g.,
# below I use a regression bandwidth object but you could use, say, a
# density bandwidth object instead).

bw <- npregbw(xdat=age, ydat=logwage, bws=c(1.06),
              bwscaling=TRUE,
              bandwidth.compute=FALSE)

ksum <- npksum(txdat=age,
              tydat=error^2,
              bws=bw$bw,
              leave.one.out=TRUE,
              kernel.pow=2)

# Obviously, if we wanted the sum of these weighted kernel sums then,
# trivially,

sum(ksum$ksum)

detach(cps71)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Note that weighted leave-one-out sums of squared residuals are used to
# construct consistent model specification tests. In fact, the
# npcctest() routine in this package is constructed purely from calls
# to npksum(). You can type npcctest to see the npcctest()
# code and also examine some examples of the many uses of
# npksum().

# EXAMPLE 3: For this example, we conduct local-constant (i.e.,
# Nadaraya-Watson) kernel regression. We shall use cross-validated
# bandwidths using npregbw() by way of example. Note we extract
# the kernel sum from npksum() via the '$ksum' argument in both
# the numerator and denominator.

data("cps71")
attach(cps71)

bw <- npregbw(xdat=age, ydat=logwage)

fit.lc <- npksum(txdat=age, tydat=logwage, bws=bw$bw)$ksum/
         npksum(txdat=age, bws=bw$bw)$ksum
```

```
plot(age, logwage, xlab="Age", ylab="log(wage)")
lines(age, fit.lc)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# If you wished to compute the kernel sum for a set of evaluation points,
# you first generate the set of points then feed this to npksum,
# e.g., for the set (20, 30, ..., 60) we would use

age.seq <- seq(20, 60, 10)

fit.lc <- npksum(txdat=age, exdat=age.seq, tydat=logwage, bws=bw$bw)$ksum/
          npksum(txdat=age, exdat=age.seq, bws=bw$bw)$ksum

# Note that now fit.lc contains the 5 values of the local constant
# estimator corresponding to age.seq...

fit.lc

detach(cps71)

# EXAMPLE 4: For this example, we conduct least-squares cross-validation
# for the local-constant regression estimator. We first write an R
# function 'ss' that computes the leave-one-out sum of squares using the
# npksum() function, and then feed this function, along with
# random starting values for the bandwidth vector, to the nlm() routine
# in R (nlm = Non-Linear Minimization). Finally, we compare results with
# the function npregbw() that is written solely in C and calls
# a tightly coupled C-level search routine. Note that one could make
# repeated calls to nlm() using different starting values for h (highly
# recommended in general).

# Increase the number of digits printed out by default in R and avoid
# using scientific notation for this example (we wish to compare
# objective function minima)

options(scipen=100, digits=12)

# Generate 100 observations from a simple DGP where one explanatory
# variable is irrelevant.

n <- 100

x1 <- runif(n)
x2 <- rnorm(n)
x3 <- runif(n)

txdat <- data.frame(x1, x2, x3)

# Note - x3 is irrelevant
```

```

tydat <- x1 + sin(x2) + rnorm(n)

# Write an R function that returns the average leave-one-out sum of
# squared residuals for the local constant estimator based upon
# npksum(). This function accepts one argument and presumes that
# txdat and tydat have been defined already.

ss <- function(h) {

# Test for valid (non-negative) bandwidths - return infinite penalty
# when this occurs

  if(min(h)<=0) {

    return(.Machine$double.xmax)

  } else {

    mean <- npksum(txdat,
                  tydat,
                  leave.one.out=TRUE,
                  bandwidth.divide=TRUE,
                  bws=h)$ksum/
    npksum(txdat,
           leave.one.out=TRUE,
           bandwidth.divide=TRUE,
           bws=h)$ksum

    return(sum((tydat-mean)^2)/length(tydat))

  }

}

# Now pass this function to R's nlm() routine along with random starting
# values and place results in 'nlm.return'.

nlm.return <- nlm(ss, runif(length(txdat)))

bw <- npregbw(xdat=txdat, ydat=tydat)

# Bandwidths from nlm()

nlm.return$estimate

# Bandwidths from npregbw()

bw$bw

# Function value (minimum) from nlm()

nlm.return$minimum

```

```

# Function value (minimum) from npregbw()

bw$fval

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 5: For this example, we use npksum() to plot the kernel
# function itself. Our 'training data' is the singleton, 0, and our
# evaluation data a grid in [-4,4], while we use a bandwidth of 1. By
# way of example we plot a sixth order Gaussian kernel (note that this
# kernel function can assume negative values)

x <- 0
x.eval <- seq(-4,4,length=500)

kernel <- npksum(txdat=x,exdat=x.eval,bws=1,
                 bandwidth.divide=TRUE,
                 ckertype="gaussian",
                 ckerorder=6)$ksum

plot(x.eval,kernel,type="l",xlab="X",ylab="Kernel Function")
abline(0,0)

## End(Not run)

```

npplot

General Purpose Plotting of Nonparametric Objects

Description

npplot is invoked by `plot` and generates plots of nonparametric statistical objects such as regressions, quantile regressions, partially linear regressions, single-index models, densities and distributions, given training data and a bandwidth object.

Usage

```

npplot(bws = stop("'bws' has not been set"), ..., random.seed = 42)

## S3 method for class 'bandwidth'
npplot(bws,
       xdat,
       data = NULL,
       xq = 0.5,
       xtrim = 0.0,
       neval = 50,
       common.scale = TRUE,

```

```

perspective = TRUE,
main = "",
theta = 0.0,
phi = 10.0,
view = c("rotate", "fixed"),
type = "l",
ylim = NULL,
cdf = FALSE,
plot.behavior = c("plot", "plot-data", "data"),
plot.errors.method = c("none", "bootstrap", "asymptotic"),
plot.errors.boot.method = c("inid", "fixed", "geom"),
plot.errors.boot.blocklen = NULL,
plot.errors.boot.num = 399,
plot.errors.center = c("estimate", "bias-corrected"),
plot.errors.type = c("standard", "quantiles"),
plot.errors.quantiles = c(0.025, 0.975),
plot.errors.style = c("bar", "band"),
plot.errors.bar = c("|", "I"),
plot.errors.bar.num = min(neval, 25),
plot.bxp = FALSE,
plot.bxp.out = TRUE,
...,
random.seed)

## S3 method for class 'conbandwidth'
npplot(bws,
      xdat,
      ydat,
      data = NULL,
      xq = 0.5,
      yq = 0.5,
      xtrim = 0.0,
      ytrim = 0.0,
      neval = 50,
      quantreg = FALSE,
      gradients = FALSE,
      cdf = FALSE,
      common.scale = TRUE,
      perspective = TRUE,
      main = "",
      theta = 0.0,
      phi = 10.0,
      tau = 0.5,
      view = c("rotate", "fixed"),
      type = "l",
      ylim = NULL,
      plot.behavior = c("plot", "plot-data", "data"),
      plot.errors.method = c("none", "bootstrap", "asymptotic"),

```

```

plot.errors.boot.method = c("inid", "fixed", "geom"),
plot.errors.boot.blocklen = NULL,
plot.errors.boot.num = 399,
plot.errors.center = c("estimate","bias-corrected"),
plot.errors.type = c("standard","quantiles"),
plot.errors.quantiles = c(0.025,0.975),
plot.errors.style = c("bar","band"),
plot.errors.bar = c("|","I"),
plot.errors.bar.num = min(neval,25),
plot.bxp = FALSE,
plot.bxp.out = TRUE,
...,
random.seed)

## S3 method for class 'plbandwidth'
npplot(bws,
      xdat,
      ydat,
      zdat,
      data = NULL,
      xq = 0.5,
      zq = 0.5,
      xtrim = 0.0,
      ztrim = 0.0,
      neval = 50,
      common.scale = TRUE,
      perspective = TRUE,
      gradients = FALSE,
      main = "",
      theta = 0.0,
      phi = 10.0,
      view = c("rotate","fixed"),
      type = "l",
      ylim = NULL,
      plot.behavior = c("plot","plot-data","data"),
      plot.errors.method = c("none","bootstrap","asymptotic"),
      plot.errors.boot.method = c("inid", "fixed", "geom"),
      plot.errors.boot.blocklen = NULL,
      plot.errors.boot.num = 399,
      plot.errors.center = c("estimate","bias-corrected"),
      plot.errors.type = c("standard","quantiles"),
      plot.errors.quantiles = c(0.025,0.975),
      plot.errors.style = c("bar","band"),
      plot.errors.bar = c("|","I"),
      plot.errors.bar.num = min(neval,25),
      plot.bxp = FALSE,
      plot.bxp.out = TRUE,
      ...,

```

```
        random.seed)

## S3 method for class 'rbandwidth'
npplot(bws,
       xdat,
       ydat,
       data = NULL,
       xq = 0.5,
       xtrim = 0.0,
       neval = 50,
       common.scale = TRUE,
       perspective = TRUE,
       gradients = FALSE,
       main = "",
       theta = 0.0,
       phi = 10.0,
       view = c("rotate", "fixed"),
       type = "l",
       ylim = NULL,
       plot.behavior = c("plot", "plot-data", "data"),
       plot.errors.method = c("none", "bootstrap", "asymptotic"),
       plot.errors.boot.num = 399,
       plot.errors.boot.method = c("inid", "fixed", "geom"),
       plot.errors.boot.blocklen = NULL,
       plot.errors.center = c("estimate", "bias-corrected"),
       plot.errors.type = c("standard", "quantiles"),
       plot.errors.quantiles = c(0.025, 0.975),
       plot.errors.style = c("bar", "band"),
       plot.errors.bar = c("|", "I"),
       plot.errors.bar.num = min(neval, 25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       ...,
       random.seed)

## S3 method for class 'scbandwidth'
npplot(bws,
       xdat,
       ydat,
       zdat = NULL,
       data = NULL,
       xq = 0.5,
       zq = 0.5,
       xtrim = 0.0,
       ztrim = 0.0,
       neval = 50,
       common.scale = TRUE,
       perspective = TRUE,
```

```

    gradients = FALSE,
    main = "",
    theta = 0.0,
    phi = 10.0,
    view = c("rotate", "fixed"),
    type = "l",
    ylim = NULL,
    plot.behavior = c("plot", "plot-data", "data"),
    plot.errors.method = c("none", "bootstrap", "asymptotic"),
    plot.errors.boot.num = 399,
    plot.errors.boot.method = c("inid", "fixed", "geom"),
    plot.errors.boot.blocklen = NULL,
    plot.errors.center = c("estimate", "bias-corrected"),
    plot.errors.type = c("standard", "quantiles"),
    plot.errors.quantiles = c(0.025, 0.975),
    plot.errors.style = c("bar", "band"),
    plot.errors.bar = c("|", "I"),
    plot.errors.bar.num = min(neval, 25),
    plot.bxp = FALSE,
    plot.bxp.out = TRUE,
    ...,
    random.seed)

## S3 method for class 'sibandwidth'
npplot(bws,
      xdat,
      ydat,
      data = NULL,
      common.scale = TRUE,
      gradients = FALSE,
      main = "",
      type = "l",
      ylim = NULL,
      plot.behavior = c("plot", "plot-data", "data"),
      plot.errors.method = c("none", "bootstrap", "asymptotic"),
      plot.errors.boot.num = 399,
      plot.errors.boot.method = c("inid", "fixed", "geom"),
      plot.errors.boot.blocklen = NULL,
      plot.errors.center = c("estimate", "bias-corrected"),
      plot.errors.type = c("standard", "quantiles"),
      plot.errors.quantiles = c(0.025, 0.975),
      plot.errors.style = c("bar", "band"),
      plot.errors.bar = c("|", "I"),
      plot.errors.bar.num = NULL,
      ...,
      random.seed)

```

Arguments

bws	a bandwidth specification. This should be a bandwidth object returned from an invocation of <code>npudensbw</code> , <code>npcdensbw</code> , <code>npregbw</code> , <code>npplregbw</code> , <code>npindexbw</code> , or <code>npscoefbw</code> .
...	additional arguments supplied to control various aspects of plotting, depending on the type of object to be plotted, detailed below.
data	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in data, the variables are taken from <code>environment(bws)</code> , typically the environment where the bandwidth object was generated.
xdat	a p -variate data frame of sample realizations (training data).
ydat	a q -variate data frame of sample realizations (training data). In a regression or conditional density context, this is the dependent data.
zdat	a p -variate data frame of sample realizations (training data).
xq	a numeric p -vector of quantiles. Each element i of <code>xq</code> corresponds to the i th column of <code>txdat</code> . Defaults to the median (0.5). See details.
yq	a numeric q -vector of quantiles. Each element i of <code>yq</code> corresponds to the i th column of <code>tydat</code> . Only to be specified in a conditional density context. Defaults to the median (0.5). See details.
zq	a numeric q -vector of quantiles. Each element i of <code>zq</code> corresponds to the i th column of <code>tzdat</code> . Only to be specified in a semiparametric model context. Defaults to the median (0.5). See details.
xtrim	a numeric p -vector of quantiles. Each element i of <code>xtrim</code> corresponds to the i th column of <code>txdat</code> . Defaults to 0.0. See details.
ytrim	a numeric q -vector of quantiles. Each element i of <code>ytrim</code> corresponds to the i th column of <code>tydat</code> . Defaults to 0.0. See details.
ztrim	a numeric q -vector of quantiles. Each element i of <code>ztrim</code> corresponds to the i th column of <code>tzdat</code> . Defaults to 0.0. See details.
neval	an integer specifying the number of evaluation points. Only applies to continuous variables however, as discrete variables will be evaluated once at each category. Defaults to 50.
common.scale	a logical value specifying whether or not all graphs are to be plotted on a common scale. Defaults to TRUE.
perspective	a logical value specifying whether a perspective plot should be displayed (if possible). Defaults to TRUE.
quantreg	a logical value specifying whether a quantile regression should be plotted (conbandwidth objects only). See related parameter <code>'tau'</code> . Defaults to FALSE.
gradients	a logical value specifying whether gradients should be plotted (if possible). Defaults to FALSE.
cdf	a logical value specifying whether cumulative distributions should be plotted instead of densities (when appropriate). Defaults to FALSE.
main	the title, see <code>title</code>

<code>type</code>	character indicating the type of plotting; actually any of the types as in <code>plot.default</code>
<code>ylim</code>	a two-element numeric vector of the minimum and maximum y plotting limits. Defaults to NULL.
<code>theta</code>	a numeric value specifying the starting azimuthal angle of the perspective plot. Defaults to 0.0.
<code>phi</code>	a numeric value specifying the starting zenith angle of the perspective plot. Defaults to 10.0.
<code>tau</code>	a numeric value specifying the τ th quantile is desired when plotting quantile regressions.
<code>view</code>	a character string used to specify the viewing mode of the perspective plot. Can be set as <code>rotate</code> or <code>fixed</code> . Defaults to <code>rotate</code> .
<code>plot.behavior</code>	a character string used to specify the net behavior of <code>npplot</code> . Can be set as <code>plot</code> , <code>plot-data</code> or <code>data</code> . Defaults to <code>plot</code> . See <code>value</code> .
<code>plot.errors.method</code>	a character string used to specify the method to calculate errors. Can be set as <code>none</code> , <code>bootstrap</code> , or <code>asymptotic</code> . Defaults to <code>none</code> .
<code>plot.errors.boot.method</code>	a character string used to specify the bootstrap method. Can be set as <code>inid</code> , <code>fixed</code> , or <code>geom</code> (see below for details). Defaults to <code>inid</code> .
<code>plot.errors.boot.blocklen</code>	an integer used to specify the block length b for the <code>fixed</code> or <code>geom</code> bootstrap (see below for details).
<code>plot.errors.boot.num</code>	an integer used to specify the number of bootstrap samples to use for the calculation of errors. Defaults to 399.
<code>plot.errors.center</code>	a character string used to specify where to center the errors on the plot(s). Can be set as <code>estimate</code> or <code>bias-corrected</code> . Defaults to <code>estimate</code> .
<code>plot.errors.type</code>	a character string used to specify the type of error to calculate. Can be set as <code>standard</code> or <code>quantiles</code> . Defaults to <code>standard</code> .
<code>plot.errors.quantiles</code>	a numeric vector specifying the quantiles of the statistic to calculate for the purpose of error plotting. Defaults to <code>c(0.025, 0.975)</code> .
<code>plot.errors.style</code>	a character string used to specify the style of error plotting. Can be set as <code>bar</code> or <code>band</code> . Defaults to <code>bar</code> . Bands are not drawn for discrete variables.
<code>plot.errors.bar</code>	a character string used to specify the error bar shape. Can be set as <code> </code> (vertical bar character) for a dashed vertical bar, or as <code>I</code> for an 'I' shaped error bar with horizontal bounding bars. Defaults to <code> </code> .
<code>plot.errors.bar.num</code>	an integer specifying the number of error bars to plot. Defaults to <code>min(neval, 25)</code> .
<code>plot.bxp</code>	a logical value specifying whether boxplots should be produced when appropriate. Defaults to <code>FALSE</code> .

<code>plot.bxp.out</code>	a logical value specifying whether outliers should be plotted on boxplots. Defaults to TRUE.
<code>random.seed</code>	an integer used to seed R's random number generator. This ensures replicability of the bootstrapped errors. Defaults to 42.

Details

`npplot` is a general purpose plotting routine for visually exploring objects generated by the `np` library, such as regressions, quantile regressions, partially linear regressions, single-index models, densities and distributions. There is no need to call `npplot` directly as it is automatically invoked when `plot` is used with an object generated by the `np` package.

Visualizing one and two dimensional datasets is a straightforward process. The default behavior of `npplot` is to generate a standard 2D plot to visualize univariate data, and a perspective plot for bivariate data. When visualizing higher dimensional data, `npplot` resorts to plotting a series of 1D slices of the data. For a slice along dimension i , all other variables at indices $j \neq i$ are held constant at the quantiles specified in the j th element of `xq`. The default is the median.

The slice itself is evaluated on a uniformly spaced sequence of *neval* points. The interval of evaluation is determined by the training data. The default behavior is to evaluate from `min(txdat[, i])` to `max(txdat[, i])`. The `xtrim` variable allows for control over this behavior. When `xtrim` is set, data is evaluated from the `xtrim[i]`th quantile of `txdat[, i]` to the `1.0-xtrim[i]`th quantile of `txdat[, i]`.

Furthermore, `xtrim` can be set to a negative value in which case it will expand the limits of the evaluation interval beyond the support of the training data, by measuring the distance between `min(txdat[, i])` and the `xtrim[i]`th quantile of `txdat[, i]`, and extending the support by that distance on the lower limit of the interval. `npplot` uses an analogous procedure to extend the upper limit of the interval.

Bootstrap resampling is conducted pairwise on (y, X, Z) (i.e., by resampling from rows of the (y, X) data or (y, X, Z) data where appropriate). `inid` admits general heteroskedasticity of unknown form, though it does not allow for dependence. `fixed` conducts Kunsch's (1988) block bootstrap for dependent data, while `geom` conducts Politis and Romano's (1994) stationary bootstrap.

For consistency of the block and stationary bootstrap, the (mean) block length b should grow with the sample size n at an appropriate rate. If b is not given, then a default growth rate of $const \times n^{1/3}$ is used. This rate is "optimal" under certain conditions (see Politis and Romano (1994) for more details). However, in general the growth rate depends on the specific properties of the DGP. A default value for *const* (3.15) has been determined by a Monte Carlo simulation using a Gaussian AR(1) process (AR(1)-parameter of 0.5, 500 observations). *const* has been chosen such that the mean square error for the bootstrap estimate of the variance of the empirical mean is minimized.

Value

Setting `plot.behavior` will instruct `npplot` what data to return. Option summary:

`plot`: instruct `npplot` to just plot the data and return NULL

`plot-data`: instruct `npplot` to plot the data and return the data used to generate the plots. The data will be a list of objects of the appropriate type, with one object per plot. For example, invoking `npplot` on 3D density data will have it return a list of three `npdensity` objects. If biases were

calculated, they are stored in a component named `bias`
`data`: instruct `npplot` to generate data only and no plots

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, 99, 1015-1026.
- Kunsch, H.R. (1989), "The jackknife and the bootstrap for general stationary observations," *The Annals of Statistics*, 17, 1217-1241.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Politis, D.N. and J.P. Romano (1994), "The stationary bootstrap," *Journal of the American Statistical Association*, 89, 1303-1313.
- Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.
- Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

Examples

```
## Not run:
# EXAMPLE 1: For this example, we load Giovanni Baiocchi's Italian GDP
# panel (see Italy for details), then create a data frame in which year
# is an ordered factor, GDP is continuous, compute bandwidths using
# likelihood cross-validation, then create a grid of data on which the
# density will be evaluated for plotting purposes

data("Italy")
attach(Italy)

data <- data.frame(ordered(year), gdp)

# Compute bandwidths using likelihood cross-validation (default). Note
# that this may take a minute or two depending on the speed of your
```

```
# computer...

bw <- npudensbw(dat=data)

# You can always do things manually, as the following example demonstrates

# Create an evaluation data matrix

year.seq <- sort(unique(year))
gdp.seq <- seq(1,36,length=50)
data.eval <- expand.grid(year=year.seq,gdp=gdp.seq)

# Generate the estimated density computed for the evaluation data

fhat <- fitted(npudens(tdat = data, edat = data.eval, bws=bw))

# Coerce the data into a matrix for plotting with persp()

f <- matrix(fhat, length(unique(year)), 50)

# Next, create a 3D perspective plot of the PDF f

persp(as.integer(levels(year.seq)), gdp.seq, f, col="lightblue",
      ticktype="detailed", ylab="GDP", xlab="Year", zlab="Density",
      theta=300, phi=50)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# However, npplot simply streamlines this process and aids in the
# visualization process (<ctrl>-C will interrupt on *NIX systems, <esc>
# will interrupt on MS Windows systems).

plot(bw)

# npplot also streamlines construction of variability bounds (<ctrl>-C
# will interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems)

plot(bw, plot.errors.method = "asymptotic")

# EXAMPLE 2: For this example, we simulate multivariate data, and plot the
# partial regression surfaces for a locally linear estimator and its
# derivatives.

set.seed(123)

n <- 100

x1 <- runif(n)
x2 <- runif(n)
x3 <- runif(n)
```

```

x4 <- rbinom(n, 2, .3)

y <- 1 + x1 + x2 + x3 + x4 + rnorm(n)

X <- data.frame(x1, x2, x3, ordered(x4))

bw <- npregbw(xdat=X, ydat=y, regtype="l1", bwmethod="cv.aic")

plot(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Now plot the gradients...

plot(bw, gradients=TRUE)

# Plot the partial regression surfaces with bias-corrected bootstrapped
# nonparametric confidence intervals... this may take a minute or two
# depending on the speed of your computer as the bootstrapping must be
# completed prior to results being displayed...

plot(bw,
      plot.errors.method="bootstrap",
      plot.errors.center="bias-corrected",
      plot.errors.type="quantiles")

# Note - if you wished to create, say, a postscript graph for inclusion
# in, say, a latex document, use R's 'postscript' command to switch to
# the postscript device (turn off the device once completed). The
# following will create a disk file 'graph.ps' that can be pulled into,
# say, a latex document via \includegraphics[width=5in, height=5in,
# angle=270]{graph.ps}

# Note - make sure to include the graphicx package in your latex
# document via adding \usepackage{graphicx} in your latex file. Also,
# you might want to use larger fonts, which can be achieved by adding the
# pointsize= argument, e.g., postscript(file="graph.ps", pointsize=20)

postscript(file="graph.ps")
plot(bw)
dev.off()

# The following latex file compiled in the same directory as graph.ps
# ought to work (remove the #s and place in a file names, e.g.,
# test.tex).
# \documentclass[]{article}
# \usepackage{graphicx}
# \begin{document}
# \begin{figure}[!ht]
# \includegraphics[width=5in, height=5in, angle=270]{graph.ps}
# \caption{Local linear partial regression surfaces.}

```

```

# \end{figure}
# \end{document}

# EXAMPLE 3: This example demonstrates how to retrieve plotting data from
# npplot(). When npplot() is called with the arguments
# 'plot.behavior="plot-data"' (or "data"), it returns plotting objects
# named r1, r2, and so on (rg1, rg2, and so on when 'gradients=TRUE' is
# set). Each plotting object's index (1,2,...) corresponds to the index
# of the explanatory data data frame xdat (and zdat if appropriate).

# Take the cps71 data by way of example. In this case, there is only one
# object returned by default, 'r1', since xdat is univariate.

data("cps71")
attach(cps71)

# Compute bandwidths for local linear regression using cv.aic...

bw <- npregbw(xdat=age,ydat=logwage,regtype="ll",bwmethod="cv.aic")

# Generate the plot and return plotting data, and store output in
# 'plot.out' (NOTE: the call to 'plot.behavior' is necessary).

plot.out <- plot(bw,
                 perspective=FALSE,
                 plot.errors.method="bootstrap",
                 plot.errors.boot.num=25,
                 plot.behavior="plot-data")

# Now grab the r1 object that npplot plotted on the screen, and take
# what you need. First, take the output, lower error bound and upper
# error bound...

logwage.eval <- fitted(plot.out$r1)
logwage.se <- se(plot.out$r1)
logwage.lower.ci <- logwage.eval + logwage.se[,1]
logwage.upper.ci <- logwage.eval + logwage.se[,2]

# Next grab the x data evaluation data. xdat is a data.frame(), so we
# need to coerce it into a vector (take the 'first column' of data frame
# even though there is only one column)

age.eval <- plot.out$r1$eval[,1]

# Now we could plot this if we wished, or direct it to whatever end use
# we envisioned. We plot the results using R's plot() routines...

plot(age,logwage,cex=0.2,xlab="Age",ylab="log(Wage)")
lines(age.eval,logwage.eval)
lines(age.eval,logwage.lower.ci,lty=3)
lines(age.eval,logwage.upper.ci,lty=3)

```

```

# If you wanted npplot() data for gradients, you would use the argument
# 'gradients=TRUE' in the call to npplot() as the following
# demonstrates...

plot.out <- plot(bw,
                perspective=FALSE,
                plot.errors.method="bootstrap",
                plot.errors.boot.num=25,
                plot.behavior="plot-data",
                gradients=TRUE)

# Now grab object that npplot() plotted on the screen. First, take the
# output, lower error bound and upper error bound... note that gradients
# are stored in objects rg1, rg2 etc.

grad.eval <- gradients(plot.out$rg1)
grad.se <- gradients(plot.out$rg1, errors = TRUE)
grad.lower.ci <- grad.eval + grad.se[,1]
grad.upper.ci <- grad.eval + grad.se[,2]

# Next grab the x evaluation data. xdat is a data.frame(), so we need to
# coerce it into a vector (take 'first column' of data frame even though
# there is only one column)

age.eval <- plot.out$rg1$eval[,1]

# We plot the results using R's plot() routines...

plot(age.eval, grad.eval, cex=0.2,
      ylim=c(min(grad.lower.ci), max(grad.upper.ci)),
      xlab="Age", ylab="d log(Wage)/d Age", type="l")
lines(age.eval, grad.lower.ci, lty=3)
lines(age.eval, grad.upper.ci, lty=3)

detach(cps71)

## End(Not run)

```

Description

npplreg computes a partially linear kernel regression estimate of a one (1) dimensional dependent variable on $p + q$ -variate explanatory data, using the model $Y = X\beta + \Theta(Z) + \epsilon$ given a set of estimation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification, which can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```

npplreg(bws, ...)

## S3 method for class 'formula'
npplreg(bws, data = NULL, newdata = NULL, ...)

## S3 method for class 'call'
npplreg(bws, ...)

## S3 method for class 'plbandwidth'
npplreg(bws,
        txdat = stop("training data txdat missing"),
        tydat = stop("training data tydat missing"),
        tzdat = stop("training data tzdat missing"),
        exdat,
        eydat,
        ezdat,
        residuals = FALSE,
        ...)

```

Arguments

<code>bws</code>	a bandwidth specification. This can be set as a <code>plbandwidth</code> object returned from an invocation of <code>npplregbw</code> , or as a matrix of bandwidths, each row is a set of bandwidths for Z , with a column for each variable Z_i . In the first row are the bandwidths for the regression of Y on Z , the following rows contain the bandwidths for the regressions of the columns of X on Z . If specified as a matrix additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, training data, and so on.
<code>...</code>	additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on. To do this, you may specify any of <code>regtype</code> , <code>bwmethod</code> , <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> , as described in npplregbw .
<code>data</code>	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in data, the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npplregbw</code> was called.
<code>newdata</code>	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
<code>txdat</code>	a p -variate data frame of explanatory data (training data), corresponding to X in the model equation, whose linear relationship with the dependent data Y is posited. Defaults to the training data used to compute the bandwidth object.
<code>tydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>txdat</code> . Defaults to the training data used to compute the bandwidth object.
<code>tzdat</code>	a q -variate data frame of explanatory data (training data), corresponding to Z in the model equation, whose relationship to the dependent variable is unspecified

	(nonparametric). Defaults to the training data used to compute the bandwidth object.
exdat	a p -variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by txdat.
eydat	a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors. By default, evaluation takes place on the data provided by tydat.
ezdat	a q -variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by tzdat.
residuals	a logical value indicating that you want residuals computed and returned in the resulting plregression object. Defaults to FALSE.

Details

npplreg uses a combination of OLS and nonparametric regression to estimate the parameter β in the model $Y = X\beta + \Theta(Z) + \epsilon$.

npplreg implements a variety of methods for nonparametric regression on multivariate (q -variate) explanatory data defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

Data contained in the data frame tzdat may be a mix of continuous (default), unordered discrete (to be specified in the data frame tzdat using [factor](#)), and ordered discrete (to be specified in the data frame tzdat using [ordered](#)). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see [np](#) for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npplreg returns a plregression object. The generic accessor functions [coef](#), [fitted](#), [residuals](#), [predict](#), and [vcov](#), extract (or estimate) coefficients, estimated values, residuals, predictions, and variance-covariance matrices, respectively, from the returned object. Furthermore, the functions [summary](#) and [plot](#) support objects of this type. The returned object has the following components:

evalx	evaluation points
evalz	evaluation points
mean	estimation of the regression, or conditional mean, at the evaluation points
xcoef	coefficient(s) corresponding to the components β_i in the model

xcoeferr	standard errors of the coefficients
xcoefvcov	covariance matrix of the coefficients
bw	the bandwidths, stored as a plbandwidth object
resid	if residuals = TRUE, in-sample or out-of-sample residuals where appropriate (or possible)
R2	coefficient of determination
MSE	mean squared error
MAE	mean absolute error
MAPE	mean absolute percentage error
CORR	absolute value of Pearson's correlation coefficient
SIGN	fraction of observations where fitted and observed values agree in sign

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," *Statistica Sinica*, 14, 485-512.
- Racine, J.S. and L. Liu (2006), "A partially linear kernel estimator for categorical data," manuscript.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," *Journal of Econometrics*, 119, 99-130.
- Robinson, P.M. (1988), "Root-n-consistent semiparametric regression," *Econometrica*, 56, 931-954.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also

[npregbw](#), [npreg](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we simulate an
# example for a partially linear model, and compare the coefficient
# estimates from the partially linear model with those from a correctly
# specified parametric model...

set.seed(42)

n <- 250
x1 <- rnorm(n)
x2 <- rbinom(n, 1, .5)

z1 <- rbinom(n, 1, .5)
z2 <- rnorm(n)

y <- 1 + x1 + x2 + z1 + sin(z2) + rnorm(n)

# First, compute data-driven bandwidths. This may take a few minutes
# depending on the speed of your computer...

bw <- npplregbw(formula=y~x1+factor(x2)|factor(z1)+z2)

# Next, compute the partially linear fit

pl <- npplreg(bws=bw)

# Print a summary of the model...

summary(pl)

## Not run:

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Retrieve the coefficient estimates and their standard errors...

coef(pl)
coef(pl, errors = TRUE)

# Compare the partially linear results to those from a correctly
# specified model's coefficients for x1 and x2

ols <- lm(y~x1+factor(x2)+factor(z1)+I(sin(z2)))

# The intercept is coef()[1], and those for x1 and x2 are coef()[2] and
# coef()[3]. The standard errors are the square root of the diagonal of
# the variance-covariance matrix (elements 2 and 3)

coef(ols)[2:3]
sqrt(diag(vcov(ols)))[2:3]
```

```
# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Plot the regression surfaces via plot() (i.e., plot the 'partial
# regression surface plots').

plot(bw)

# Note - to plot regression surfaces with variability bounds constructed
# from bootstrapped standard errors, use the following (note that this
# may take a minute or two depending on the speed of your computer as
# the bootstrapping is done in real time, and note also that we override
# the default number of bootstrap replications (399) reducing them to 25
# in order to quickly compute standard errors in this instance - don't
# of course do this in general)

plot(bw,
      plot.errors.boot.num=25,
      plot.errors.method="bootstrap")

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we simulate an
# example for a partially linear model, and compare the coefficient
# estimates from the partially linear model with those from a correctly
# specified parametric model...

set.seed(42)

n <- 250
x1 <- rnorm(n)
x2 <- rbinom(n, 1, .5)

z1 <- rbinom(n, 1, .5)
z2 <- rnorm(n)

y <- 1 + x1 + x2 + z1 + sin(z2) + rnorm(n)

X <- data.frame(x1, factor(x2))
Z <- data.frame(factor(z1), z2)

# First, compute data-driven bandwidths. This may take a few minutes
# depending on the speed of your computer...

bw <- npplregbw(xdat=X, zdat=Z, ydat=y)

# Next, compute the partially linear fit

pl <- npplreg(bws=bw)

# Print a summary of the model...
```

```

summary(p1)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Retrieve the coefficient estimates and their standard errors...

coef(p1)
coef(p1, errors = TRUE)

# Compare the partially linear results to those from a correctly
# specified model's coefficients for x1 and x2

ols <- lm(y~x1+factor(x2)+factor(z1)+I(sin(z2)))

# The intercept is coef()[1], and those for x1 and x2 are coef()[2] and
# coef()[3]. The standard errors are the square root of the diagonal of
# the variance-covariance matrix (elements 2 and 3)

coef(ols)[2:3]
sqrt(diag(vcov(ols)))[2:3]

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Plot the regression surfaces via plot() (i.e., plot the 'partial
# regression surface plots').

plot(bw)

# Note - to plot regression surfaces with variability bounds constructed
# from bootstrapped standard errors, use the following (note that this
# may take a minute or two depending on the speed of your computer as
# the bootstrapping is done in real time, and note also that we override
# the default number of bootstrap replications (399) reducing them to 25
# in order to quickly compute standard errors in this instance - don't
# of course do this in general)

plot(bw,
      plot.errors.boot.num=25,
      plot.errors.method="bootstrap")

## End(Not run)

```

Description

npplregbw computes a bandwidth object for a partially linear kernel regression estimate of a one (1) dimensional dependent variable on $p + q$ -variate explanatory data, using the model $Y = X\beta + \Theta(Z) + \epsilon$ given a set of estimation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification, which can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npplregbw(...)

## S3 method for class 'formula'
npplregbw(formula, data, subset, na.action, call, ...)

## S3 method for class 'NULL'
npplregbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = stop("invoked without data 'zdat'"),
          bws,
          ...)

## Default S3 method:
npplregbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = stop("invoked without data 'zdat'"),
          bws,
          ...,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          ftol,
          tol,
          small)

## S3 method for class 'plbandwidth'
npplregbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = stop("invoked without data 'zdat'"),
          bws,
          nmulti,
          ...)
```

Arguments

formula a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

<code>data</code>	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
<code>subset</code>	an optional vector specifying a subset of observations to be used in the fitting process.
<code>na.action</code>	a function which indicates what should happen when the data contain NAs. The default is set by the <code>na.action</code> setting of options, and is <code>na.fail</code> if that is unset. The (recommended) default is <code>na.omit</code> .
<code>call</code>	the original function call. This is passed internally by <code>np</code> when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.
<code>xdat</code>	a p -variate data frame of explanatory data (training data), corresponding to X in the model equation, whose linear relationship with the dependent data Y is posited.
<code>ydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>xdat</code> .
<code>zdat</code>	a q -variate data frame of explanatory data (training data), corresponding to Z in the model equation, whose relationship to the dependent variable is unspecified (nonparametric)
<code>bws</code>	a bandwidth specification. This can be set as a <code>plbandwidth</code> object returned from an invocation of <code>npplregbw</code> , or as a matrix of bandwidths, each row is a set of bandwidths for Z , with a column for each variable Z_i . In the first row are the bandwidths for the regression of Y on Z , the following rows contain the bandwidths for the regressions of the columns of X on Z . If specified as a matrix additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on. If left unspecified, <code>npplregbw</code> will search for optimal bandwidths using <code>npregbw</code> in the course of calculations. If specified, <code>npplregbw</code> will use the given bandwidths as the starting point for the numerical search for optimal bandwidths, unless you specify <code>bandwidth.compute = FALSE</code> .
<code>...</code>	additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on. To do this, you may specify any of <code>regtype</code> , <code>bwmethod</code> , <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> , as described in <code>npregbw</code> .
<code>bandwidth.compute</code>	a logical value which specifies whether to do a numerical search for bandwidths or not. If set to <code>FALSE</code> , a <code>plbandwidth</code> object will be returned with bandwidths set to those specified in <code>bws</code> . Defaults to <code>TRUE</code> .
<code>nmulti</code>	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to <code>min(5, ncol(zdat))</code> .
<code>remin</code>	a logical value which when set as <code>TRUE</code> the search routine restarts from located minima for a minor gain in accuracy. Defaults to <code>TRUE</code>
<code>itmax</code>	integer number of iterations before failure in the numerical optimization routine. Defaults to 10000

ftol	tolerance on the value of the cross-validation function evaluated at located minima. Defaults to $1.19e-07$ (FLT_EPSILON)
tol	tolerance on the position of located minima of the cross-validation function. Defaults to $1.49e-08$ (sqrt(DBL_EPSILON))
small	a small number, at about the precision of the data type used. Defaults to $2.22e-16$ (DBL_EPSILON)

Details

nplregbw implements a variety of methods for nonparametric regression on multivariate (q -variate) explanatory data defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

nplregbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the xdat, ydat, and zdat parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame zdat may be a mix of continuous (default), unordered discrete (to be specified in the data frame zdat using `factor`), and ordered discrete (to be specified in the data frame zdat using `ordered`). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `np` for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data ~ parametric explanatory data | nonparametric explanatory data, where dependent data is a univariate response, and parametric explanatory data and nonparametric explanatory data are both series of variables specified by name, separated by the separation character ‘|’. For example, `y1 ~ x1 + x2 | z1` specifies that the bandwidth object for the partially linear model with response y_1 , linear parametric regressors x_1 and x_2 , and nonparametric regressor z_1 is to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

if bwtype is set to `fixed`, an object containing bandwidths (or scale factors if `bwscaling = TRUE`) is returned. If it is set to `generalized_nn` or `adaptive_nn`, then instead the k th nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored in a list under the component name `bw`. Each element is an `rbandwidth` object. The first element of the list corresponds to the regression of Y on Z . Each subsequent element is the bandwidth object corresponding to the regression of the i th column of X on Z . See examples for more information.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the i th observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting `ftol=.01` and `tol=.01` and conduct multistarting (the default is to restart `min(5, ncol(zdat))` times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set `bws=bw` on subsequent calls to this routine where `bw` is the initial bandwidth object). A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," *Statistica Sinica*, 14, 485-512.
- Racine, J.S. and L. Liu (2006), "A partially linear kernel estimator for categorical data," manuscript.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," *Journal of Econometrics*, 119, 99-130.
- Robinson, P.M. (1988), "Root-n-consistent semiparametric regression," *Econometrica*, 56, 931-954.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also

[npregbw](#), [npreg](#)

Examples

```

# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we simulate an
# example for a partially linear model and perform bandwidth selection

set.seed(42)

n <- 250
x1 <- rnorm(n)
x2 <- rbinom(n, 1, .5)

z1 <- rbinom(n, 1, .5)
z2 <- rnorm(n)

y <- 1 + x1 + x2 + z1 + sin(z2) + rnorm(n)

X <- data.frame(x1, factor(x2))
Z <- data.frame(factor(z1), z2)

# Compute data-driven bandwidths... this may take a minute or two
# depending on the speed of your computer...

bw <- npplregbw(formula=y~x1+factor(x2)|factor(z1)+z2)

summary(bw)

## Not run:

# Note - the default is to use the local constant estimator. If you wish
# to use instead a local linear estimator, this is accomplished via
# npplregbw(xdat=X, zdat=Z, ydat=y, regtype="ll")

# Note - see the example for npudensbw() for multiple illustrations
# of how to change the kernel function, kernel order, and so forth.

# You may want to manually specify your bandwidths
bw.mat <- matrix(data = c(0.19, 0.34, # y on Z
                        0.00, 0.74, # X[,1] on Z
                        0.29, 0.23), # X[,2] on Z
                 ncol = ncol(Z), byrow=TRUE)

bw <- npplregbw(formula=y~x1+factor(x2)|factor(z1)+z2,
                bws=bw.mat, bandwidth.compute=FALSE)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# You may want to tweak some of the bandwidths
bw$bw[[1]] # y on Z, alternatively bw$bw$yzbw
bw$bw[[1]]$bw <- c(0.17, 0.30)

```

```

bw$bw[[2]] # X[,1] on Z
bw$bw[[2]]$bw[1] <- 0.00054

summary(bw)

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we simulate an
# example for a partially linear model and perform bandwidth selection

set.seed(42)

n <- 250
x1 <- rnorm(n)
x2 <- rbinom(n, 1, .5)

z1 <- rbinom(n, 1, .5)
z2 <- rnorm(n)

y <- 1 + x1 + x2 + z1 + sin(z2) + rnorm(n)

X <- data.frame(x1, factor(x2))
Z <- data.frame(factor(z1), z2)

# Compute data-driven bandwidths... this may take a minute or two
# depending on the speed of your computer...

bw <- npplregbw(xdat=X, zdat=Z, ydat=y)

summary(bw)

# Note - the default is to use the local constant estimator. If you wish
# to use instead a local linear estimator, this is accomplished via
# npplregbw(xdat=X, zdat=Z, ydat=y, regtype="ll")

# Note - see the example for npudensbw() for multiple illustrations
# of how to change the kernel function, kernel order, and so forth.

# You may want to manually specify your bandwidths
bw.mat <- matrix(data = c(0.19, 0.34, # y on Z
                        0.00, 0.74, # X[,1] on Z
                        0.29, 0.23), # X[,2] on Z
                 ncol = ncol(Z), byrow=TRUE)

bw <- npplregbw(xdat=X, zdat=Z, ydat=y,
               bws=bw.mat, bandwidth.compute=FALSE)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# You may want to tweak some of the bandwidths
bw$bw[[1]] # y on Z, alternatively bw$bw$ybw
bw$bw[[1]]$bw <- c(0.17, 0.30)

```

```

bw$bw[[2]] # X[,1] on Z
bw$bw[[2]]$bw[1] <- 0.00054

summary(bw)

## End(Not run)

```

npqcmstest	<i>Kernel Consistent Quantile Regression Model Specification Test with Mixed Data Types</i>
------------	---

Description

npqcmstest implements a consistent test for correct specification of parametric quantile regression models (linear or nonlinear) as described in Racine (2006) which extends the work of Zheng (1998).

Usage

```

npqcmstest(formula,
            data = NULL,
            subset,
            xdat,
            ydat,
            model = stop(paste(sQuote("model"), " has not been provided")),
            tau = 0.5,
            distribution = c("bootstrap", "asymptotic"),
            bwydat = c("y", "varepsilon"),
            boot.method = c("iid", "wild", "wild-rademacher"),
            boot.num = 399,
            pivot = TRUE,
            density.weighted = TRUE,
            random.seed = 42,
            ...)

```

Arguments

formula	a symbolic description of variables on which the test is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
subset	an optional vector specifying a subset of observations to be used.
model	a model object obtained from a call to <code>rq</code> . Important: the call to <code>rq</code> must have the argument <code>model=TRUE</code> or <code>npqcmstest</code> will not work.

<code>xdat</code>	a p -variate data frame of explanatory data (training data) used to calculate the quantile regression estimators.
<code>ydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>xdat</code> .
<code>tau</code>	a numeric value specifying the τ th quantile is desired
<code>distribution</code>	a character string used to specify the method of estimating the distribution of the statistic to be calculated. <code>bootstrap</code> will conduct bootstrapping. <code>asymptotic</code> will use the normal distribution. Defaults to <code>bootstrap</code> .
<code>bwydat</code>	a character string used to specify <code>ydat</code> used in bandwidth selection. <code>varepsilon</code> uses $1 - \tau$, $-\tau$ for <code>ydat</code> while <code>y</code> will use <code>y</code> . Defaults to <code>y</code> .
<code>boot.method</code>	a character string used to specify the bootstrap method. <code>iid</code> will generate independent identically distributed draws. <code>wild</code> will use a wild bootstrap. <code>wild-rademacher</code> will use a wild bootstrap with Rademacher variables. Defaults to <code>iid</code> .
<code>boot.num</code>	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
<code>pivot</code>	a logical value specifying whether the statistic should be normalised such that it approaches $N(0, 1)$ in distribution. Defaults to <code>TRUE</code> .
<code>density.weighted</code>	a logical value specifying whether the statistic should be weighted by the density of <code>xdat</code> . Defaults to <code>TRUE</code> .
<code>random.seed</code>	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
<code>...</code>	additional arguments supplied to control bandwidth selection on the residuals. One can specify the bandwidth type, kernel types, and so on. To do this, you may specify any of <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> , as described in npregbw . This is necessary if you specify <code>bws</code> as a p -vector and not a bandwidth object, and you do not desire the default behaviours.

Value

`npqcmstest` returns an object of type `cmstest` with the following components, components will contain information related to J_n or I_n depending on the value of `pivot`:

<code>Jn</code>	the statistic J_n
<code>In</code>	the statistic I_n
<code>Omega.hat</code>	as described in Racine, J.S. (2006).
<code>q.*</code>	the various quantiles of the statistic J_n (or I_n if <code>pivot=FALSE</code>) are in components <code>q.90</code> , <code>q.95</code> , <code>q.99</code> (one-sided 1%, 5%, 10% critical values)
<code>P</code>	the P-value of the statistic
<code>Jn.bootstrap</code>	if <code>pivot=TRUE</code> contains the bootstrap replications of J_n
<code>In.bootstrap</code>	if <code>pivot=FALSE</code> contains the bootstrap replications of I_n

[summary](#) supports object of type `cmstest`.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Koenker, R.W. and G.W. Bassett (1978), "Regression quantiles," *Econometrica*, 46, 33-50.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Murphy, K. M. and F. Welch (1990), "Empirical age-earnings profiles," *Journal of Labor Economics*, 8, 202-229.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Racine, J.S. (2006), "Consistent specification testing of heteroskedastic parametric regression quantile models with mixed data," manuscript.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.
- Zheng, J. (1998), "A consistent nonparametric test of parametric regression models under conditional quantile restrictions," *Econometric Theory*, 14, 123-138.

Examples

```
# EXAMPLE 1: For this example, we conduct a consistent quantile regression
# model specification test for a parametric wage quantile regression
# model that is quadratic in age. The work of Murphy and Welch (1990)
# would suggest that this parametric quantile regression model is
# misspecified.
```

```
library("quantreg")
```

```
data("cps71")
attach(cps71)
```

```
model <- rq(logwage~age+I(age^2), tau=0.5, model=TRUE)
```

```
plot(age, logwage)
lines(age, fitted(model))
```

```
X <- data.frame(age)
```

```
# Note - this may take a few minutes depending on the speed of your
# computer...
```

```

npqcmstest(model = model, xdat = X, ydat = logwage, tau=0.5)

## Not run:

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next try Murphy & Welch's (1990) suggested quintic specification.

model <- rq(logwage~age+I(age^2)+I(age^3)+I(age^4)+I(age^5), model=TRUE)

plot(age, logwage)
lines(age, fitted(model))

X <- data.frame(age)

# Note - this may take a few minutes depending on the speed of your
# computer...

npqcmstest(model = model, xdat = X, ydat = logwage, tau=0.5)

detach(cps71)

## End(Not run)

```

npqreg

Kernel Quantile Regression with Mixed Data Types

Description

npqreg computes a kernel quantile regression estimate of a one (1) dimensional dependent variable on p -variate explanatory data, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification using the method of Li and Racine (2008). A bandwidth specification can be a conbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```

npqreg(bws, ...)

## S3 method for class 'formula'
npqreg(bws, data = NULL, newdata = NULL, ...)

## S3 method for class 'call'
npqreg(bws, ...)

## S3 method for class 'conbandwidth'

```

```

npqreg(bws,
      txdat = stop("training data 'txdat' missing"),
      tydat = stop("training data 'tydat' missing"),
      exdat,
      tau = 0.5,
      gradients = FALSE,
      ftol = 1.19209e-07,
      tol = 1.49012e-08,
      small = 2.22045e-16,
      itmax = 10000,
      ...)

## Default S3 method:
npqreg(bws, txdat, tydat, ...)

```

Arguments

bws	a bandwidth specification. This can be set as a <code>conbandwidth</code> object returned from an invocation of <code>npcdensbw</code> , or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in <code>txdat</code> . If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on.
tau	a numeric value specifying the τ th quantile is desired. Defaults to 0.5.
...	additional arguments supplied to specify the regression type, bandwidth type, kernel types, training data, and so on. To do this, you may specify any of <code>bwmethod</code> , <code>bwscaling</code> , <code>bwtype</code> , <code>cxkertype</code> , <code>cxkerorder</code> , <code>cykertype</code> , <code>cykerorder</code> , <code>uxkertype</code> , <code>uykertype</code> , <code>oxkertype</code> , <code>oykertype</code> , as described in <code>npcdensbw</code> .
data	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npcdensbw</code> was called.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
txdat	a p -variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the training data used to compute the bandwidth object.
tydat	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>txdat</code> . Defaults to the training data used to compute the bandwidth object.
exdat	a p -variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>txdat</code> .
gradients	[currently not supported] a logical value indicating that you want gradients computed and returned in the resulting <code>npregression</code> object. Defaults to <code>FALSE</code> .
itmax	integer number of iterations before failure in the numerical optimization routine. Defaults to 10000.

ftol	tolerance on the value of the objective function evaluated at located minima. Defaults to $1.19e-07$ (FLT_EPSILON).
tol	tolerance on the position of located minima of the objective function. Defaults to $1.49e-08$ (sqrt(DBL_EPSILON)).
small	a small number, at about the precision of the data type used. Defaults to $2.22e-16$ (DBL_EPSILON).

Value

npqreg returns a npqregression object. The generic functions `fitted` (or `quantile`), `se`, `predict`, and `gradients`, extract (or generate) estimated values, asymptotic standard errors on estimates, predictions, and gradients, respectively, from the returned object. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

eval	evaluation points
quantile	estimation of the quantile regression function (conditional quantile) at the evaluation points
quanterr	standard errors of the quantile regression estimates
quantgrad	gradients at each evaluation point
tau	the τ th quantile computed

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, 99, 1015-1026.
- Koenker, R. W. and G.W. Bassett (1978), "Regression quantiles," *Econometrica*, 46, 33-50.
- Koenker, R. (2005), *Quantile Regression*, Econometric Society Monograph Series, Cambridge University Press.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2008), "Nonparametric estimation of conditional CDF and quantile functions with mixed categorical and continuous data," *Journal of Business and Economic Statistics*, 26, 423-434.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also**quantreg****Examples**

```

# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we compute a
# bivariate nonparametric quantile regression estimate for Giovanni
# Baiocchi's Italian income panel (see Italy for details)

data("Italy")
attach(Italy)

# First, compute the likelihood cross-validation bandwidths (default).
# Note - this may take a few minutes depending on the speed of your
# computer...

bw <- npcdensbw(formula=gdp~ordered(year))

# Note - numerical search for computing the quantiles will take a
# minute or so...

model.q0.25 <- npqreg(bws=bw, tau=0.25)
model.q0.50 <- npqreg(bws=bw, tau=0.50)
model.q0.75 <- npqreg(bws=bw, tau=0.75)

# Plot the resulting quantiles manually...

plot(ordered(year), gdp,
     main="CDF Quantile Estimates for the Italian Income Panel",
     xlab="Year",
     ylab="GDP Quantiles")

lines(ordered(year), model.q0.25$quantile, col="red", lty=2)
lines(ordered(year), model.q0.50$quantile, col="blue", lty=3)
lines(ordered(year), model.q0.75$quantile, col="red", lty=2)

legend(ordered(1951), 32, c("tau = 0.25", "tau = 0.50", "tau = 0.75"),
      lty=c(2, 3, 2), col=c("red", "blue", "red"))

detach(Italy)

## Not run:

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we compute a
# bivariate nonparametric quantile regression estimate for Giovanni
# Baiocchi's Italian income panel (see Italy for details)

data("Italy")
attach(Italy)
data <- data.frame(ordered(year), gdp)

# First, compute the likelihood cross-validation bandwidths (default).

```

```

# Note - this may take a few minutes depending on the speed of your
# computer...

bw <- npcdensbw(xdat=ordered(year), ydat=gdp)

# Note - numerical search for computing the quantiles will take a
# minute or so...

model.q0.25 <- npqreg(bws=bw, tau=0.25)
model.q0.50 <- npqreg(bws=bw, tau=0.50)
model.q0.75 <- npqreg(bws=bw, tau=0.75)

# Plot the resulting quantiles manually...

plot(ordered(year), gdp,
     main="CDF Quantile Estimates for the Italian Income Panel",
     xlab="Year",
     ylab="GDP Quantiles")

lines(ordered(year), model.q0.25$quantile, col="red", lty=2)
lines(ordered(year), model.q0.50$quantile, col="blue", lty=3)
lines(ordered(year), model.q0.75$quantile, col="red", lty=2)

legend(ordered(1951), 32, c("tau = 0.25", "tau = 0.50", "tau = 0.75"),
      lty=c(2, 3, 2), col=c("red", "blue", "red"))

detach(Italy)

## End(Not run)

```

npreg

Kernel Regression with Mixed Data Types

Description

npreg computes a kernel regression estimate of a one (1) dimensional dependent variable on p -variate explanatory data, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification using the method of Racine and Li (2004) and Li and Racine (2004). A bandwidth specification can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```

npreg(bws, ...)

## S3 method for class 'formula'
npreg(bws, data = NULL, newdata = NULL, ...)

## S3 method for class 'call'

```

```

npreg(bws, ...)

## Default S3 method:
npreg(bws, txdat, tydat, ...)

## S3 method for class 'rbandwidth'
npreg(bws,
      txdat = stop("training data 'txdat' missing"),
      tydat = stop("training data 'tydat' missing"),
      exdat,
      eydat,
      gradients = FALSE,
      residuals = FALSE,
      ...)

```

Arguments

<code>bws</code>	a bandwidth specification. This can be set as a <code>rbandwidth</code> object returned from an invocation of <code>npregbw</code> , or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in <code>txdat</code> . If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on.
<code>...</code>	additional arguments supplied to specify the regression type, bandwidth type, kernel types, training data, and so on, detailed below.
<code>data</code>	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npregbw</code> was called.
<code>newdata</code>	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
<code>txdat</code>	a p -variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the training data used to compute the bandwidth object.
<code>tydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>txdat</code> . Defaults to the training data used to compute the bandwidth object.
<code>exdat</code>	a p -variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>txdat</code> .
<code>eydat</code>	a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors.
<code>gradients</code>	a logical value indicating that you want gradients computed and returned in the resulting <code>npregression</code> object. Defaults to <code>FALSE</code> .
<code>residuals</code>	a logical value indicating that you want residuals computed and returned in the resulting <code>npregression</code> object. Defaults to <code>FALSE</code> .

Details

Typical usages are (see below for a complete list of options and also the examples at the end of this help file)

Usage 1: first compute the bandwidth object via `npregbw` and then compute the conditional mean:

```
bw <- npregbw(y~x)
ghat <- npreg(bw)
```

Usage 2: alternatively, compute the bandwidth object indirectly:

```
ghat <- npreg(y~x)
```

Usage 3: modify the default kernel and order:

```
ghat <- npreg(y~x, ckertype="epanechnikov", ckerorder=4)
```

Usage 4: use the data frame interface rather than the formula interface:

```
ghat <- npreg(tydat=y, txdat=x, ckertype="epanechnikov", ckerorder=4)
```

`npreg` implements a variety of methods for regression on multivariate (p -variate) data, the types of which are possibly continuous and/or discrete (unordered, ordered). The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

Data contained in the data frame `txdat` may be a mix of continuous (default), unordered discrete (to be specified in the data frame `txdat` using `factor`), and ordered discrete (to be specified in the data frame `txdat` using `ordered`). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `np` for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

`npreg` returns a `npregression` object. The generic functions `fitted`, `residuals`, `se`, `predict`, and `gradients`, extract (or generate) estimated values, residuals, asymptotic standard errors on estimates, predictions, and gradients, respectively, from the returned object. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

eval	evaluation points
mean	estimates of the regression function (conditional mean) at the evaluation points
merr	standard errors of the regression function estimates
grad	estimates of the gradients at each evaluation point
gerr	standard errors of the gradient estimates
resid	if residuals = TRUE, in-sample or out-of-sample residuals where appropriate (or possible)
R2	coefficient of determination
MSE	mean squared error
MAE	mean absolute error
MAPE	mean absolute percentage error
CORR	absolute value of Pearson's correlation coefficient
SIGN	fraction of observations where fitted and observed values agree in sign

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and Q. Li and J.S. Racine (2007), "Nonparametric estimation of regression functions in the presence of irrelevant regressors," *The Review of Economics and Statistics*, 89, 784-789.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," *Statistica Sinica*, 14, 485-512.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous Data," *Journal of Econometrics*, 119, 99-130.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also

[loess](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we compute a
# bivariate nonparametric regression estimate for Giovanni Baiocchi's
# Italian income panel (see Italy for details)

data("Italy")
attach(Italy)

# First, compute the least-squares cross-validated bandwidths for the
# local constant estimator (default).

bw <- npregbw(formula=gdp~ordered(year))

# Now take these bandwidths and fit the model and gradients

model <- npreg(bws = bw, gradients = TRUE)

summary(model)

## Not run:

# Use plot() to visualize the regression function, add bootstrap
# error bars, and overlay the data on the same plot.

# Note - this may take a minute or two depending on the speed of your
# computer due to bootstrapping being conducted (<ctrl>-C will
# interrupt). Note - nothing will appear in the graphics window until
# all computations are completed (if you use
# plot.errors.method="asymptotic" the figure will instantly appear).

plot(bw, plot.errors.method="bootstrap")
points(ordered(year), gdp, cex=.2, col="red")

detach(Italy)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we compute a
# bivariate nonparametric regression estimate for Giovanni Baiocchi's
# Italian income panel (see Italy for details)

data("Italy")
attach(Italy)

# First, compute the least-squares cross-validated bandwidths for the
# local constant estimator (default).

bw <- npregbw(xdat=ordered(year), ydat=gdp)

# Now take these bandwidths and fit the model and gradients
```

```
model <- npreg(bws = bw, gradients = TRUE)

summary(model)

# Use plot() to visualize the regression function, add bootstrap
# error bars, and overlay the data on the same plot.

# Note - this may take a minute or two depending on the speed of your
# computer due to bootstrapping being conducted (<ctrl>-C will
# interrupt). Note - nothing will appear in the graphics window until
# all computations are completed (if you use
# plot.errors.method="asymptotic" the figure will instantly appear).

plot(bw, plot.errors.method="bootstrap")
points(ordered(year), gdp, cex=.2, col="red")

detach(Italy)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 2 (INTERFACE=FORMULA): For this example, we compute a local
# linear fit using the AIC_c bandwidth selection criterion. We then plot
# the estimator and its gradient using asymptotic standard errors.

data("cps71")
attach(cps71)

bw <- npregbw(logwage~age, regtype="ll", bwmethod="cv.aic")

# Next, plot the regression function...

plot(bw, plot.errors.method="asymptotic")
points(age, logwage, cex=.2, col="red")

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next, plot the derivative...

plot(bw, plot.errors.method="asymptotic", gradient=TRUE)

detach(cps71)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 2 (INTERFACE=DATA FRAME): For this example, we compute a local
# linear fit using the AIC_c bandwidth selection criterion. We then plot
```

```

# the estimator and its gradient using asymptotic standard errors.

data("cps71")
attach(cps71)

bw <- npregbw(xdat=age, ydat=logwage, regtype="ll", bwmethod="cv.aic")

# Next, plot the regression function...

plot(bw, plot.errors.method="asymptotic")
points(age, logwage, cex=.2, col="red")

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next, plot the derivative...

plot(bw, plot.errors.method="asymptotic", gradient=TRUE)

detach(cps71)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# EXAMPLE 3 (INTERFACE=FORMULA): For this example, we replicate the
# nonparametric regression in Maasoumi, Racine, and Stengos
# (2007) (see oecdpanel for details). Note that X is multivariate
# containing a mix of unordered, ordered, and continuous data types. Note
# - this may take a few minutes depending on the speed of your computer.

data("oecdpanel")
attach(oecdpanel)

bw <- npregbw(formula=growth~
               factor(oecd)+
               factor(year)+
               initgdp+
               popgro+
               inv+
               humancap)

plot(bw, plot.errors.method="asymptotic")

detach(oecdpanel)

# EXAMPLE 3 (INTERFACE=DATA FRAME): For this example, we replicate the
# nonparametric regression in Maasoumi, Racine, and Stengos
# (2007) (see oecdpanel for details). Note that X is multivariate
# containing a mix of unordered, ordered, and continuous data types. Note
# - this may take a few minutes depending on the speed of your computer.

```

```
data("oecdpanel")
attach(oecdpanel)

y <- growth
X <- data.frame(factor(oecd), factor(year), initgdp, popgro, inv, humancap)

bw <- npregbw(xdat=X, ydat=y)

plot(bw, plot.errors.method="asymptotic")

detach(oecdpanel)

# EXAMPLE 4 (INTERFACE=FORMULA): Experimental data - the effect of
# vitamin C on tooth growth in guinea pigs
#
# Description:
#
# The response is the length of odontoblasts (teeth) in each of 10
# guinea pigs at each of three dose levels of Vitamin C (0.5, 1, and
# 2 mg) with each of two delivery methods (orange juice or ascorbic
# acid).
#
# Usage:
#
# ToothGrowth
#
# Format:
#
# A data frame with 60 observations on 3 variables.
#
# [ ,1] len    numeric  Tooth length
# [ ,2] supp   factor    Supplement type (VC or OJ).
# [ ,3] dose   numeric  Dose in milligrams.

library("datasets")
attach(ToothGrowth)

# Note - in this example, there are six cells.

bw <- npregbw(formula=len~factor(supp)+ordered(dose))

# Now plot the partial regression surfaces with bootstrapped
# nonparametric confidence bounds

plot(bw, plot.errors.method="bootstrap", plot.errors.type="quantile")

detach(ToothGrowth)

# EXAMPLE 4 (INTERFACE=DATA FRAME): Experimental data - the effect of
# vitamin C on tooth growth in guinea pigs
#
# Description:
```

```

#
# The response is the length of odontoblasts (teeth) in each of 10
# guinea pigs at each of three dose levels of Vitamin C (0.5, 1, and
# 2 mg) with each of two delivery methods (orange juice or ascorbic
# acid).
#
# Usage:
#
# ToothGrowth
#
# Format:
#
# A data frame with 60 observations on 3 variables.
#
#      [,1] len      numeric  Tooth length
#      [,2] supp     factor    Supplement type (VC or OJ).
#      [,3] dose     numeric   Dose in milligrams.

library("datasets")
attach(ToothGrowth)

# Note - in this example, there are six cells.

y <- len
X <- data.frame(supp=factor(supp), dose=ordered(dose))

bw <- npregbw(X, y)

# Now plot the partial regression surfaces with bootstrapped
# nonparametric confidence bounds

plot(bw, plot.errors.method="bootstrap", plot.errors.type="quantile")

detach(ToothGrowth)

# EXAMPLE 5 (INTERFACE=FORMULA): a pretty 2-d smoothing example adapted
# from the R mgcv library which was written by Simon N. Wood.

set.seed(12345)

# This function generates a smooth nonlinear DGP

dgp.func <- function(x, z, sx=0.3, sz=0.4)
{ (pi*sx*sz)*(1.2*exp(-(x-0.2)^2/sx^2-(z-0.3)^2/sz^2)+
  0.8*exp(-(x-0.7)^2/sx^2-(z-0.8)^2/sz^2))
}

# Generate 500 observations, compute the true DGP (i.e., no noise),
# then a noisy sample

n <- 500

x <- runif(n)

```

```

z <- runif(n)

xs <- seq(0, 1, length=30)
zs <- seq(0, 1, length=30)

X.eval <- data.frame(x=rep(xs, 30), z=rep(zs, rep(30, 30)))

dgp <- matrix(dgp.func(X.eval$x, X.eval$z), 30, 30)

y <- dgp.func(x, z)+rnorm(n)*0.1

# Prepare the screen for output... first, plot the true DGP

split.screen(c(2, 1))

screen(1)

persp(xs, zs, dgp, xlab="x1", ylab="x2", zlab="y", main="True DGP")

# Next, compute a local linear fit and plot that

bw <- npregbw(formula=y~x+z, regtype="ll", bwmethod="cv.aic")
fit <- fitted(npreg(bws=bw, newdata=X.eval))
fit.mat <- matrix(fit, 30, 30)

screen(2)

persp(xs, zs, fit.mat, xlab="x1", ylab="x2", zlab="y",
      main="Local linear estimate")

# EXAMPLE 5 (INTERFACE=DATA FRAME): a pretty 2-d smoothing example
# adapted from the R mgcv library which was written by Simon N. Wood.

set.seed(12345)

# This function generates a smooth nonlinear DGP

dgp.func <- function(x, z, sx=0.3, sz=0.4)
{ (pi*sx*sz)*(1.2*exp(-(x-0.2)^2/sx^2-(z-0.3)^2/sz^2)+
  0.8*exp(-(x-0.7)^2/sx^2-(z-0.8)^2/sz^2))
}

# Generate 500 observations, compute the true DGP (i.e., no noise),
# then a noisy sample

n <- 500

x <- runif(n)
z <- runif(n)

xs <- seq(0, 1, length=30)
zs <- seq(0, 1, length=30)

```

```

X <- data.frame(x, z)
X.eval <- data.frame(x=rep(xs, 30), z=rep(zs, rep(30, 30)))

dgp <- matrix(dgp.func(X.eval$x, X.eval$z), 30, 30)

y <- dgp.func(x, z)+rnorm(n)*0.1

# Prepare the screen for output... first, plot the true DGP

split.screen(c(2, 1))

screen(1)

persp(xs, zs, dgp, xlab="x1", ylab="x2", zlab="y", main="True DGP")

# Next, compute a local linear fit and plot that

bw <- npregbw(xdat=X, ydat=y, regtype="ll", bwmethod="cv.aic")
fit <- fitted(npreg(exdat=X.eval, bws=bw))
fit.mat <- matrix(fit, 30, 30)

screen(2)

persp(xs, zs, fit.mat, xlab="x1", ylab="x2", zlab="y",
      main="Local linear estimate")

## End(Not run)

```

npregbw

Kernel Regression Bandwidth Selection with Mixed Data Types

Description

npregbw computes a bandwidth object for a p -variate kernel regression estimator defined over mixed continuous and discrete (unordered, ordered) data using expected Kullback-Leibler cross-validation, or least-squares cross validation using the method of Racine and Li (2004) and Li and Racine (2004).

Usage

```

npregbw(...)

## S3 method for class 'formula'
npregbw(formula, data, subset, na.action, call, ...)

## S3 method for class 'NULL'
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,

```

```

... )

## Default S3 method:
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        bandwidth.compute = TRUE,
        nmulti,
        remin,
        itmax,
        ftol,
        tol,
        small,
        regtype,
        bwmethod,
        bwscaling,
        bwtype,
        ckertype,
        ckerorder,
        ukertype,
        okertype,
        ...)

## S3 method for class 'rbandwidth'
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        bandwidth.compute = TRUE,
        nmulti,
        remin = TRUE,
        itmax = 10000,
        ftol = 1.19209e-07,
        tol = 1.49012e-08,
        small = 2.22045e-16,
        ...)

```

Arguments

formula	a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset.

	The (recommended) default is <code>na.omit</code> .
<code>call</code>	the original function call. This is passed internally by <code>np</code> when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.
<code>xdat</code>	a p -variate data frame of regressors which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
<code>ydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>xdat</code> .
<code>bws</code>	a bandwidth specification. This can be set as a <code>rbandwidth</code> object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in <code>xdat</code> . In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.
<code>...</code>	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
<code>regtype</code>	a character string specifying which type of kernel regression estimator to use. <code>lc</code> specifies a local-constant estimator (Nadaraya-Watson) and <code>ll</code> specifies a local-linear estimator. Defaults to <code>lc</code> .
<code>bwmethod</code>	which method to use to select bandwidths. <code>cv.aic</code> specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and <code>cv.ls</code> specifies least-squares cross-validation. Defaults to <code>cv.ls</code> .
<code>bwscaling</code>	a logical value that when set to <code>TRUE</code> the supplied bandwidths are interpreted as ‘scale factors’ (c_j), otherwise when the value is <code>FALSE</code> they are interpreted as ‘raw bandwidths’ (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j \sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of continuous variable j defined as $\min(\text{standard deviation, interquartile range}/1.349)$, n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j . Defaults to <code>FALSE</code> .
<code>bwtype</code>	character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the bandwidth object. Defaults to <code>fixed</code> . Option summary: <code>fixed</code> : compute fixed bandwidths <code>generalized_nn</code> : compute generalized nearest neighbors <code>adaptive_nn</code> : compute adaptive nearest neighbors
<code>bandwidth.compute</code>	a logical value which specifies whether to do a numerical search for bandwidths or not. If set to <code>FALSE</code> , a <code>rbandwidth</code> object will be returned with bandwidths set to those specified in <code>bws</code> . Defaults to <code>TRUE</code> .
<code>ckertype</code>	character string used to specify the continuous kernel type. Can be set as <code>gaussian</code> , <code>epanechnikov</code> , or <code>uniform</code> . Defaults to <code>gaussian</code> .

ckerorder	numeric value specifying kernel order (one of (2, 4, 6, 8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.
ukertype	character string used to specify the unordered categorical kernel type. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.
okertype	character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin or liracine. Defaults to wangvanryzin.
nmulti	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to $\min(5, \text{ncol}(\text{xdat}))$.
remin	a logical value which when set as TRUE the search routine restarts from located minima for a minor gain in accuracy. Defaults to TRUE.
itmax	integer number of iterations before failure in the numerical optimization routine. Defaults to 10000.
ftol	tolerance on the value of the cross-validation function evaluated at located minima. Defaults to $1.19\text{e-}07$ (FLT_EPSILON).
tol	tolerance on the position of located minima of the cross-validation function. Defaults to $1.49\text{e-}08$ ($\sqrt{\text{DBL_EPSILON}}$).
small	a small number, at about the precision of the data type used. Defaults to $2.22\text{e-}16$ (DBL_EPSILON).

Details

npregbw implements a variety of methods for choosing bandwidths for multivariate (p -variate) regression data defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms (direction set (Powell’s) methods in multidimensions).

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

npregbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the xdat and ydat parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame xdat may be a mix of continuous (default), unordered discrete (to be specified in the data frame xdat using [factor](#)), and ordered discrete (to be specified in the data frame xdat using [ordered](#)). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see [np](#) for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data ~ explanatory data, where dependent data is a univariate

response, and explanatory data is a series of variables specified by name, separated by the separation character '+'. For example, $y_1 \sim x_1 + x_2$ specifies that the bandwidths for the regression of response y_1 and nonparametric regressors x_1 and x_2 are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npregbw returns a rbandwidth object, with the following components:

bw	bandwidth(s), scale factor(s) or nearest neighbours for the data, xdat
fval	objective function value at minimum

if bwtype is set to fixed, an object containing bandwidths (or scale factors if bwscaling = TRUE) is returned. If it is set to generalized_nn or adaptive_nn, then instead the k th nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored under the component name bw, with each element i corresponding to column i of input data xdat.

The functions [predict](#), [summary](#), and [plot](#) support objects of this class.

Usage Issues

If you are using data of mixed types, then it is advisable to use the [data.frame](#) function to construct your input data and not [cbind](#), since [cbind](#) will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the i th observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(5, ncol(xdat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set bws=bw on subsequent calls to this routine where bw is the initial bandwidth object). A version of this package using the Rmpi wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Hall, P. and Q. Li and J.S. Racine (2007), "Nonparametric estimation of regression functions in the presence of irrelevant regressors," *The Review of Economics and Statistics*, 89, 784-789.
- Hurvich, C.M. and J.S. Simonoff and C.L. Tsai (1998), "Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion," *Journal of the Royal Statistical Society B*, 60, 271-293.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," *Statistica Sinica*, 14, 485-512.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," *Journal of Econometrics*, 119, 99-130.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also

[npreg](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we compute a
# Bivariate nonparametric regression estimate for Giovanni Baiocchi's
# Italian income panel (see Italy for details)

data("Italy")
attach(Italy)

# Compute the least-squares cross-validated bandwidths for the local
# constant estimator (default)

bw <- npregbw(formula=gdp~ordered(year))

summary(bw)

## Not run:

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Supply your own bandwidth...

bw <- npregbw(formula=gdp~ordered(year), bws=c(0.75),
              bandwidth.compute=FALSE)
```

```

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Treat year as continuous and supply your own scaling factor c in
# c sigma n^{-1/(2p+q)}

bw <- npregbw(formula=gdp~year, bws=c(1.06),
               bandwidth.compute=FALSE,
               bwscaling=TRUE)

summary(bw)

# Note - see also the example for npudensbw() for more extensive
# multiple illustrations of how to change the kernel function, kernel
# order, bandwidth type and so forth.

detach(Italy)

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we compute a
# Bivariate nonparametric regression estimate for Giovanni Baiocchi's
# Italian income panel (see Italy for details)

data("Italy")
attach(Italy)

# Compute the least-squares cross-validated bandwidths for the local
# constant estimator (default)

bw <- npregbw(xdat=ordered(year), ydat=gdp)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Supply your own bandwidth...

bw <- npregbw(xdat=ordered(year), ydat=gdp, bws=c(0.75),
               bandwidth.compute=FALSE)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Treat year as continuous and supply your own scaling factor c in
# c sigma n^{-1/(2p+q)}

```

```

bw <- npregbw(xdat=year, ydat=gdp, bws=c(1.06),
             bandwidth.compute=FALSE,
             bwscaling=TRUE)

summary(bw)

# Note - see also the example for npudensbw() for more extensive
# multiple illustrations of how to change the kernel function, kernel
# order, bandwidth type and so forth.

detach(Italy)

## End(Not run)

```

npregiv

Nonparametric Instrumental Regression

Description

npregiv computes nonparametric estimation of an instrumental regression function φ defined by conditional moment restrictions stemming from a structural econometric model: $E[Y - \varphi(Z, X)|W] = 0$, and involving endogenous variables Y and Z and exogenous variables X and instruments W . The function φ is the solution of an ill-posed inverse problem.

When method="Tikhonov", npregiv uses the approach of Darolles, Fan, Florens and Renault (2011) modified for local polynomial kernel regression of any order (Darolles et al use local constant kernel weighting which corresponds to setting $p=0$; see below for details). When method="Landweber-Fridman", npregiv uses the approach of Horowitz (2011) again using local polynomial kernel regression (Horowitz uses B-spline weighting).

Usage

```

npregiv(y,
        z,
        w,
        x = NULL,
        zeval = NULL,
        weval = NULL,
        xeval = NULL,
        p = 1,
        nmulti = 1,
        random.seed = 42,
        optim.maxattempts = 10,
        optim.method = c("Nelder-Mead", "BFGS", "CG"),
        optim.reltol = sqrt(.Machine$double.eps),
        optim.abstol = .Machine$double.eps,
        optim.maxit = 500,
        alpha = NULL,

```

```

alpha.min = 1e-10,
alpha.max = 1e-01,
alpha.tol = .Machine$double.eps^0.25,
iterate.max = 1000,
iterate.tol = 1.0e-04,
iterate.diff.tol = 1.0e-08,
constant = 0.5,
method = c("Landweber-Fridman", "Tikhonov"),
stop.on.increase = TRUE,
...)
```

Arguments

y	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of z .
z	a p -variate data frame of endogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
w	a q -variate data frame of instruments. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
x	an r -variate data frame of exogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
zeval	a p -variate data frame of endogenous regressors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by z .
weval	a q -variate data frame of instruments on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by w .
xeval	an r -variate data frame of exogenous regressors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by x .
p	the order of the local polynomial regression (defaults to $p=1$, i.e. local linear).
nmulti	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.
random.seed	an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.
optim.method	method used by <code>optim</code> for minimization of the objective function. See <code>?optim</code> for references. Defaults to "Nelder-Mead". the default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions. method "BFGS" is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized. method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak-Ribiere or Beale-Sorenson updates). Conjugate gradient methods will generally be more fragile than the

BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.

<code>optim.maxattempts</code>	maximum number of attempts taken trying to achieve successful convergence in <code>optim</code> . Defaults to 100.
<code>optim.abstol</code>	the absolute convergence tolerance used by <code>optim</code> . Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to <code>.Machine\$double.eps</code> .
<code>optim.reltol</code>	relative convergence tolerance used by <code>optim</code> . The algorithm stops if it is unable to reduce the value by a factor of <code>'reltol * (abs(val) + reltol)'</code> at a step. Defaults to <code>sqrt(.Machine\$double.eps)</code> , typically about <code>1e-8</code> .
<code>optim.maxit</code>	maximum number of iterations used by <code>optim</code> . Defaults to 500.
<code>alpha</code>	a numeric scalar that, if supplied, is used rather than numerically solving for <code>alpha</code> , when using <code>method="Tikhonov"</code> .
<code>alpha.min</code>	minimum of search range for <code>alpha</code> , the Tikhonov regularization parameter, when using <code>method="Tikhonov"</code> .
<code>alpha.max</code>	maximum of search range for <code>alpha</code> , the Tikhonov regularization parameter, when using <code>method="Tikhonov"</code> .
<code>alpha.tol</code>	the search tolerance for <code>optimize</code> when solving for <code>alpha</code> , the Tikhonov regularization parameter, when using <code>method="Tikhonov"</code> .
<code>iterate.max</code>	an integer indicating the maximum number of iterations permitted before termination occurs when using <code>method="Landweber-Fridman"</code> .
<code>iterate.tol</code>	the search tolerance for the stopping rule when using <code>method="Landweber-Fridman"</code> .
<code>iterate.diff.tol</code>	the search tolerance for the difference in the stopping rule from iteration to iteration when using <code>method="Landweber-Fridman"</code> (disable by setting to zero).
<code>constant</code>	the constant to use when using <code>method="Landweber-Fridman"</code> .
<code>method</code>	the regularization method employed (defaults to <code>"Landweber-Fridman"</code> , see Horowitz (2011); see Darolles, Fan, Florens and Renault (2011) for details for <code>"Tikhonov"</code>).
<code>stop.on.increase</code>	a logical value (defaults to <code>TRUE</code>) indicating whether to halt iteration if the stopping criterion (see below) increases over the course of one iteration (i.e. it may be above the iteration tolerance but increased).
<code>...</code>	additional arguments supplied to <code>npksum</code> .

Details

Tikhonov regularization requires computation of weight matrices of dimension $n \times n$ which can be computationally costly in terms of memory requirements and may be unsuitable for large datasets. Landweber-Fridman will be preferred in such settings as it does not require construction and storage of these weight matrices while it also avoids the need for numerical optimization methods to determine α .

method="Landweber-Fridman" uses an optimal stopping rule based upon $\|E(y|w) - E(\varphi_k(z, x)|w)\|^2$. However, if insufficient training is conducted the estimates can be overly noisy. To best guard against this eventuality set `nmulti` to a larger number than the default `nmulti=0` for `npreg`.

When using method="Landweber-Fridman", iteration will terminate when either the change in the value of $\|(E(y|w) - E(\varphi_k(z, x)|w))/E(y|w)\|^2$ from iteration to iteration is less than `iterate.tol` or we hit `iterate.max` or $\|E(y|w) - E(\varphi_k(z, x)|w)\|^2$ stops falling in value and starts rising.

Value

`npregiv` returns a list with components `phi` and either `alpha` when method="Tikhonov" or `num.iterations` and `norm.stop` when method="Landweber-Fridman".

Note

This function should be considered to be in 'beta test' status until further notice.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>, Samuele Centorrino <samuele.centorrino@univ-tlse1.fr>

References

Carrasco, M. and J.P. Florens and E. Renault (2007), "Linear Inverse Problems in Structural Econometrics Estimation Based on Spectral Decomposition and Regularization," In: James J. Heckman and Edward E. Leamer, Editor(s), *Handbook of Econometrics*, Elsevier, 2007, Volume 6, Part 2, Chapter 77, Pages 5633-5751

Darolles, S. and Y. Fan and J.P. Florens and E. Renault (2011), "Nonparametric Instrumental Regression," *Econometrica*, 79, 1541-1565.

Fève, F. and J.P. Florens (2010), "The practice of non-parametric estimation by solving inverse problems: the example of transformation models," *Econometrics Journal*, 13, S1-S27.

Florens, J.P. and J.S. Racine (2012), "Nonparametric Instrumental Derivatives," Working Paper.

Fridman, V. M. (1956), "A Method of Successive Approximations for Fredholm Integral Equations of the First Kind," *Uspekhi, Math. Nauk.*, 11, 233-334, in Russian.

Horowitz, J.L. (2011), "Applied Nonparametric Instrumental Variables Estimation," *Econometrica*, 79, 347-394.

Landweber, L. (1951), "An iterative formula for Fredholm integral equations of the first kind," *American Journal of Mathematics*, 73, 615-24.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," *Statistica Sinica*, 14, 485-512.

See Also

[npregivderiv](#), [npreg](#)

Examples

```

## Not run:
## This illustration was made possible by Samuele Centorrino
## <samuele.centorrino@univ-tlse1.fr>

set.seed(42)
n <- 1500

## The DGP is as follows:

## 1)  $y = \phi(z) + u$ 

## 2)  $E(u|z) \neq 0$  (endogeneity present)

## 3) Suppose there exists an instrument  $w$  such that  $z = f(w) + v$  and
##  $E(u|w) = 0$ 

## 4) We generate  $v$ ,  $w$ , and generate  $u$  such that  $u$  and  $z$  are
## correlated. To achieve this we express  $u$  as a function of  $v$  (i.e.  $u =$ 
##  $\gamma v + \epsilon$ )

v <- rnorm(n,mean=0,sd=0.27)
eps <- rnorm(n,mean=0,sd=0.05)
u <- -0.5*v + eps
w <- rnorm(n,mean=0,sd=1)

## In Darolles et al (2011) there exist two DGPs. The first is
##  $\phi(z)=z^2$  and the second is  $\phi(z)=\exp(-\text{abs}(z))$  (which is
## discontinuous and has a kink at zero).

fun1 <- function(z) { z^2 }
fun2 <- function(z) { exp(-abs(z)) }

z <- 0.2*w + v

## Generate two y vectors for each function.

y1 <- fun1(z) + u
y2 <- fun2(z) + u

## You set y to be either y1 or y2 (ditto for phi) depending on which
## DGP you are considering:

y <- y1
phi <- fun1

## Sort on z (for plotting)

ivdata <- data.frame(y,z,w)
ivdata <- ivdata[order(ivdata$z),]
rm(y,z,w)
attach(ivdata)

```

```

model.iv <- npregiv(y=y,z=z,w=w)
phi.iv <- model.iv$phi

## Now the non-iv local linear estimator of E(y|z)

ll.mean <- fitted(npreg(y~z,regtype="ll"))

## For the plots, restrict focal attention to the bulk of the data
## (i.e. for the plotting area trim out 1/4 of one percent from each
## tail of y and z)

trim <- 0.0025

curve(phi,min(z),max(z),
      xlim=quantile(z,c(trim,1-trim)),
      ylim=quantile(y,c(trim,1-trim)),
      ylab="Y",
      xlab="Z",
      main="Nonparametric Instrumental Kernel Regression",
      lwd=2,lty=1)

points(z,y,type="p",cex=.25,col="grey")

lines(z,phi.iv,col="blue",lwd=2,lty=2)

lines(z,ll.mean,col="red",lwd=2,lty=4)

legend(quantile(z,trim),quantile(y,1-trim),
      c(expression(paste(varphi(z))),
        expression(paste("Nonparametric ",hat(varphi)(z))),
        "Nonparametric E(y|z)"),
      lty=c(1,2,4),
      col=c("black","blue","red"),
      lwd=c(2,2,2))

## End(Not run)

```

 npregivderiv

Nonparametric Instrumental Derivatives

Description

npregivderiv uses the approach of Florens and Racine (2012) to compute the partial derivative of a nonparametric estimation of an instrumental regression function φ defined by conditional moment restrictions stemming from a structural econometric model: $E[Y - \varphi(Z, X)|W] = 0$, and involving endogenous variables Y and Z and exogenous variables X and instruments W . The derivative function φ' is the solution of an ill-posed inverse problem, and is computed using Landweber-Fridman regularization.

Usage

```

npregivderiv(y,
             z,
             w,
             x = NULL,
             zeval = NULL,
             weval = NULL,
             xeval = NULL,
             p = 1,
             nmulti = 1,
             random.seed = 42,
             optim.maxattempts = 10,
             optim.method = c("Nelder-Mead", "BFGS", "CG"),
             optim.reltol = sqrt(.Machine$double.eps),
             optim.abstol = .Machine$double.eps,
             optim.maxit = 500,
             iterate.max = 1000,
             iterate.tol = 1.0e-04,
             iterate.diff.tol = 1.0e-08,
             constant = 0.5,
             start.phi.zero = FALSE,
             smooth.while.iterating = TRUE,
             stop.on.increase = TRUE,
             smooth.residuals = TRUE,
             ...)

```

Arguments

<i>y</i>	a one (1) dimensional numeric or integer vector of dependent data, each element <i>i</i> corresponding to each observation (row) <i>i</i> of <i>z</i> .
<i>z</i>	a <i>p</i> -variate data frame of endogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
<i>w</i>	a <i>q</i> -variate data frame of instruments. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
<i>x</i>	an <i>r</i> -variate data frame of exogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
<i>zeval</i>	a <i>p</i> -variate data frame of endogenous regressors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <i>z</i> .
<i>weval</i>	a <i>q</i> -variate data frame of instruments on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <i>w</i> .
<i>xeval</i>	an <i>r</i> -variate data frame of exogenous regressors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <i>x</i> .
<i>p</i>	the order of the local polynomial regression (defaults to <i>p</i> =1, i.e. local linear).

nmulti	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.
random.seed	an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.
optim.method	method used by <code>optim</code> for minimization of the objective function. See <code>?optim</code> for references. Defaults to "Nelder-Mead". the default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions. method "BFGS" is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized. method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak-Ribiere or Beale-Sorenson updates). Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.
optim.maxattempts	maximum number of attempts taken trying to achieve successful convergence in <code>optim</code> . Defaults to 100.
optim.abstol	the absolute convergence tolerance used by <code>optim</code> . Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to <code>.Machine\$double.eps</code> .
optim.reltol	relative convergence tolerance used by <code>optim</code> . The algorithm stops if it is unable to reduce the value by a factor of <code>'reltol * (abs(val) + reltol)'</code> at a step. Defaults to <code>sqrt(.Machine\$double.eps)</code> , typically about <code>1e-8</code> .
optim.maxit	maximum number of iterations used by <code>optim</code> . Defaults to 500.
iterate.max	an integer indicating the maximum number of iterations permitted before termination occurs for Landweber-Fridman iteration.
iterate.tol	the search tolerance for the stopping rule for Landweber-Fridman iteration.
iterate.diff.tol	the search tolerance for the difference in the stopping rule from iteration to iteration when using Landweber-Fridman iteration (disable by setting to zero).
constant	the constant to use for Landweber-Fridman iteration.
start.phi.zero	a logical value indicating whether to commence Landweber-Fridman assuming $\varphi_{-1} = 0$ (proper Landweber-Fridman) or instead begin from $E(y z)$ (defaults to FALSE, see details below).
smooth.while.iterating	a logical value indicating whether to re-smooth $E(\varphi w)$ during Landweber-Fridman iteration (defaults to TRUE).
stop.on.increase	a logical value (defaults to TRUE) indicating whether to halt iteration if the stopping criterion (see below) increases over the course of one iteration (i.e. it may be above the iteration tolerance but increased).

smooth.residuals

a logical value (defaults to TRUE) indicating whether to optimize bandwidths for the regression of $y - \varphi(z)$ on w or for the regression of $\varphi(z)$ on w during Landweber-Fridman iteration.

... additional arguments supplied to `npksum`.

Details

Note that Landweber-Fridman iteration presumes that $\varphi_{-1} = 0$, and so for derivative estimation we commence iterating from a model having derivatives all equal to zero. Given this starting point it may require a fairly large number of iterations in order to converge. Other perhaps more reasonable starting values might present themselves. When `start.phi.zero` is set to FALSE iteration will commence instead using derivatives from the conditional mean model $E(y|z)$. Should the default iteration terminate quickly or you are concerned about your results, it would be prudent to verify that this alternative starting value produces the same result. Also, check the `norm.stop` vector for any anomalies (such as the error criterion increasing immediately).

Landweber-Fridman iteration uses an optimal stopping rule based upon $\|E(y|w) - E(\varphi_k(z, x)|w)\|^2$. However, if insufficient training is conducted the estimates can be overly noisy. To best guard against this eventuality set `nmulti` to a larger number than the default `nmulti=0` for `npreg`.

Iteration will terminate when either the change in the value of $\|(E(y|w) - E(\varphi_k(z, x)|w))/E(y|w)\|^2$ from iteration to iteration is less than `iterate.tol` or we hit `iterate.max` or $\|E((y|w) - E(\varphi_k(z, x)|w))/E(y|w)\|^2$ stops falling in value and starts rising.

Value

`npregivderiv` returns a list with components `phi.prime`, `phi`, `num.iterations` and `norm.stop`.

Note

This function currently supports univariate z only. This function should be considered to be in ‘beta test’ status until further notice.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Carrasco, M. and J.P. Florens and E. Renault (2007), “Linear Inverse Problems in Structural Econometrics Estimation Based on Spectral Decomposition and Regularization,” In: James J. Heckman and Edward E. Leamer, Editor(s), Handbook of Econometrics, Elsevier, 2007, Volume 6, Part 2, Chapter 77, Pages 5633-5751

Darolles, S. and Y. Fan and J.P. Florens and E. Renault (2011), “Nonparametric Instrumental Regression,” *Econometrica*, 79, 1541-1565.

Fève, F. and J.P. Florens (2010), “The practice of non-parametric estimation by solving inverse problems: the example of transformation models,” *Econometrics Journal*, 13, S1-S27.

Florens, J.P. and J.S. Racine (2012), “Nonparametric Instrumental Derivatives,” Working Paper.

Fridman, V. M. (1956), "A Method of Successive Approximations for Fredholm Integral Equations of the First Kind," *Uspekhi, Math. Nauk.*, 11, 233-334, in Russian.

Horowitz, J.L. (2011), "Applied Nonparametric Instrumental Variables Estimation," *Econometrica*, 79, 347-394.

Landweber, L. (1951), "An iterative formula for Fredholm integral equations of the first kind," *American Journal of Mathematics*, 73, 615-24.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," *Statistica Sinica*, 14, 485-512.

See Also

[npregiv](#), [npreg](#)

Examples

```
## Not run:
## This illustration was made possible by Samuele Centorrino
## <samuele.centorrino@univ-tlse1.fr>

set.seed(42)
n <- 1500

## For trimming the plot (trim .5% from each tail)

trim <- 0.005

## The DGP is as follows:

## 1)  $y = \phi(z) + u$ 

## 2)  $E(u|z) \neq 0$  (endogeneity present)

## 3) Suppose there exists an instrument  $w$  such that  $z = f(w) + v$  and
##  $E(u|w) = 0$ 

## 4) We generate  $v$ ,  $w$ , and generate  $u$  such that  $u$  and  $z$  are
## correlated. To achieve this we express  $u$  as a function of  $v$  (i.e.  $u =$ 
##  $\gamma v + \epsilon$ )

v <- rnorm(n,mean=0,sd=0.27)
eps <- rnorm(n,mean=0,sd=0.05)
u <- -0.5*v + eps
w <- rnorm(n,mean=0,sd=1)

## In Darolles et al (2011) there exist two DGPs. The first is
##  $\phi(z)=z^2$  and the second is  $\phi(z)=\exp(-\text{abs}(z))$  (which is
## discontinuous and has a kink at zero).
```

```

fun1 <- function(z) { z^2 }
fun2 <- function(z) { exp(-abs(z)) }

z <- 0.2*w + v

## Generate two y vectors for each function.

y1 <- fun1(z) + u
y2 <- fun2(z) + u

## You set y to be either y1 or y2 (ditto for phi) depending on which
## DGP you are considering:

y <- y1
phi <- fun1

## Sort on z (for plotting)

ivdata <- data.frame(y,z,w,u,v)
ivdata <- ivdata[order(ivdata$z),]
rm(y,z,w,u,v)
attach(ivdata)

model.ivderiv <- npregivderiv(y=y,z=z,w=w)

ylim <-c(quantile(model.ivderiv$phi.prime,trim),
         quantile(model.ivderiv$phi.prime,1-trim))

plot(z,model.ivderiv$phi.prime,
     xlim=quantile(z,c(trim,1-trim)),
     main="",
     ylim=ylim,
     xlab="Z",
     ylab="Derivative",
     type="l",
     lwd=2)
rug(z)

## End(Not run)

```

Description

npscoef computes a kernel regression estimate of a one (1) dimensional dependent variable on p -variate explanatory data, using the model $Y_i = W_i' \gamma(Z_i) + u_i$ where $W_i' = (1, X_i')$, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification. A bandwidth specification can be a scbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```

npscoef(bws, ...)

## S3 method for class 'formula'
npscoef(bws, data = NULL, newdata = NULL, ...)

## S3 method for class 'call'
npscoef(bws, ...)

## Default S3 method:
npscoef(bws, txdat, tydat, tzdat, ...)

## S3 method for class 'scbandwidth'
npscoef(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        tzdat = NULL,
        exdat,
        eydat,
        ezdat,
        residuals = FALSE,
        errors = TRUE,
        iterate = TRUE,
        maxiter = 100,
        tol = .Machine$double.eps,
        leave.one.out = FALSE,
        betas = FALSE,
        ...)

```

Arguments

<code>bws</code>	a bandwidth specification. This can be set as a <code>scbandwidth</code> object returned from an invocation of <code>npscoefbw</code> , or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in <code>tzdat</code> . If specified as a vector additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, training data, and so on.
<code>...</code>	additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on. To do this, you may specify any of <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , as described in <code>npscoefbw</code> .
<code>data</code>	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npscoefbw</code> was called.
<code>newdata</code>	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
<code>txdat</code>	a p -variate data frame of explanatory data (training data), which, by default, populates the columns 2 through $p + 1$ of W in the model equation, and in the

	absence of <code>zdat</code> , will also correspond to Z from the model equation. Defaults to the training data used to compute the bandwidth object.
<code>tydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>txdat</code> . Defaults to the training data used to compute the bandwidth object.
<code>tzdat</code>	a optionally specified q -variate data frame of explanatory data (training data), which corresponds to Z in the model equation. Defaults to the training data used to compute the bandwidth object.
<code>exdat</code>	a p -variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>txdat</code> .
<code>eydat</code>	a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors.
<code>ezdat</code>	an optionally specified q -variate data frame of points on which the regression will be estimated (evaluation data), which corresponds to Z in the model equation. Defaults to be the same as <code>txdat</code> .
<code>errors</code>	a logical value indicating whether or not asymptotic standard errors should be computed and returned in the resulting <code>smoothcoefficient</code> object. Defaults to TRUE.
<code>residuals</code>	a logical value indicating that you want residuals computed and returned in the resulting <code>smoothcoefficient</code> object. Defaults to FALSE.
<code>iterate</code>	a logical value indicating whether or not backfitted estimates should be iterated for self-consistency. Defaults to TRUE.
<code>maxiter</code>	integer specifying the maximum number of times to iterate the backfitted estimates while attempting make the backfitted estimates converge to the desired tolerance. Defaults to 100.
<code>tol</code>	desired tolerance on the relative convergence of backfit estimates. Defaults to <code>.Machine\$double.eps</code> .
<code>leave.one.out</code>	a logical value to specify whether or not to compute the leave one out estimates. Will not work if <code>e[xyz]dat</code> is specified. Defaults to FALSE.
<code>betas</code>	a logical value indicating whether or not estimates of the components of γ should be returned in the <code>smoothcoefficient</code> object along with the regression estimates. Defaults to FALSE.

Value

`npscoef` returns a `smoothcoefficient` object. The generic functions `fitted`, `residuals`, `coef`, `se`, and `predict`, extract (or generate) estimated values, residuals, coefficients, bootstrapped standard errors on estimates, and predictions, respectively, from the returned object. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

<code>eval</code>	evaluation points
<code>mean</code>	estimation of the regression function (conditional mean) at the evaluation points
<code>merr</code>	if <code>errors = TRUE</code> , standard errors of the regression estimates

beta	if betas = TRUE, estimates of the coefficients γ at the evaluation points
resid	if residuals = TRUE, in-sample or out-of-sample residuals where appropriate (or possible)
R2	coefficient of determination
MSE	mean squared error
MAE	mean absolute error
MAPE	mean absolute percentage error
CORR	absolute value of Pearson's correlation coefficient
SIGN	fraction of observations where fitted and observed values agree in sign

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Support for backfitted bandwidths is experimental and is limited in functionality. The code does not support asymptotic standard errors or out of sample estimates with backfitting.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Cai Z. (2007), "Trending time-varying coefficient time series models with serially correlated errors," *Journal of Econometrics*, 136, 163-188.
- Hastie, T. and R. Tibshirani (1993), "Varying-coefficient models," *Journal of the Royal Statistical Society, B* 55, 757-796.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2010), "Smooth varying-coefficient estimation and inference for qualitative and quantitative data," *Econometric Theory*, 26, 1-31.
- Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
- Racine, J.S. and D. Ouyang and Q. Li (2010), "Semiparametric Hierarchical Models," manuscript.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

See Also

[bw.nrd](#), [bw.SJ](#), [hist](#), [npudens](#), [npudist](#), [npudensbw](#), [npscoefbw](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA):

n <- 250
x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))
bw <- npscoefbw(y~x|z)
model <- npscoef(bw)
plot(model)

## Not run:
# EXAMPLE 1 (INTERFACE=DATA FRAME):

n <- 250
x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))
bw <- npscoefbw(xdat=x, ydat=y, zdat=z)
model <- npscoef(bw)
plot(model)

## End(Not run)
```

npscoefbw

Smooth Coefficient Kernel Regression Bandwidth Selection

Description

npscoefbw computes a bandwidth object for a smooth coefficient kernel regression estimate of a one (1) dimensional dependent variable on $p + q$ -variate explanatory data, using the model $Y_i = W_i' \gamma(Z_i) + u_i$ where $W_i' = (1, X_i')$ given training points (consisting of explanatory data and dependent data), and a bandwidth specification, which can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npscoefbw(...)
```

S3 method for class 'formula'

```
npscoefbw(formula, data, subset, na.action, call, ...)
```

S3 method for class 'NULL'

```
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          ...)
```

```
## Default S3 method:
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          nmulti,
          random.seed,
          cv.iterate,
          cv.num.iterations,
          backfit.iterate,
          backfit.maxiter,
          backfit.tol,
          bandwidth.compute = TRUE,
          bwmethod,
          bwscaling,
          bwtype,
          ckertype,
          ckerorder,
          ukertype,
          okertype,
          optim.method,
          optim.maxattempts,
          optim.reltol,
          optim.abstol,
          optim.maxit,
          ...)

## S3 method for class 'scbandwidth'
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          nmulti,
          random.seed = 42,
          cv.iterate = FALSE,
          cv.num.iterations = 1,
          backfit.iterate = FALSE,
          backfit.maxiter = 100,
          backfit.tol = .Machine$double.eps,
          bandwidth.compute = TRUE,
          optim.method = c("Nelder-Mead", "BFGS", "CG"),
          optim.maxattempts = 10,
          optim.reltol = sqrt(.Machine$double.eps),
          optim.abstol = .Machine$double.eps,
          optim.maxit = 500,
          ...)
```

Arguments

formula	a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the <code>na.action</code> setting of options, and is <code>na.fail</code> if that is unset. The (recommended) default is <code>na.omit</code> .
call	the original function call. This is passed internally by <code>np</code> when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.
xdat	a p -variate data frame of explanatory data (training data), which, by default, populates the columns 2 through $p + 1$ of W in the model equation, and in the absence of <code>zdat</code> , will also correspond to Z from the model equation.
ydat	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>xdat</code> .
zdat	an optionally specified q -variate data frame of explanatory data (training data), which corresponds to Z in the model equation. Defaults to be the same as <code>xdat</code> .
bws	a bandwidth specification. This can be set as a <code>scbandwidth</code> object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in <code>xdat</code> . In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.
...	additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on, detailed below.
bandwidth.compute	a logical value which specifies whether to do a numerical search for bandwidths or not. If set to <code>FALSE</code> , a <code>scbandwidth</code> object will be returned with bandwidths set to those specified in <code>bws</code> . Defaults to <code>TRUE</code> .
bwmethod	which method was used to select bandwidths. <code>cv.ls</code> specifies least-squares cross-validation, which is all that is currently supported. Defaults to <code>cv.ls</code> .
bwscaling	a logical value that when set to <code>TRUE</code> the supplied bandwidths are interpreted as ‘scale factors’ (c_j), otherwise when the value is <code>FALSE</code> they are interpreted as ‘raw bandwidths’ (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j \sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of continuous variable j defined as $\min(\text{standard deviation, interquartile range}/1.349)$, n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j . Defaults to <code>FALSE</code> .

bwtype	character string used for the continuous variable bandwidth type, specifying the type of bandwidth provided. Defaults to fixed. Option summary: fixed: fixed bandwidths or scale factors generalized_nn: generalized nearest neighbors adaptive_nn: adaptive nearest neighbors
ckertype	character string used to specify the continuous kernel type. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.
ckerorder	numeric value specifying kernel order (one of (2,4,6,8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.
ukertype	character string used to specify the unordered categorical kernel type. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.
okertype	character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin or liracine. Defaults to wangvanryzin.
nmulti	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to $\min(5, \text{ncol}(\text{xdat}))$.
random.seed	an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.
optim.method	method used by <code>optim</code> for minimization of the objective function. See <code>?optim</code> for references. Defaults to "Nelder-Mead". the default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions. method "BFGS" is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized. method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak-Ribiere or Beale-Sorenson updates). Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.
optim.maxattempts	maximum number of attempts taken trying to achieve successful convergence in <code>optim</code> . Defaults to 100.
optim.abstol	the absolute convergence tolerance used by <code>optim</code> . Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to <code>.Machine\$double.eps</code> .
optim.reltol	relative convergence tolerance used by <code>optim</code> . The algorithm stops if it is unable to reduce the value by a factor of <code>'reltol * (abs(val) + reltol)'</code> at a step. Defaults to <code>sqrt(.Machine\$double.eps)</code> , typically about $1e-8$.
optim.maxit	maximum number of iterations used by <code>optim</code> . Defaults to 500.
cv.iterate	boolean value specifying whether or not to perform iterative, cross-validated backfitting on the data. See details for limitations of the backfitting procedure. Defaults to FALSE.

<code>cv.num.iterations</code>	integer specifying the number of times to iterate the backfitting process over all covariates. Defaults to 1.
<code>backfit.iterate</code>	boolean value specifying whether or not to iterate evaluations of the smooth coefficient estimator, for extra accuracy, during the cross-validated backfitting procedure. Defaults to FALSE.
<code>backfit.maxiter</code>	integer specifying the maximum number of times to iterate the evaluation of the smooth coefficient estimator in the attempt to obtain the desired accuracy. Defaults to 100.
<code>backfit.tol</code>	tolerance to determine convergence of iterated evaluations of the smooth coefficient estimator. Defaults to <code>.Machine\$double.eps</code> .

Details

`npscoefbw` implements a variety of methods for semiparametric regression on multivariate ($p + q$ -variate) explanatory data defined over a set of possibly continuous data. The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

`npscoefbw` may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the `xdat`, `ydat`, and `zdat` parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame `xdat` may be continuous and in `zdat` may be of mixed type. Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see [np](#) for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data \sim parametric explanatory data | nonparametric explanatory data, where dependent data is a univariate response, and parametric explanatory data and nonparametric explanatory data are both series of variables specified by name, separated by the separation character ‘+’. For example, `y1 ~ x1 + x2 | z1` specifies that the bandwidth object for the smooth coefficient model with response `y1`, linear parametric regressors `x1` and `x2`, and nonparametric regressor (that is, the slope-changing variable) `z1` is to be estimated. See below for further examples. In the case where the nonparametric (slope-changing) variable is not specified, it is assumed to be the same as the parametric variable.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

if `bwtype` is set to `fixed`, an object containing bandwidths (or scale factors if `bwscaling = TRUE`) is returned. If it is set to `generalized_nn` or `adaptive_nn`, then instead the k th nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored in a vector under the component name `bw`. Backfitted bandwidths are stored under the component name `bw.fitted`.

The functions `predict`, `summary`, and `plot` support objects of this class.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the i th observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting `optim.reltol=.1` and conduct multistarting (the default is to restart `min(5,ncol(zdat))` times). Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set `bws=bw` on subsequent calls to this routine where `bw` is the initial bandwidth object). A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Support for backfitted bandwidths is experimental and is limited in functionality. The code does not support asymptotic standard errors or out of sample estimates with backfitting.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Cai Z. (2007), "Trending time-varying coefficient time series models with serially correlated errors," *Journal of Econometrics*, 136, 163-188.
- Hastie, T. and R. Tibshirani (1993), "Varying-coefficient models," *Journal of the Royal Statistical Society*, B 55, 757-796.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Li, Q. and J.S. Racine (2010), "Smooth varying-coefficient estimation and inference for qualitative and quantitative data," *Econometric Theory*, 26, 1-31.

Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.
 Racine, J.S. and D. Ouyang and Q. Li (2010), "Semiparametric Hierarchical Models," manuscript.
 Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions,"
Biometrika, 68, 301-309.

See Also

[npregbw](#), [npreg](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA):
set.seed(42)

n <- 100
x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))
bw <- npscoefbw(formula=y~x|z)
summary(bw)

## Not run:
# EXAMPLE 1 (INTERFACE=DATA FRAME):

n <- 100
x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))
bw <- npscoefbw(xdat=x, ydat=y, zdat=z)
summary(bw)

## End(Not run)
```

npsdeptest

Kernel Consistent Serial Dependence Test for Univariate Nonlinear Processes

Description

npsdeptest implements the consistent metric entropy test of nonlinear serial dependence as described in Granger, Maasoumi and Racine (2004).

Usage

```
npsdeptest(data = NULL,
            lag.num = 1,
            method = c("integration", "summation"),
            bootstrap = TRUE,
```

```
boot.num = 399,
random.seed = 42)
```

Arguments

data	a vector containing the variable that can be of type <code>numeric</code> or <code>ts</code> .
lag.num	an integer value specifying the maximum number of lags to use. Defaults to 1.
method	a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as <code>integration</code> or <code>summation</code> (see below for details). Defaults to <code>integration</code> .
bootstrap	a logical value which specifies whether to conduct the bootstrap test or not. If set to <code>FALSE</code> , only the statistic will be computed. Defaults to <code>TRUE</code> .
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Details

`npsdeptest` computes the nonparametric metric entropy (normalized Hellinger of Granger, Maaoumi and Racine (2004)) for testing for nonlinear serial dependence, $D[f(y_t, \hat{y}_{t-k}), f(y_t) \times f(\hat{y}_{t-k})]$. Default bandwidths are of the Kullback-Leibler variety obtained via likelihood cross-validation.

The test may be applied to a raw data series or to residuals of user estimated models.

The summation version of this statistic may be numerically unstable when data is sparse (the summation version involves division of densities while the integration version involves differences). Warning messages are produced should this occur ('integration recommended') and should be heeded.

Value

`npsdeptest` returns an object of type `deptest` with the following components

<code>Srho</code>	the statistic vector <code>Srho</code>
<code>Srho.cumulant</code>	the cumulant statistic vector <code>Srho.cumulant</code>
<code>Srho.bootstrap.mat</code>	contains the bootstrap replications of <code>Srho</code>
<code>Srho.cumulant.bootstrap.mat</code>	contains the bootstrap replications of <code>Srho.cumulant</code>
<code>P</code>	the P-value vector of the <code>Srho</code> statistic vector
<code>P.cumulant</code>	the P-value vector of the cumulant <code>Srho</code> statistic vector
<code>bootstrap</code>	a logical value indicating whether bootstrapping was performed
<code>boot.num</code>	number of bootstrap replications
<code>lag.num</code>	the number of lags

bw.y	the numeric vector of bandwidths for data marginal density at lag num.lag
bw.y.lag	the numeric vector of bandwidths for lagged data marginal density at lag num.lag
bw.joint	the numeric matrix of bandwidths for data and lagged data joint density at lag num.lag

[summary](#) supports object of type deptest.

Usage Issues

The integration version of the statistic uses multidimensional numerical methods from the [cubature](#) package. See [adaptIntegrate](#) for details. The integration version of the statistic will be substantially slower than the summation version, however, it will likely be both more accurate and powerful.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), “A dependence metric for possibly non-linear processes”, *Journal of Time Series Analysis*, 25, 649-669.

See Also

[npdeptest](#), [npdeneqtest](#), [npsymtest](#), [npunitest](#)

Examples

```
set.seed(1234)

## A function to create a time series

ar.series <- function(phi,epsilon) {
  n <- length(epsilon)
  series <- numeric(n)
  series[1] <- epsilon[1]/(1-phi)
  for(i in 2:n) {
    series[i] <- phi*series[i-1] + epsilon[i]
  }
  return(series)
}

n <- 100

## Stationary persistent time-series

yt <- ar.series(0.95,rnorm(n))
npsdeptest(yt,lag.num=2,boot.num=99,method="summation")

Sys.sleep(5)
```

```

## Stationary independent time-series

yt <- ar.series(0.0,rnorm(n))
npsdeptest(yt,lag.num=2,boot.num=99,method="summation")

## Not run:

## Stationary persistent time-series

yt <- ar.series(0.95,rnorm(n))
npsdeptest(yt,lag.num=2,boot.num=99,method="integration")

Sys.sleep(5)

## Stationary independent time-series

yt <- ar.series(0.0,rnorm(n))
npsdeptest(yt,lag.num=2,boot.num=99,method="integration")

## End(Not run)

```

npseed

Set Random Seed

Description

npseed is a function which sets the random seed in the [np](#) C backend, resetting the random number generator.

Usage

```
npseed(seed)
```

Arguments

seed an integer seed for the random number generator.

Details

npseed provides an interface for setting the random seed (and resetting the random number generator) used by [np](#). The random number generator is used during the bandwidth search procedure to set the search starting point, and in subsequent searches when using multistarting, to avoid being trapped in local minima if the objective function is not globally concave.

Calling npseed will only affect the numerical search if it is performed by the C backend. The affected functions include: [npudensbw](#), [npcdensbw](#), [npregbw](#), [npplregbw](#), [npqreg](#), [npcmstest](#) (via [npregbw](#)), [npqcmstest](#) (via [npregbw](#)), [npsigtest](#) (via [npregbw](#)).

Value

None.

Note

This method currently only supports objects from the [np](#) library.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

See Also

[set.seed](#)

Examples

```
npseed(712)
x <- runif(10)
y <- x + rnorm(10, sd = 0.1)
bw <- npregbw(y~x)
```

npsigtest

Kernel Regression Significance Test with Mixed Data Types

Description

npsigtest implements a consistent test of significance of an explanatory variable in a nonparametric regression setting that is analogous to a simple t -test in a parametric regression setting. The test is based on Racine, Hart, and Li (2006) and Racine (1997).

Usage

```
npsigtest(bws, ...)
```

S3 method for class 'formula'

```
npsigtest(bws, data = NULL, ...)
```

S3 method for class 'call'

```
npsigtest(bws, ...)
```

S3 method for class 'npregression'

```
npsigtest(bws, ...)
```

```
## Default S3 method:
npsigtest(bws, xdat, ydat, ...)

## S3 method for class 'rbandwidth'
npsigtest(bws,
          xdat = stop("data xdat missing"),
          ydat = stop("data ydat missing"),
          boot.num = 399,
          boot.method = c("iid", "wild", "wild-rademacher"),
          boot.type = c("I", "II"),
          index = seq(1, ncol(xdat)),
          random.seed = 42,
          ...)
```

Arguments

<code>bws</code>	a bandwidth specification. This can be set as a <code>rbandwidth</code> object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in <code>xdat</code> . In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths when using <code>boot.type="II"</code> . If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on.
<code>data</code>	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npregbw</code> was called.
<code>xdat</code>	a p -variate data frame of explanatory data (training data) used to calculate the regression estimators.
<code>ydat</code>	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of <code>xdat</code> .
<code>boot.method</code>	a character string used to specify the bootstrap method. <code>iid</code> will generate independent identically distributed draws. <code>wild</code> will use a wild bootstrap. <code>wild-rademacher</code> will use a wild bootstrap with Rademacher variables. Defaults to <code>iid</code> .
<code>boot.num</code>	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
<code>boot.type</code>	a character string specifying whether to use a 'Bootstrap I' or 'Bootstrap II' method (see Racine, Hart, and Li (2006) for details). The 'Bootstrap II' method re-runs cross-validation for each bootstrap replication and uses the new cross-validated bandwidth for variable i and the original ones for the remaining variables. Defaults to <code>boot.type="I"</code> .
<code>index</code>	a vector of indices for the columns of <code>xdat</code> for which the test of significance is to be conducted. Defaults to $(1, 2, \dots, p)$ where p is the number of columns in <code>xdat</code> .

random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
...	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Value

npsigtest returns an object of type sigtest. [summary](#) supports sigtest objects. It has the following components:

In	the vector of statistics In
P	the vector of P-values for each statistic in In
In.bootstrap	contains a matrix of the bootstrap replications of the vector In, each column corresponding to replications associated with explanatory variables in xdat indexed by index (e.g., if you selected index = c(1,4) then In.bootstrap will have two columns, the first being the bootstrap replications of In associated with variable 1, the second with variable 4).

Usage Issues

If you are using data of mixed types, then it is advisable to use the [data.frame](#) function to construct your input data and not [cbind](#), since [cbind](#) will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: bootstrap methods are, by their nature, *computationally intensive*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default number of bootstrap replications, say, setting them to boot.num=99 A version of this package using the [Rmpi](#) wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Racine, J.S., J. Hart, and Q. Li (2006), "Testing the significance of categorical predictor variables in nonparametric regression models," *Econometric Reviews*, 25, 523-544.
- Racine, J.S. (1997), "Consistent significance testing for nonparametric regression," *Journal of Business and Economic Statistics* 15, 369-379.
- Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, 68, 301-309.

Examples

```

# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we simulate 100 draws
# from a DGP in which z, the first column of X, is an irrelevant
# discrete variable

set.seed(12345)

n <- 100

z <- rbinom(n,1,.5)
x1 <- rnorm(n)
x2 <- runif(n,-2,2)

y <- x1 + x2 + rnorm(n)

# Next, we must compute bandwidths for our regression model. In this
# case we conduct local linear regression. Note - this may take a few
# minutes depending on the speed of your computer...

bw <- npregbw(formula=y~factor(z)+x1+x2,regtype="ll",bwmethod="cv.aic")

# We then compute a vector of tests corresponding to the columns of
# X. Note - this may take a few minutes depending on the speed of your
# computer... we have to generate the null distribution of the statistic
# for each variable whose significance is being tested using 399
# bootstrap replications for each...

npsigtest(bws=bw)

## Not run:

# If you wished, you could conduct the test for, say, variables 1 and 3
# only, as in

npsigtest(bws=bw,index=c(1,3))

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we simulate 100
# draws from a DGP in which z, the first column of X, is an irrelevant
# discrete variable

set.seed(12345)

n <- 100

z <- rbinom(n,1,.5)
x1 <- rnorm(n)
x2 <- runif(n,-2,2)

X <- data.frame(factor(z),x1,x2)

y <- x1 + x2 + rnorm(n)

```

```

# Next, we must compute bandwidths for our regression model. In this
# case we conduct local linear regression. Note - this may take a few
# minutes depending on the speed of your computer...

bw <- npregbw(xdat=X,ydat=y,regtype="ll",bwmethod="cv.aic")

# We then compute a vector of tests corresponding to the columns of
# X. Note - this may take a few minutes depending on the speed of your
# computer... we have to generate the null distribution of the statistic
# for each variable whose significance is being tested using 399
# bootstrap replications for each...

npsigtest(bws=bw)

# If you wished, you could conduct the test for, say, variables 1 and 3
# only, as in

npsigtest(bws=bw,index=c(1,3))

## End(Not run)

```

npsymtest

Kernel Consistent Density Asymmetry Test with Mixed Data Types

Description

npsymtest implements the consistent metric entropy test of asymmetry as described in Maasoumi and Racine (2009).

Usage

```

npsymtest(data = NULL,
           method = c("integration","summation"),
           boot.num = 399,
           bw = NULL,
           boot.method = c("iid", "geom"),
           random.seed = 42,
           ...)

```

Arguments

data	a vector containing the variable.
method	a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as integration or summation (see below for details). Defaults to integration.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
bw	a numeric (scalar) bandwidth. Defaults to plug-in (see details below).

<code>boot.method</code>	a character string used to specify the bootstrap method. Can be set as <code>iid</code> or <code>geom</code> (see below for details). Defaults to <code>iid</code> .
<code>random.seed</code>	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
<code>...</code>	additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is used since we specify <code>bw</code> as a numeric scalar and not a bandwidth object, and is of interest if you do not desire the default behaviours. To change the defaults, you may specify any of <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> .

Details

`npsymtest` computes the nonparametric metric entropy (normalized Hellinger of Granger, Maasoumi and Racine (2004)) for testing symmetry using the densities/probabilities of the data and the rotated data, $D[f(y), f(\hat{y})]$. See Maasoumi and Racine (2009) for details. Default bandwidths are of the plug-in variety (`bw.SJ` for continuous variables and direct plug-in for discrete variables).

For bootstrapping the null distribution of the statistic, `iid` conducts simple random resampling, while `geom` conducts Politis and Romano's (1994) stationary bootstrap using automatic block length selection via the `b.star` function in the `np` package. See the `boot` package for details.

The summation version of this statistic may be numerically unstable when `y` is sparse (the summation version involves division of densities while the integration version involves differences). Warning messages are produced should this occur ('integration recommended') and should be heeded.

Value

`npsymtest` returns an object of type `symtest` with the following components

<code>Srho</code>	the statistic <code>Srho</code>
<code>Srho.bootstrap</code>	contains the bootstrap replications of <code>Srho</code>
<code>P</code>	the P-value of the statistic
<code>boot.num</code>	number of bootstrap replications
<code>data.rotate</code>	the rotated data series
<code>bw</code>	the numeric (scalar) bandwidth

`summary` supports object of type `symtest`.

Usage Issues

When using data of type `factor` it is crucial that the variable not be an alphabetic character string (i.e. the factor must be integer-valued). The rotation is conducted about the median after conversion to type `numeric` which is then converted back to type `factor`. Failure to do so will have unpredictable results. See the example below for proper usage.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly non-linear processes", *Journal of Time Series Analysis*, 25, 649-669.

Maasoumi, E. and J.S. Racine (2009), "A robust entropy-based test of asymmetry for discrete and continuous processes," *Econometric Reviews*, 28, 246-261.

Politis, D.N. and J.P. Romano (1994), "The stationary bootstrap," *Journal of the American Statistical Association*, 89, 1303-1313.

See Also

[npdeneqtest](#), [npdeptest](#), [npsdeptest](#), [npunitest](#)

Examples

```
set.seed(1234)

n <- 100

## Asymmetric discrete probability distribution function

x <- factor(rbinom(n,2,.8))
npsymtest(x,boot.num=99)

Sys.sleep(5)

## Symmetric discrete probability distribution function

x <- factor(rbinom(n,2,.5))
npsymtest(x,boot.num=99)

Sys.sleep(5)

## Asymmetric continuous distribution function

y <- rchisq(n,df=2)
npsymtest(y,boot.num=99)

Sys.sleep(5)

## Symmetric continuous distribution function

y <- rnorm(n)
npsymtest(y,boot.num=99)

## Not run:

## Time-series bootstrap

ar.series <- function(phi,epsilon) {
  n <- length(epsilon)
  series <- numeric(n)
```

```

series[1] <- epsilon[1]/(1-phi)
for(i in 2:n) {
  series[i] <- phi*series[i-1] + epsilon[i]
}
return(series)
}

## Asymmetric time-series

yt <- ar.series(0.5,rchisq(n,df=3))
npsymtest(yt,boot.num=99,boot.method="geom")

## Symmetric time-series

yt <- ar.series(0.5,rnorm(n))
npsymtest(yt,boot.num=99,boot.method="geom")

## End(Not run)

```

npudens

Kernel Density and Distribution Estimation with Mixed Data Types

Description

npudens computes kernel unconditional density estimates on evaluation data, given a set of training data and a bandwidth specification (a bandwidth object or a bandwidth vector, bandwidth type, and kernel type) using the method of Li and Racine (2003). Similarly [npudist](#) computes kernel unconditional cumulative distribution estimates.

Usage

```

npudens(bws, ...)

## S3 method for class 'formula'
npudens(bws, data = NULL, newdata = NULL, ...)

## S3 method for class 'bandwidth'
npudens(bws,
        tdat = stop("invoked without training data 'tdat'"),
        edat,
        ...)

## S3 method for class 'call'
npudens(bws, ...)

## Default S3 method:
npudens(bws, tdat, ...)

```

Arguments

bws	a bandwidth specification. This can be set as a bandwidth object returned from an invocation of <code>npudensbw</code> , or as a p -vector of bandwidths, with an element for each variable in the training data. If specified as a vector, then additional arguments will need to be supplied as necessary to change them from the defaults to specify the bandwidth type, kernel types, training data, and so on.
...	additional arguments supplied to specify, the training data, the bandwidth type, kernel types, and so on. This is necessary if you specify <code>bws</code> as a p -vector and not a bandwidth object, and you do not desire the default behaviours. To do this, you may specify any of <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> , as described in <code>npudensbw</code> .
tdata	a p -variate data frame of sample realizations (training data) used to estimate the density. Defaults to the training data used to compute the bandwidth object.
edata	a p -variate data frame of density evaluation points. By default, evaluation takes place on the data provided by <code>tdata</code> .
data	an optional data frame, list or environment (or object coercible to a data frame by <code>as.data.frame</code>) containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(bws)</code> , typically the environment from which <code>npudensbw</code> was called.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Details

Typical usages are (see below for a complete list of options and also the examples at the end of this help file)

Usage 1: first compute the bandwidth object via `npudensbw` and then compute the density:

```
bw <- npudensbw(~y)
fhat <- npudens(bw)
```

Usage 2: alternatively, compute the bandwidth object indirectly:

```
fhat <- npudens(~y)
```

Usage 3: modify the default kernel and order:

```
fhat <- npudens(~y, ckertype="epanechnikov", ckerorder=4)
```

Usage 4: use the data frame interface rather than the formula interface:

```
fhat <- npudens(tdat = y, ckertype="epanechnikov", ckerorder=4)
```

npudens and npudist implement a variety of methods for estimating multivariate distributions (p -variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

Data contained in the data frame tdat (and also edat) may be a mix of continuous (default), unordered discrete (to be specified in the data frame tdat using the `factor` command), and ordered discrete (to be specified in the data frame tdat using the `ordered` command). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `np` for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken’s (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npudens returns a npdensity object, similarly npudist returns a npdistribution object. The generic accessor functions `fitted`, and `se`, extract estimated values and asymptotic standard errors on estimates, respectively, from the returned object. Furthermore, the functions `predict`, `summary` and `plot` support objects of both classes. The returned objects have the following components:

<code>eval</code>	the evaluation points.
<code>dens</code> or <code>dist</code>	estimation of the density (cumulative distribution) at the evaluation points
<code>derr</code>	standard errors of the density (cumulative distribution) estimates
<code>log_likelihood</code>	log likelihood of the density estimates

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), “Multivariate binary discrimination by the kernel method,” *Biometrika*, 63, 413-420.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Li, Q. and J.S. Racine (2003), “Nonparametric estimation of distributions with categorical and continuous data,” *Journal of Multivariate Analysis*, 86, 266-292.

Ouyang, D. and Q. Li and J.S. Racine (2006), “Cross-validation and the estimation of probability distributions with categorical data,” *Journal of Nonparametric Statistics*, 18, 69-100.

Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.

Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), “A class of smooth estimators for discrete distributions,” *Biometrika*, 68, 301-309.

See Also

[npudensbw](#), [density](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we load Giovanni
# Baiocchi's Italian GDP panel (see Italy for details), then create a
# data frame in which year is an ordered factor, GDP is continuous,
# compute bandwidths using likelihood cross-validation, then create a
# grid of data on which the density will be evaluated for plotting
# purposes.

data("Italy")
attach(Italy)

# Compute bandwidths using likelihood cross-validation (default).

bw <- npudensbw(formula=~ordered(year)+gdp)

# At this stage you could use npudens() to do a variety of
# things. Here we compute the npudens() object and place it in fhat.

fhat <- npudens(bws=bw)

# Note that simply typing the name of the object returns some useful
# information. For more info, one can call summary:

summary(fhat)

## Not run:

# Next, we illustrate how to create a grid of 'evaluation data' and feed
# it to the perspective plotting routines in R, among others.

# Create an evaluation data matrix

year.seq <- sort(unique(year))
gdp.seq <- seq(1,36,length=50)
```

```

data.eval <- expand.grid(year=year.seq,gdp=gdp.seq)

# Generate the estimated density computed for the evaluation data

fhat <- fitted(npudens(bws=bw, newdata=data.eval))

# Coerce the data into a matrix for plotting with persp()

f <- matrix(fhat, length(unique(year)), 50)

# Next, create a 3D perspective plot of the PDF f, and a 2D
# contour plot.

persp(as.integer(levels(year.seq)), gdp.seq, f, col="lightblue",
      ticktype="detailed", ylab="GDP", xlab="Year", zlab="Density",
      theta=300, phi=50)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

contour(as.integer(levels(year.seq)),
       gdp.seq,
       f,
       xlab="Year",
       ylab="GDP",
       main = "Density Contour Plot",
       col=topo.colors(100))

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Alternatively, you could use the plot() command (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems).

plot(bw)

detach(Italy)

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we load Giovanni
# Baiocchi's Italian GDP panel (see Italy for details), then create a
# data frame in which year is an ordered factor, GDP is continuous,
# compute bandwidths using likelihood cross-validation, then create a
# grid of data on which the density will be evaluated for plotting
# purposes.

data("Italy")
attach(Italy)

data <- data.frame(year=ordered(year), gdp)

```

```
# Compute bandwidths using likelihood cross-validation (default).

bw <- npudensbw(dat=data)

# At this stage you could use npudens() to do a variety of
# things. Here we compute the npudens() object and place it in fhat.

fhat <- npudens(bws=bw)

# Note that simply typing the name of the object returns some useful
# information. For more info, one can call summary:

summary(fhat)

# Next, we illustrate how to create a grid of 'evaluation data' and feed
# it to the perspective plotting routines in R, among others.

# Create an evaluation data matrix

year.seq <- sort(unique(year))
gdp.seq <- seq(1,36,length=50)
data.eval <- expand.grid(year=year.seq,gdp=gdp.seq)

# Generate the estimated density computed for the evaluation data

fhat <- fitted(npudens(edat = data.eval, bws=bw))

# Coerce the data into a matrix for plotting with persp()

f <- matrix(fhat, length(unique(year)), 50)

# Next, create a 3D perspective plot of the PDF f, and a 2D
# contour plot.

persp(as.integer(levels(year.seq)), gdp.seq, f, col="lightblue",
      ticktype="detailed", ylab="GDP", xlab="Year", zlab="Density",
      theta=300, phi=50)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

contour(as.integer(levels(year.seq)),
       gdp.seq,
       f,
       xlab="Year",
       ylab="GDP",
       main = "Density Contour Plot",
       col=topo.colors(100))

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)
```

```
# Alternatively, you could use the plot() command (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems).

plot(bw)

detach(Italy)

# EXAMPLE 2 (INTERFACE=FORMULA): For this example, we load the old
# faithful geyser data and compute the density and distribution
# functions.

library("datasets")
data("faithful")
attach(faithful)

# Note - this may take a few minutes depending on the speed of your
# computer...

bw <- npudensbw(formula=~eruptions+waiting)

summary(bw)

# Plot the density function (<ctrl>-C will interrupt on *NIX systems,
# <esc> will interrupt on MS Windows systems). Note that we use xtrim =
# -0.2 to extend the plot outside the support of the data (i.e., extend
# the tails of the estimate to meet the horizontal axis).

plot(bw, xtrim=-0.2)

# Plot the distribution function (cdf=TRUE) (<ctrl>-C will interrupt)

plot(bw, cdf=TRUE)

detach(faithful)

# EXAMPLE 2 (INTERFACE=DATA FRAME): For this example, we load the old
# faithful geyser data and compute the density and distribution
# functions.

library("datasets")
data("faithful")
attach(faithful)

# Note - this may take a few minutes depending on the speed of your
# computer...

bw <- npudensbw(dat=faithful)

summary(bw)

# Plot the density function (<ctrl>-C will interrupt on *NIX systems,
```

```

# <esc> will interrupt on MS Windows systems). Note that we use xtrim =
# -0.2 to extend the plot outside the support of the data (i.e., extend
# the tails of the estimate to meet the horizontal axis).

plot(bw, xtrim=-0.2)

# Plot the distribution function (cdf=TRUE) (<ctrl>-C will interrupt)

plot(bw, cdf=TRUE)

detach(faithful)

## End(Not run)

```

npudensbw

Kernel Density Bandwidth Selection with Mixed Data Types

Description

npudensbw computes a bandwidth object for a p -variate kernel unconditional density estimator defined over mixed continuous and discrete (unordered, ordered) data using either the normal reference rule-of-thumb, likelihood cross-validation, or least-squares cross validation using the method of Li and Racine (2003).

Usage

```

npudensbw(...)

## S3 method for class 'formula'
npudensbw(formula, data, subset, na.action, call, ...)

## S3 method for class 'NULL'
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          ...)

## S3 method for class 'bandwidth'
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin = TRUE,
          itmax = 10000,
          ftol = 1.19209e-07,
          tol = 1.49012e-08,
          small = 2.22045e-16,
          ...)

```

```
## Default S3 method:
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          ftol,
          tol,
          small,
          bwmethod,
          bwscaling,
          bwtype,
          ckertype,
          ckerorder,
          ukertype,
          okertype,
          ...)
```

Arguments

formula	a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which the function is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The (recommended) default is na.omit .
call	the original function call. This is passed internally by np when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.
dat	a p -variate data frame on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
bws	a bandwidth specification. This can be set as a bandwidth object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in <code>dat</code> . In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.

...	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
bwmethod	a character string specifying the bandwidth selection method. <code>cv.ml</code> specifies likelihood cross-validation, <code>cv.ls</code> specifies least-squares cross-validation, and <code>normal-reference</code> just computes the ‘rule-of-thumb’ bandwidth h_j using the standard formula $h_j = 1.06\sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of the j th continuous variable defined as $\min(\text{standard deviation, interquartile range}/1.349)$, n the number of observations, P the order of the kernel, and l the number of continuous variables. Note that when there exist factors and the normal-reference rule is used, there is zero smoothing of the factors. Defaults to <code>cv.ml</code> .
bwscaling	a logical value that when set to <code>TRUE</code> the supplied bandwidths are interpreted as ‘scale factors’ (c_j), otherwise when the value is <code>FALSE</code> they are interpreted as ‘raw bandwidths’ (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j \sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of the j th continuous variable defined as $\min(\text{standard deviation, interquartile range}/1.349)$, n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j . Defaults to <code>FALSE</code> .
bwtype	character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the bandwidth object. Defaults to <code>fixed</code> . Option summary: <code>fixed</code> : compute fixed bandwidths <code>generalized_nn</code> : compute generalized nearest neighbors <code>adaptive_nn</code> : compute adaptive nearest neighbors
bandwidth.compute	a logical value which specifies whether to do a numerical search for bandwidths or not. If set to <code>FALSE</code> , a bandwidth object will be returned with bandwidths set to those specified in <code>bws</code> . Defaults to <code>TRUE</code> .
ckertype	character string used to specify the continuous kernel type. Can be set as <code>gaussian</code> , <code>epanechnikov</code> , or <code>uniform</code> . Defaults to <code>gaussian</code> .
ckerorder	numeric value specifying kernel order (one of (2, 4, 6, 8)). Kernel order specified along with a <code>uniform</code> continuous kernel type will be ignored. Defaults to 2.
ukertype	character string used to specify the unordered categorical kernel type. Can be set as <code>aitchisonaitken</code> .
okertype	character string used to specify the ordered categorical kernel type. Can be set as <code>wangvanryzin</code> .
nmulti	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.
remin	a logical value which when set as <code>TRUE</code> the search routine restarts from located minima for a minor gain in accuracy. Defaults to <code>TRUE</code> .
itmax	integer number of iterations before failure in the numerical optimization routine. Defaults to 10000.

ftol	tolerance on the value of the cross-validation function evaluated at located minima. Defaults to $1.19e-07$ (FLT_EPSILON).
tol	tolerance on the position of located minima of the cross-validation function. Defaults to $1.49e-08$ (sqrt(DBL_EPSILON)).
small	a small number, at about the precision of the data type used. Defaults to $2.22e-16$ (DBL_EPSILON).

Details

Typical usages are (see below for a complete list of options and also the examples at the end of this help file)

Usage 1: compute a bandwidth object using the formula interface:

```
bw <- npudensbw(~y)
```

Usage 2: compute a bandwidth object using the data frame interface and change the default kernel and order:

```
fhat <- npudensbw(tdat = y, ckertype="epanechnikov", ckerorder=4)
```

npudensbw implements a variety of methods for choosing bandwidths for multivariate (p -variate) distributions defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ ‘generalized product kernels’ that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms (direction set (Powell’s) methods in multidimensions).

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i , when estimating the density at the point x . Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x . Fixed bandwidths are constant over the support of x .

npudensbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the dat parameter. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame dat may be a mix of continuous (default), unordered discrete (to be specified in the data frame dat using [factor](#)), and ordered discrete (to be specified in the data frame dat using [ordered](#)). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see [np](#) for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form \sim data, where data is a series of variables specified by name, separated by the separation character ‘+’. For example, $\sim x + y$ specifies that the bandwidths for the joint distribution of variables x and y are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npudensbw returns a bandwidth object, with the following components:

bw	bandwidth(s), scale factor(s) or nearest neighbours for the data, dat
fval	objective function value at minimum

if bwtype is set to fixed, an object containing bandwidths, of class bandwidth (or scale factors if bwscaling = TRUE) is returned. If it is set to generalized_nn or adaptive_nn, then instead the k th nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored under the component name bw, with each element i corresponding to column i of input data dat.

The functions `predict`, `summary` and `plot` support objects of type bandwidth.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the i th observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting `ftol=.01` and `tol=.01` and conduct multistarting (the default is to restart `min(5, ncol(dat))` times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set `bws=bw` on subsequent calls to this routine where `bw` is the initial bandwidth object). A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, 63, 413-420.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Li, Q. and J.S. Racine (2003), “Nonparametric estimation of distributions with categorical and continuous data,” *Journal of Multivariate Analysis*, 86, 266-292.

Ouyang, D. and Q. Li and J.S. Racine (2006), “Cross-validation and the estimation of probability distributions with categorical data,” *Journal of Nonparametric Statistics*, 18, 69-100.

Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization*, New York: Wiley.

Silverman, B.W. (1986), *Density Estimation*, London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), “A class of smooth estimators for discrete distributions,” *Biometrika*, 68, 301-309.

See Also

[bw.nrd](#), [bw.SJ](#), [hist](#), [npudens](#), [npudist](#)

Examples

```
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we load Giovanni
# Baiocchi's Italian GDP panel (see Italy for details), then create a
# data frame in which year is an ordered factor, GDP is continuous.

data("Italy")
attach(Italy)

data <- data.frame(ordered(year), gdp)

# We compute bandwidths for the kernel density estimator using the
# normal-reference rule-of-thumb, otherwise, we use the defaults (second
# order Gaussian kernel, fixed bandwidths). Note that the bandwidth
# object you compute inherits all properties of the estimator (kernel
# type, kernel order, estimation method) and can be fed directly into
# the plotting utility plot() or into the npudens() function.

bw <- npudensbw(formula=~ordered(year)+gdp, bwmethod="normal-reference")

summary(bw)

## Not run:

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next, specify a value for the bandwidths manually (0.5 for the first
# variable, 1.0 for the second)...

bw <- npudensbw(formula=~ordered(year)+gdp, bws=c(0.5, 1.0),
                bandwidth.compute=FALSE)

summary(bw)
```

```
# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next, if you wanted to use the  $1.06 \sigma n^{-1/(2p+q)}$  rule-of-thumb
# for the bandwidth for the continuous variable and, say, no smoothing
# for the discrete variable, you would use the bwscaling=TRUE argument
# and feed in the values 0 for the first variable (year) and 1.06 for
# the second (gdp). Note that in the printout it reports the 'scale
# factors' rather than the 'bandwidth' as reported in some of the
# previous examples.

bw <- npudensbw(formula=~ordered(year)+gdp, bws=c(0, 1.06),
                bwscaling=TRUE,
                bandwidth.compute=FALSE)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# If you wished to use, say, an eighth order Epanechnikov kernel for the
# continuous variables and specify your own bandwidths, you could do
# that as follows.

bw <- npudensbw(formula=~ordered(year)+gdp, bws=c(0.5, 1.0),
                bandwidth.compute=FALSE,
                ckertype="epanechnikov",
                ckerorder=8)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# If you preferred, say, nearest-neighbor bandwidths and a generalized
# kernel estimator for the continuous variable, you would use the
# bwtype="generalized_nn" argument.

bw <- npudensbw(formula=~ordered(year)+gdp, bwtype = "generalized_nn")

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next, compute bandwidths using likelihood cross-validation, fixed
# bandwidths, and a second order Gaussian kernel for the continuous
# variable (default). Note - this may take a few minutes depending on
```

```
# the speed of your computer.

bw <- npudensbw(formula=~ordered(year)+gdp)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Finally, if you wish to use initial values for numerical search, you
# can either provide a vector of bandwidths as in bws=c(...) or a
# bandwidth object from a previous run, as in

bw <- npudensbw(formula=~ordered(year)+gdp, bws=c(1, 1))

summary(bw)

detach(Italy)

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we load Giovanni
# Baiocchi's Italian GDP panel (see Italy for details), then create a
# data frame in which year is an ordered factor, GDP is continuous.

data("Italy")
attach(Italy)

data <- data.frame(ordered(year), gdp)

# We compute bandwidths for the kernel density estimator using the
# normal-reference rule-of-thumb, otherwise, we use the defaults (second
# order Gaussian kernel, fixed bandwidths). Note that the bandwidth
# object you compute inherits all properties of the estimator (kernel
# type, kernel order, estimation method) and can be fed directly into
# the plotting utility plot() or into the npudens() function.

bw <- npudensbw(dat=data, bwmethod="normal-reference")

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next, specify a value for the bandwidths manually (0.5 for the first
# variable, 1.0 for the second)...

bw <- npudensbw(dat=data, bws=c(0.5, 1.0), bandwidth.compute=FALSE)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...
```

```
Sys.sleep(5)

# Next, if you wanted to use the  $1.06 \sigma n^{-1/(2p+q)}$  rule-of-thumb
# for the bandwidth for the continuous variable and, say, no smoothing
# for the discrete variable, you would use the bwscaling=TRUE argument
# and feed in the values 0 for the first variable (year) and 1.06 for
# the second (gdp). Note that in the printout it reports the 'scale
# factors' rather than the 'bandwidth' as reported in some of the
# previous examples.

bw <- npudensbw(dat=data, bws=c(0, 1.06),
                bwscaling=TRUE,
                bandwidth.compute=FALSE)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# If you wished to use, say, an eighth order Epanechnikov kernel for the
# continuous variables and specify your own bandwidths, you could do
# that as follows.

bw <- npudensbw(dat=data, bws=c(0.5, 1.0),
                bandwidth.compute=FALSE,
                ckertype="epanechnikov",
                ckerorder=8)

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# If you preferred, say, nearest-neighbor bandwidths and a generalized
# kernel estimator for the continuous variable, you would use the
# bwtype="generalized_nn" argument.

bw <- npudensbw(dat=data, bwtype = "generalized_nn")

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Next, compute bandwidths using likelihood cross-validation, fixed
# bandwidths, and a second order Gaussian kernel for the continuous
# variable (default). Note - this may take a few minutes depending on
# the speed of your computer.

bw <- npudensbw(dat=data)
```

```

summary(bw)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Finally, if you wish to use initial values for numerical search, you
# can either provide a vector of bandwidths as in bws=c(...) or a
# bandwidth object from a previous run, as in

bw <- npudensbw(dat=data, bws=c(1, 1))

summary(bw)

detach(Italy)

## End(Not run)

```

npunitest	<i>Kernel Consistent Univariate Density Equality Test with Mixed Data Types</i>
-----------	---

Description

npunitest implements the consistent metric entropy test of Maasoumi and Racine (2002) for two arbitrary, stationary univariate nonparametric densities on common support.

Usage

```

npunitest(data.x = NULL,
          data.y = NULL,
          method = c("integration", "summation"),
          bootstrap = TRUE,
          boot.num = 399,
          bw.x = NULL,
          bw.y = NULL,
          random.seed = 42,
          ...)

```

Arguments

`data.x`, `data.y` common support univariate vectors containing the variables.

`method` a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as `integration` or `summation`. Defaults to `integration`. See ‘Details’ below for important information regarding the use of summation when `data.x` and `data.y` lack common support and/or are sparse.

<code>bootstrap</code>	a logical value which specifies whether to conduct the bootstrap test or not. If set to FALSE, only the statistic will be computed. Defaults to TRUE.
<code>boot.num</code>	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
<code>bw.x, bw.y</code>	numeric (scalar) bandwidths. Defaults to plug-in (see details below).
<code>random.seed</code>	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
<code>...</code>	additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is used since we specify <code>bw</code> as a numeric scalar and not a bandwidth object, and is of interest if you do not desire the default behaviours. To change the defaults, you may specify any of <code>bwscaling</code> , <code>bwtype</code> , <code>ckertype</code> , <code>ckerorder</code> , <code>ukertype</code> , <code>okertype</code> .

Details

`npunitest` computes the nonparametric metric entropy (normalized Hellinger of Granger, Maasoumi and Racine (2004)) for testing equality of two univariate density/probability functions, $D[f(x), f(y)]$. See Maasoumi and Racine (2002) for details. Default bandwidths are of the plug-in variety ([bw.SJ](#) for continuous variables and direct plug-in for discrete variables). The bootstrap is conducted via simple resampling with replacement from the pooled `data.x` and `data.y` (`data.x` only for summation).

The summation version of this statistic can be numerically unstable when `data.x` and `data.y` lack common support or when the overlap is sparse (the summation version involves division of densities while the integration version involves differences, and the statistic in such cases can be reported as exactly 0.5 or 0). Warning messages are produced when this occurs ('integration recommended') and should be heeded.

Numerical integration can occasionally fail when the `data.x` and `data.y` distributions lack common support and/or lie an extremely large distance from one another (the statistic in such cases will be reported as exactly 0.5 or 0). However, in these extreme cases, simple tests will reveal the obvious differences in the distributions and entropy-based tests for equality will be clearly unnecessary.

Value

`npunitest` returns an object of type `unitest` with the following components

<code>Srho</code>	the statistic <code>Srho</code>
<code>Srho.bootstrap</code>	contains the bootstrap replications of <code>Srho</code>
<code>P</code>	the P-value of the statistic
<code>boot.num</code>	number of bootstrap replications
<code>bw.x, bw.y</code>	scalar bandwidths for <code>data.x</code> , <code>data.y</code>

[summary](#) supports object of type `unitest`.

Usage Issues

See the example below for proper usage.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), “A dependence metric for possibly non-linear processes”, *Journal of Time Series Analysis*, 25, 649-669.

Maasoumi, E. and J.S. Racine (2002), “Entropy and Predictability of Stock Market Returns,” *Journal of Econometrics*, 107, 2, pp 291-312.

See Also

[npdeneqtest](#), [npdeptest](#), [npsdeptest](#), [npsymtest](#)

Examples

```
set.seed(1234)
n <- 1000

## Compute the statistic only for data drawn from same distribution

x <- rnorm(n)
y <- rnorm(n)

npunitest(x,y,bootstrap=FALSE)

Sys.sleep(5)

## Conduct the test for this data

npunitest(x,y,boot.num=99)

Sys.sleep(5)

## Conduct the test for data drawn from different distributions having
## the same mean and variance

x <- rchisq(n,df=5)
y <- rnorm(n,mean=5,sd=sqrt(10))
mean(x)
mean(y)
sd(x)
sd(y)

npunitest(x,y,boot.num=99)

Sys.sleep(5)

## Not run:
## Two sample t-test for equality of means
t.test(x,y)
```

```

## F test for equality of variances and asymptotic
## critical values
F <- var(x)/var(y)
qf(c(0.025,0.975),df1=n-1,df2=n-1)

## End(Not run)

## Plot the nonparametric density estimates on the same axes

fx <- density(x)
fy <- density(y)
xlim <- c(min(fx$x,fy$x),max(fx$x,fy$x))
ylim <- c(min(fx$y,fy$y),max(fx$y,fy$y))
plot(fx,xlim=xlim,ylim=ylim,xlab="Data",main="f(x), f(y)")
lines(fy$x,fy$y,col="red")

Sys.sleep(5)

## Test for equality of log(wage) distributions

data(wage1)
attach(wage1)
lwage.male <- lwage[female=="Male"]
lwage.female <- lwage[female=="Female"]

npunitest(lwage.male,lwage.female,boot.num=99)

Sys.sleep(5)

## Plot the nonparametric density estimates on the same axes

f.m <- density(lwage.male)
f.f <- density(lwage.female)
xlim <- c(min(f.m$x,f.f$x),max(f.m$x,f.f$x))
ylim <- c(min(f.m$y,f.f$y),max(f.m$y,f.f$y))
plot(f.m,xlim=xlim,ylim=ylim,
     xlab="log(wage)",
     main="Male/Female log(wage) Distributions")
lines(f.f$x,f.f$y,col="red",lty=2)
rug(lwage.male)
legend(-1,1.2,c("Male","Female"),lty=c(1,2),col=c("black","red"))

detach(wage1)

Sys.sleep(5)

## Conduct the test for data drawn from different discrete probability
## distributions

x <- factor(rbinom(n,2,.5))
y <- factor(rbinom(n,2,.1))

npunitest(x,y,boot.num=99)

```

oecdpanel

*Cross Country Growth Panel***Description**

Cross country GDP growth panel covering the period 1960-1995 used by Liu and Stengos (2000) and Maasoumi, Racine, and Stengos (2007). There are 616 observations in total. `data("oecdpanel")` makes available the dataset "oecdpanel" plus an additional object "bw".

Usage

```
data("oecdpanel")
```

Format

A data frame with 7 columns, and 616 rows. This panel covers 7 5-year periods: 1960-1964, 1965-1969, 1970-1974, 1975-1979, 1980-1984, 1985-1989 and 1990-1994.

A separate local-linear rbandwidth object ('bw') has been computed for the user's convenience which can be used to visualize this dataset using `plot(bw)`.

growth the first column, of type numeric: growth rate of real GDP per capita for each 5-year period

oecd the second column, of type factor: equal to 1 for OECD members, 0 otherwise

year the third column, of type integer

initgdp the fourth column, of type numeric: per capita real GDP at the beginning of each 5-year period

popgro the fifth column, of type numeric: average annual population growth rate for each 5-year period

inv the sixth column, of type numeric: average investment/GDP ratio for each 5-year period

humancap the seventh column, of type numeric: average secondary school enrollment rate for each 5-year period

Source

Thanasis Stengos

References

Liu, Z. and T. Stengos (1999), "Non-linearities in cross country growth regressions: a semiparametric approach," *Journal of Applied Econometrics*, 14, 527-538.

Maasoumi, E. and J.S. Racine and T. Stengos (2007), "Growth and convergence: a profile of distribution dynamics and mobility," *Journal of Econometrics*, 136, 483-508

Examples

```
data("oecdpanel")
attach(oecdpanel)
summary(oecdpanel)
detach(oecdpanel)
```

se

Extract Standard Errors

Description

se is a generic function which extracts standard errors from objects.

Usage

```
se(x)
```

Arguments

x an object for which the extraction of standard errors is meaningful.
... other arguments.

Details

This function provides a generic interface for extraction of standard errors from objects.

Value

Standard errors extracted from the model object x.

Note

This method currently only supports objects from the [np](#) library.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

[fitted](#), [residuals](#), [coef](#), and [gradients](#), for related methods; [np](#) for supported objects.

Examples

```
x <- rnorm(10)
se(npudens(npudensbw(~x)))
```

`uocquantile`*Compute Quantiles*

Description

`uocquantile` is a function which computes quantiles of an unordered, ordered, or continuous variable `x`.

Usage

```
uocquantile(x, prob)
```

Arguments

<code>x</code>	an ordered, unordered or continuous variable.
<code>prob</code>	quantile to compute.

Details

`uocquantile` is a function which computes quantiles of unordered, ordered, or a continuous variable `x`. If `x` is unordered, the mode is returned. If `x` is ordered, the level for which the cumulative distribution is \geq `prob` is returned. If `x` is continuous, [quantile](#) is invoked and the result returned.

Value

A quantile computed from `x`.

Author(s)

Tristen Hayfield <hayfield@mpia.de>, Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

[quantile](#)

Examples

```
x <- rbinom(n = 100, size = 10, prob = 0.5)
uocquantile(x, 0.5)
```

wage1

*Cross-Sectional Data on Wages***Description**

Cross-section wage data consisting of a random sample taken from the U.S. Current Population Survey for the year 1976. There are 526 observations in total. `data("wage1")` makes available the dataset "wage" plus additional objects "bw.all" and "bw.subset".

Usage

```
data("wage1")
```

Format

A data frame with 24 columns, and 526 rows.

Two local-linear rbandwidth objects ('bw.all' and 'bw.subset') have been computed for the user's convenience which can be used to visualize this dataset using `plot(bw.all)`

wage column 1, of type numeric, average hourly earnings

educ column 2, of type numeric, years of education

exper column 3, of type numeric, years potential experience

tenure column 4, of type numeric, years with current employer

nonwhite column 5, of type factor, ="Nonwhite" if nonwhite, "White" otherwise

female column 6, of type factor, ="Female" if female, "Male" otherwise

married column 7, of type factor, ="Married" if Married, "Nonmarried" otherwise

numdep column 8, of type numeric, number of dependents

smsa column 9, of type numeric, =1 if live in SMSA

northcen column 10, of type numeric, =1 if live in north central U.S

south column 11, of type numeric, =1 if live in southern region

west column 12, of type numeric, =1 if live in western region

construc column 13, of type numeric, =1 if work in construc. indus.

ndurman column 14, of type numeric, =1 if in nondur. manuf. indus.

trcommpu column 15, of type numeric, =1 if in trans, commun, pub ut

trade column 16, of type numeric, =1 if in wholesale or retail

services column 17, of type numeric, =1 if in services indus.

profserv column 18, of type numeric, =1 if in prof. serv. indus.

profocc column 19, of type numeric, =1 if in profess. occupation

clerocc column 20, of type numeric, =1 if in clerical occupation

servocc column 21, of type numeric, =1 if in service occupation

lwage column 22, of type numeric, $\log(\text{wage})$

expersq column 23, of type numeric, exper^2

tenursq column 24, of type numeric, tenure^2

Source

Jeffrey M. Wooldridge

References

Wooldridge, J.M. (2000), *Introductory Econometrics: A Modern Approach*, South-Western College Publishing.

Examples

```
data("wage1")
attach(wage1)
summary(wage1)
detach(wage1)
```

Index

*Topic **datasets**

cps71, 4
Engel95, 5
Italy, 8
oecdpanel, 164
wage1, 167

*Topic **instrument**

npregiv, 113
npregivderiv, 118

*Topic **nonparametric**

b.star, 2
gradients, 7
npcdens, 12
npcdensbw, 20
npcmstest, 27
npconmode, 31
npdeneqtest, 36
npdeptest, 38
npindex, 41
npindexbw, 50
npksum, 56
npplot, 64
npplreg, 76
npplregbw, 82
npqcmstest, 89
npqreg, 92
npreg, 96
npregbw, 106
npscoef, 123
npscoefbw, 127
npsdeptest, 133
npseed, 136
npsigtest, 137
npsymtest, 141
npudens, 144
npudensbw, 151
npunitest, 160
se, 165
uocquantile, 166

*Topic **package**

np, 9

*Topic **univar**

b.star, 2
npdeptest, 38
npsdeptest, 133
npsymtest, 141
npunitest, 160
uocquantile, 166

adaptIntegrate, 40, 135
as.data.frame, 13, 22, 27, 32, 42, 51, 57, 69,
77, 84, 89, 93, 97, 107, 124, 129,
138, 145, 152

b.star, 2, 142
boot, 3, 142
bw(oecdpanel), 164
bw.all(wage1), 167
bw.nrd, 26, 126, 156
bw.SJ, 26, 126, 142, 156, 161
bw.subset(wage1), 167

cbind, 10, 14, 25, 29, 32, 37, 44, 53, 59, 72,
79, 86, 91, 94, 99, 110, 126, 132,
139, 146, 155

coef, 8, 43, 53, 78, 125, 165

cps71, 4

cubature, 40, 135

data.frame, 10, 14, 25, 29, 32, 37, 44, 53, 59,
72, 79, 86, 91, 94, 99, 110, 126, 132,
139, 146, 155

density, 147

Engel95, 5

factor, 10, 14, 24, 58, 78, 85, 98, 109, 142,
146, 154

fitted, 8, 14, 32, 43, 78, 94, 98, 125, 146, 165

- glm, 27
- gradients, 7, 8, 14, 43, 94, 98, 165
- hist, 26, 126, 156
- Italy, 8
- lm, 10, 27, 29
- loess, 99
- mode, 32
- na.action, 22, 51, 57, 84, 107, 129, 152
- na.fail, 22, 51, 57, 84, 107, 129, 152
- na.omit, 22, 51, 57, 84, 108, 129, 152
- nlm, 58
- np, 8, 9, 14, 22, 24, 51, 58, 78, 84, 85, 98, 108, 109, 129, 131, 136, 137, 142, 146, 152, 154, 165
- npcdens, 12
- npcdensbw, 13, 20, 31, 32, 69, 93, 136
- npcdist, 14
- npcdist (npcdens), 12
- npcmstest, 27, 136
- npconmode, 31
- npdeneqtest, 36, 40, 135, 143, 162
- npdeptest, 37, 38, 135, 143, 162
- npindex, 41
- npindexbw, 42, 50, 69
- npksum, 10, 53, 56, 115, 121
- npplot, 10, 64
- npplreg, 76
- npplregbw, 69, 77, 82, 136
- npqcmstest, 89, 136
- npqreg, 92, 136
- npreg, 8, 10, 79, 86, 96, 111, 116, 121, 122, 133
- npregbw, 28, 58, 69, 77, 79, 84, 86, 90, 97, 106, 133, 136, 138
- npregiv, 113, 122
- npregivderiv, 116, 118
- npscoef, 123
- npscoefbw, 69, 124, 126, 127
- npsdeptest, 37, 40, 133, 143, 162
- npseed, 136
- npsigtest, 136, 137
- npsymtest, 37, 40, 135, 141, 162
- npudens, 15, 26, 126, 144, 156
- npudensbw, 69, 126, 136, 145, 147, 151
- npudist, 26, 126, 144, 146, 156
- npudist (npudens), 144
- npunitest, 37, 40, 135, 143, 160
- numeric, 10, 38, 134, 142
- oecdpanel, 164
- optim, 52, 53, 114, 115, 120, 130
- ordered, 10, 14, 24, 58, 78, 85, 98, 109, 146, 154
- plot, 9, 10, 14, 25, 43, 53, 64, 71, 94, 98, 110, 125, 132, 146, 155, 164, 167
- plot.default, 70
- predict, 14, 25, 43, 53, 78, 94, 98, 110, 125, 132, 146, 155
- quantile, 94, 166
- residuals, 8, 43, 78, 98, 125, 165
- rq, 89
- se, 8, 14, 43, 94, 98, 125, 146, 165
- set.seed, 137
- summary, 14, 25, 28, 32, 37, 39, 43, 53, 90, 94, 98, 110, 125, 132, 135, 139, 142, 146, 155, 161
- title, 69
- ts, 134
- uocquantile, 166
- vcov, 43, 44, 78
- wage1, 167