Package 'np'

February 12, 2026

Version 0.60-20

Date 2026-02-11

Imports boot, cubature, methods, quadprog, quantreg, stats

Suggests crs, MASS, logspline, ks, testthat

Title Nonparametric Kernel Smoothing Methods for Mixed Data Types

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

Description 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, <https://www.nserc-crsng.gc.ca/>), the Social Sciences and Humanities Research Council of Canada (SSHRC, <https://www.sshrc-crsh.gc.ca/>), and the Shared Hierarchical Academic Research Computing Network (SHARCNET, <https://sharcnet.ca/>). We would also like to acknowledge the contributions of the GNU GSL authors. In particular, we adapt the GNU GSL B-spline routine gsl_bspline.c adding automated support for quantile knots (in addition to uniform knots), providing missing functionality for derivatives, and for extending the splines beyond their endpoints.

License GPL

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

BugReports https://github.com/JeffreyRacine/R-Package-np/issues

Repository CRAN

NeedsCompilation yes

Author Jeffrey S. Racine [aut, cre],
Tristen Hayfield [aut]

Date/Publication 2026-02-12 06:10:12 UTC

Contents

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 x k matrix, each column being a data series.
Kn	See footnote c, page 59, Politis and White (2004). Defaults to ceiling(log10(n)).
mmax	See Politis and White (2004). Defaults to ceiling(sqrt(n))+Kn.
Bmax	See Politis and White (2004). Defaults to 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 kx2 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 <tristen.hayfield@gmail.com>, 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")
phi <- model.iv$phi

## 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,phi,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 <tristen.hayfield@gmail.com>, 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$bws`. 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, Lin, and Racine (2013), Li, Ouyang, and Racine (2013), Li and Racine (2003), Li and Racine (2004), Li and Racine (2007), Li and Racine (2010), 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 Truncated Gaussian (x is continuous) $k(z) = (\exp(-z^2/2) - \exp(-b^2/2))/(\text{erf}(b/\sqrt{2})\sqrt{2\pi} - 2b\exp(-b^2/2))$ where $z = (x_i - x)/h$, $b > 0$, $|z| \leq b$ and $h > 0$.

See `nptgauss` for details on modifying b.

Second Order Epanechnikov (x is continuous) $k(z) = 3\left(1 - z^2/5\right)/(4\sqrt{5})$ 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 $((1-\lambda)/2)\lambda^{|x_i - x|}$ 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 Normalised for Unconditional Objects (x is a (discrete) factor) $l(x_i, x, \lambda) = 1/(1 + (c-1)\lambda)$ if $x_i = x$, and $\lambda/(1 + (c-1)\lambda)$ 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.

Li and Racine Normalised for Unconditional Objects (x is a (discrete) ordered factor) $l(x_i, x, \lambda) = (1-\lambda)/(1+\lambda)$ if $|x_i - x| = 0$, and $(1-\lambda)/(1+\lambda)\lambda^{|x_i - x|}$ 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,x2=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 for conditional density and distribution objects we can specify kernels for the left-hand side and right-hand side variables in this manner using `cykertype="gaussian"`, `cxkertype="gaussian"` and `uykertype="aitchisonaitken"`, `uxkertype="aitchisonaitken"`).

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 <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

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

We are grateful to John Fox and Achim Zeleis 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. Lin and J.S. Racine (2013), "Optimal bandwidth selection for nonparametric conditional distribution and quantile functions", Journal of Business and Economic Statistics, 31, 57-65.

Li, Q. and D. Ouyang and J.S. Racine (2013), "Categorical Semiparametric Varying-Coefficient Models," Journal of Applied Econometrics, 28, 551-589.

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.

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.

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.

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 Estimation 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). 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 npcdensbw, or as a $p+q$-vector of bandwidths, with each element i up to $i = q$ corresponding to the bandwidth for column i in tydat, and each element i from $i = q+1$ to $i = p+q$ corresponding to the

bandwidth for column $i - q$ in txdat. 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 bws 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 bwmethod, bwscaling, bwtype, cxkertype, cxkerorder, cykertype, cykerorder, uxkertype, uykertype, oxkertype, oykertype, as described in npcdensbw.

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(bws), typically the environment from which npcdensbw 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 txdat.

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 tydat.

Details

npcdens implements 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. 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:

xbw	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, txdat
ybw	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, tydat
xeval	the evaluation points of the explanatory data
yeval	the evaluation points of the dependent data
condens	estimates of the conditional density at the evaluation points
conderr	standard errors of the conditional density estimates
congrad	if invoked with gradients = TRUE, estimates of the gradients at the evaluation points
congerr	if invoked with gradients = TRUE, standard errors of the gradients at the evaluation points
log_likelihood	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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# Call the plot() function to visualize the results (<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), 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)

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 function.

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)

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 function.

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)

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,
          nmulti,
          remin = TRUE,
          itmax = 10000,
          ftol = 1.490116e-07,
          tol = 1.490116e-04,
          small = 1.490116e-05,
          memfac = 500,
          lbc.dir = 0.5,
          dfc.dir = 3,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          initc.dir = 1.0,
          lbd.dir = 0.1,
          hbd.dir = 1,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          initd.dir = 1.0,
          lbc.init = 0.1,
          hbc.init = 2.0,
          cfac.init = 0.5,
          lbd.init = 0.1,
          hbd.init = 0.9,
          dfac.init = 0.375,
          scale.init.categorical.sample = FALSE,
          transform.bounds = FALSE,
          invalid.penalty = c("baseline","dbmax"),
          penalty.multiplier = 10,
```

```
           ...)

## Default S3 method:
npcdensbw(xdat = stop("data 'xdat' missing"),
           ydat = stop("data 'ydat' missing"),
           bws,
           bandwidth.compute = TRUE,
           nmulti,
           remin,
           itmax,
           ftol,
           tol,
           small,
           memfac,
           lbc.dir,
           dfc.dir,
           cfac.dir,
           initc.dir,
           lbd.dir,
           hbd.dir,
           dfac.dir,
           initd.dir,
           lbc.init,
           hbc.init,
           cfac.init,
           lbd.init,
           hbd.init,
           dfac.init,
           scale.init.categorical.sample,
           transform.bounds,
           invalid.penalty,
           penalty.multiplier,
           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 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 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.

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 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 conbandwidth object returned from a previous invocation, or as a $p+q$-vector of bandwidths, with each element i up to $i = q$ corresponding to the bandwidth for column i in ydat, and each element i from $i = q + 1$ to $i = p + q$ corresponding to the bandwidth for column $i - q$ in xdat. 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 — which method to use to select bandwidths. cv.ml specifies likelihood cross-validation, cv.ls specifies least-squares cross-validation, and normal-reference 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 jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, 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 cv.ml.

bwscaling — a logical value that when set to TRUE the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is FALSE 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(standard deviation, mean absolute deviation/1.4826, 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 FALSE.

bwtype — character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the conbandwidth object. Defaults to fixed. Option summary:
fixed: compute fixed bandwidths
generalized_nn: compute generalized nearest neighbors
adaptive_nn: compute adaptive nearest neighbors

bandwidth.compute — a logical value which specifies whether to do a numerical search for bandwidths or not. If set to FALSE, a conbandwidth object will be returned with bandwidths set to those specified in bws. Defaults to TRUE.

cxkertype — character string used to specify the continuous kernel type for xdat. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.

cxkerorder — numeric value specifying kernel order for xdat (one of (2,4,6,8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.

cykertype — character string used to specify the continuous kernel type for ydat. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.

cykerorder — numeric value specifying kernel order for ydat (one of (2,4,6,8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.

uxkertype — character string used to specify the unordered categorical kernel type. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.

uykertype — character string used to specify the unordered categorical kernel type. Can be set as aitchisonaitken or liracine.

oxkertype — character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin or liracine. Defaults to liracine.

oykertype — character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin or liracine.

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 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 — fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

tol — tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

small — a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

lbc.dir, dfc.dir, cfac.dir, initc.dir
: lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbd.dir, hbd.dir, dfac.dir, initd.dir
: lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbc.init, hbc.init, cfac.init
: lower bound, upper bound, and non-random initial values for scale factors for numeric variables for Powell's algorithm. See Details

lbd.init, hbd.init, dfac.init
: lower bound, upper bound, and non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

scale.init.categorical.sample
: a logical value that when set to TRUE scales lbd.dir, hbd.dir, dfac.dir, and initd.dir by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of numeric variables. See Details

transform.bounds
: a logical value that when set to TRUE applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to FALSE.

invalid.penalty
: a character string specifying the penalty used when the optimizer encounters invalid bandwidths. "baseline" returns a finite penalty based on a baseline objective; "dbmax" returns DBL_MAX. Defaults to "baseline".

penalty.multiplier
: a numeric multiplier applied to the baseline penalty when invalid.penalty="baseline". Defaults to 10.

memfac
: The algorithm to compute the least-squares objective function uses a block-based algorithm to eliminate or minimize redundant kernel evaluations. Due to memory, hardware and software constraints, a maximum block size must be imposed by the algorithm. This block size is roughly equal to memfac*10^5 elements. Empirical tests on modern hardware find that a memfac of 500 performs well. If you experience out of memory errors, or strange behaviour for large data sets (>100k elements) setting memfac to a lower value may fix the problem.

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.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying nmulti). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

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 kth 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 ith 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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# 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)
```

---

npcdist *Kernel Conditional Distribution Estimation with Mixed Data Types*

Description

npcdist computes kernel cumulative conditional distribution estimates on $p + q$-variate evaluation data, given a set of training data (both explanatory and dependent) and a bandwidth specification (a

condbandwidth object or a bandwidth vector, bandwidth type, and kernel type) using the method of Li and Racine (2008) and Li, Lin, and Racine (2013). The data may be continuous, discrete (unordered and ordered factors), or some combination thereof.

Usage

```
npcdist(bws, ...)

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

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

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

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

Arguments

bws	a bandwidth specification. This can be set as a condbandwidth object returned from a previous invocation of npcdistbw, or as a $p+q$-vector of bandwidths, with each element i up to $i = q$ corresponding to the bandwidth for column i in tydat, and each element i from $i = q + 1$ to $i = p + q$ corresponding to the bandwidth for column $i - q$ in txdat. 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 bws as a $p+q$-vector and not a condbandwidth object, and you do not desire the default behaviours. To do this, you may specify any of bwmethod, bwscaling, bwtype, cxkertype, cxkerorder, cykertype, cykerorder, uxkertype, oxkertype, oykertype, as described in npcdistbw.
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(bws), typically the environment from which npcdistbw 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 cumulative conditional distributions will be evaluated. By default, evaluation takes place on the data provided by txdat.
eydat	a q-variate data frame of dependent data on which cumulative conditional distributions will be evaluated. By default, evaluation takes place on the data provided by tydat.

Details

npcdist implements a variety of methods for estimating multivariate conditional cumulative 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 cumulative conditional distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative conditional distribution 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

npcdist 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:

xbw	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, txdat
ybw	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, tydat
xeval	the evaluation points of the explanatory data
yeval	the evaluation points of the dependent data

condist	estimates of the conditional cumulative distribution at the evaluation points
conderr	standard errors of the cumulative conditional distribution estimates
congrad	if invoked with gradients = TRUE, estimates of the gradients at the evaluation points
congerr	if invoked with gradients = TRUE, standard errors of the gradients at the evaluation points
log_likelihood	log likelihood of the cumulative conditional distribution 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 <tristen.hayfield@gmail.com>, 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.

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.

Li, Q. and J. Lin and J.S. Racine (2013), "Optimal bandwidth selection for nonparametric conditional distribution and quantile functions", Journal of Business and Economic Statistics, 31, 57-65.

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

```
## Not run:
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we load Giovanni
# Baiocchi's Italian GDP panel (see Italy for details), and compute the
# 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.

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

# Next, compute the condistribution object...

Fhat <- npcdist(bws=bw)

# The object Fhat now contains results such as the estimated cumulative
# conditional distribution function (Fhat$condist) 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)

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
# 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 - 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 <- npcdistbw(xdat=X, ydat=y)
```

```
# Next, compute the condistribution object...

Fhat <- npcdist(bws=bw)

# The object Fhat now contains results such as the estimated cumulative
# conditional distribution function (Fhat$condist) 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)

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 distribution function.

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

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

bw <- npcdistbw(formula=eruptions~waiting)

summary(bw)

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

plot(bw)

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
# cumulative conditional distribution function.

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 <- npcdistbw(xdat=X, ydat=y)

summary(bw)

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

plot(bw)

detach(faithful)


## End(Not run)
```

---

npcdistbw *Kernel Conditional Distribution Bandwidth Selection with Mixed Data Types*

Description

npcdistbw computes a condbandwidth object for estimating a $p + q$-variate kernel conditional cumulative distribution estimator defined over mixed continuous and discrete (unordered xdat, ordered xdat and ydat) data using either the normal-reference rule-of-thumb or least-squares cross validation method of Li and Racine (2008) and Li, Lin and Racine (2013).

Usage

```
npcdistbw(...)

## S3 method for class 'formula'
npcdistbw(formula, data, subset, na.action, call, gdata = NULL,...)

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

## S3 method for class 'condbandwidth'
npcdistbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          gydat = NULL,
          bws,
```

```
            bandwidth.compute = TRUE,
            nmulti,
            remin = TRUE,
            itmax = 10000,
            do.full.integral = FALSE,
            ngrid = 100,
            ftol = 1.490116e-07,
            tol = 1.490116e-04,
            small = 1.490116e-05,
            memfac = 500.0,
            lbc.dir = 0.5,
            dfc.dir = 3,
            cfac.dir = 2.5*(3.0-sqrt(5)),
            initc.dir = 1.0,
            lbd.dir = 0.1,
            hbd.dir = 1,
            dfac.dir = 0.25*(3.0-sqrt(5)),
            initd.dir = 1.0,
            lbc.init = 0.1,
            hbc.init = 2.0,
            cfac.init = 0.5,
            lbd.init = 0.1,
            hbd.init = 0.9,
            dfac.init = 0.375,
            scale.init.categorical.sample = FALSE,
            transform.bounds = FALSE,
            invalid.penalty = c("baseline","dbmax"),
            penalty.multiplier = 10,
            ...)

## Default S3 method:
npcdistbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          gydat,
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          do.full.integral,
          ngrid,
          ftol,
          tol,
          small,
          memfac,
          lbc.dir,
          dfc.dir,
          cfac.dir,
```

```
          initc.dir,
          lbd.dir,
          hbd.dir,
          dfac.dir,
          initd.dir,
          lbc.init,
          hbc.init,
          cfac.init,
          lbd.init,
          hbd.init,
          dfac.init,
          scale.init.categorical.sample,
          transform.bounds,
          invalid.penalty,
          penalty.multiplier,
          bwmethod,
          bwscaling,
          bwtype,
          cxkertype,
          cxkerorder,
          cykertype,
          cykerorder,
          uxkertype,
          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 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 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.
gdata	a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles).
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 on which bandwidth selection will be performed. The data types may be continuous, discrete (ordered factors), or some combination thereof.

gydat — a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles for ydat).

bws — a bandwidth specification. This can be set as a condbandwidth object returned from a previous invocation, or as a $p+q$-vector of bandwidths, with each element i up to $i = q$ corresponding to the bandwidth for column i in ydat, and each element i from $i = q + 1$ to $i = p + q$ corresponding to the bandwidth for column $i - q$ in xdat. 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 — which method to use to select bandwidths. cv.ls specifies least-squares cross-validation (Li, Lin and Racine (2013), and normal-reference 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 jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, 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 cv.ls.

bwscaling — a logical value that when set to TRUE the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is FALSE 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(standard deviation, mean absolute deviation/1.4826, 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 FALSE.

bwtype — character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the condbandwidth object. Defaults to fixed. Option summary:
fixed: compute fixed bandwidths
generalized_nn: compute generalized nearest neighbors
adaptive_nn: compute adaptive nearest neighbors

bandwidth.compute — a logical value which specifies whether to do a numerical search for bandwidths or not. If set to FALSE, a condbandwidth object will be returned with bandwidths set to those specified in bws. Defaults to TRUE.

cxkertype — character string used to specify the continuous kernel type for xdat. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.

cxkerorder numeric value specifying kernel order for xdat (one of (2,4,6,8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.

cykertype character string used to specify the continuous kernel type for ydat. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.

cykerorder numeric value specifying kernel order for ydat (one of (2,4,6,8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.

uxkertype character string used to specify the unordered categorical kernel type. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.

oxkertype character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin or liracine. Defaults to liracine.

oykertype character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin or liracine.

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 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.

do.full.integral
a logical value which when set as TRUE evaluates the moment-based integral on the entire sample.

ngrid integer number of grid points to use when computing the moment-based integral. Defaults to 100.

ftol fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

tol tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

small a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

lbc.dir, dfc.dir, cfac.dir, initc.dir
lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbd.dir, hbd.dir, dfac.dir, initd.dir
lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbc.init, hbc.init, cfac.init
lower bound, upper bound, and non-random initial values for scale factors for numeric variables for Powell's algorithm. See Details

lbd.init, hbd.init, dfac.init
: lower bound, upper bound, and non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

scale.init.categorical.sample
: a logical value that when set to TRUE scales lbd.dir, hbd.dir, dfac.dir, and initd.dir by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of numeric variables. See Details

transform.bounds
: a logical value that when set to TRUE applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to FALSE.

invalid.penalty
: a character string specifying the penalty used when the optimizer encounters invalid bandwidths. "baseline" returns a finite penalty based on a baseline objective; "dbmax" returns DBL_MAX. Defaults to "baseline".

penalty.multiplier
: a numeric multiplier applied to the baseline penalty when invalid.penalty="baseline". Defaults to 10.

memfac
: The algorithm to compute the least-squares objective function uses a block-based algorithm to eliminate or minimize redundant kernel evaluations. Due to memory, hardware and software constraints, a maximum block size must be imposed by the algorithm. This block size is roughly equal to memfac*10^5 elements. Empirical tests on modern hardware find that a memfac of around 500 performs well. If you experience out of memory errors, or strange behaviour for large data sets (>100k elements) setting memfac to a lower value may fix the problem.

Details

npcdistbw 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 xdat, ordered xdat and ydat) 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 cumulative distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative distribution is estimated, x. Fixed bandwidths are constant over the support of x.

npcdistbw 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 frames using factor), and ordered discrete (to be specified in the data

frames using ordered). Data contained in the data frame ydat may be a mix of continuous (default) 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.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying nmulti). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

Value

npcdistbw returns a condbandwidth 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 kth 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 condbandwidth.

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 ith 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 <tristen.hayfield@gmail.com>, 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.

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.

Li, Q. and J. Lin and J.S. Racine (2013), "Optimal bandwidth selection for nonparametric conditional distribution and quantile functions", Journal of Business and Economic Statistics, 31, 57-65.

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

```
## Not run:
# EXAMPLE 1 (INTERFACE=FORMULA): For this example, we compute the
# 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 <- npcdistbw(formula=gdp~ordered(year))

# The object bw can be used for further estimation using
# npcdist(), 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)

# EXAMPLE 1 (INTERFACE=DATA FRAME): For this example, we compute the
# 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 <- npcdistbw(xdat=ordered(year), ydat=gdp)

# The object bw can be used for further estimation using npcdist(),
# 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.
ydat	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of xdat.
distribution	a character string used to specify the method of estimating the distribution of the statistic to be calculated. bootstrap will conduct bootstrapping. asymptotic will use the normal distribution. Defaults to bootstrap.
boot.method	a character string used to specify the bootstrap method. iid will generate independent identically distributed draws. wild will use a wild bootstrap. wild-rademacher will use a wild bootstrap with Rademacher variables. Defaults to iid.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
pivot	a logical value specifying whether the statistic should be normalised such that it approaches $N(0,1)$ in distribution. Defaults to TRUE.
density.weighted	a logical value specifying whether the statistic should be weighted by the density of xdat. Defaults to TRUE.

random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
...	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 bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype, as described in npregbw. This is necessary if you specify bws 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 Jn or In depending on the value of pivot:

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

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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# 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 npcmstest(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.
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(bws), typically the environment from which npcdensbw 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 (conditioning data) used to calculate the regression estimators. Defaults to the training data used to compute the bandwidth object.
tydat	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.
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 (outcomes) of the dependent variable. By default, evaluation takes place on the data provided by tydat.

Value

npconmode returns a conmode object with the following components:

conmode	a vector of type factor (or ordered factor) containing the conditional mode at each evaluation point

condens	a vector of numeric type containing the modal density estimates at each evaluation point
xeval	a data frame of evaluation points
yeval	a vector of type factor (or ordered factor) containing the actual outcomes, or NA if not provided
confusion.matrix	the confusion matrix or NA if outcomes are not available
CCR.overall	the overall correct classification ratio, or NA if outcomes are not available
CCR.byoutcome	a numeric vector containing the correct classification ratio by outcome, or NA if outcomes are not available
fit.mcfadden	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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# 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)
```

---

npcopula *Kernel Copula Estimation with Mixed Data Types*

Description

npcopula implements the nonparametric mixed data kernel copula approach of Racine (2015) for an arbitrary number of dimensions

Usage

```
npcopula(bws,
         data,
         u = NULL,
         n.quasi.inv = 1000,
         er.quasi.inv = 1)
```

Arguments

bws	an unconditional joint distribution (npudistbw) or joint density (npudensbw) bandwidth object (if bws is delivered via npudistbw the copula distribution is

estimated, while if bws is delivered via npudensbw the copula density is estimated)

data — a data frame containing variables used to construct bws

u — an optional matrix of real numbers lying in [0,1], each column of which corresponds to the vector of uth quantile values desired for each variable in the copula (otherwise the u values returned are those corresponding to the sample realizations)

n.quasi.inv — number of grid points generated when u is provided in order to compute the quasi-inverse of each marginal distribution (see details)

er.quasi.inv — number passed to extendrange when u is provided specifying the fraction by which the data range should be extended when constructing the grid used to compute the quasi-inverse (see details)

Details

npcopula computes the nonparametric copula or copula density using inversion (Nelsen (2006), page 51). For the inversion approach, we exploit Sklar's theorem (Corollary 2.3.7, Nelsen (2006)) to produce copulas directly from the joint distribution function using $C(u,v) = H(F^{-1}(u), G^{-1}(v))$ rather than the typical approach that instead uses $H(x,y) = C(F(x), G(y))$. Whereas the latter requires kernel density estimation on a d-dimensional unit hypercube which necessitates the use of boundary correction methods, the former does not.

Note that if u is provided then expand.grid is called on u. As the dimension increases this can become unwieldy and potentially consume an enormous amount of memory unless the number of grid points is kept very small. Given that computing the copula on a grid is typically done for graphical purposes, providing u is typically done for two-dimensional problems only. Even here, however, providing a grid of length 100 will expand into a matrix of dimension 10000 by 2 which, though not memory intensive, may be computationally burdensome.

The 'quasi-inverse' is computed via Definition 2.3.6 from Nelsen (2006). We compute an equi-quantile grid on the range of the data of length n.quasi.inv/2. We then extend the range of the data by the factor er.quasi.inv and compute an equi-spaced grid of points of length n.quasi.inv/2 (e.g. using the default er.quasi.inv=1 we go from the minimum data value minus $1\times$ the range to the maximum data value plus $1\times$ the range for each marginal). We then take these two grids, concatenate and sort, and these form the final grid of length n.quasi.inv for computing the quasi-inverse.

Note that if u is provided and any elements of (the columns of) u are such that they lie beyond the respective values of F for the evaluation data for the respective marginal, such values are reset to the minimum/maximum values of F for the respective marginal. It is therefore prudent to inspect the values of u returned by npcopula when u is provided.

Note that copula are only defined for data of type numeric or ordered.

Value

npcopula returns an object of type data.frame with the following components

copula — the copula (bandwidth object obtained from npudistbw) or copula density (bandwidth object obtained from npudensbw)

u — the matrix of marginal u values associated with the sample realizations (u=NULL) or those created via expand.grid when u is provided

data — the matrix of marginal quantiles constructed when u is provided (data returned has the same names as data inputted)

Usage Issues

See the example below for proper usage.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Nelsen, R. B. (2006), *An Introduction to Copulas,* Second Edition, Springer-Verlag.

Racine, J.S. (2015), "Mixed Data Kernel Copulas," Empirical Economics, 48, 37-59.

See Also

npudensbw,npudens,npudist

Examples

```
## Not run:
## Example 1: Bivariate Mixed Data

require(MASS)

set.seed(42)

## Simulate correlated Gaussian data (rho(x,y)=0.99)

n <- 1000
n.eval <- 100
rho <- 0.99
mu <- c(0,0)
Sigma <- matrix(c(1,rho,rho,1),2,2)
mydat <- mvrnorm(n=n, mu, Sigma)
mydat <- data.frame(x=mydat[,1],
                    y=ordered(as.integer(cut(mydat[,2],
                      quantile(mydat[,2],seq(0,1,by=.1)),
                      include.lowest=TRUE))-1))
q.min <- 0.0
q.max <- 1.0
grid.seq <- seq(q.min,q.max,length=n.eval)
grid.dat <- cbind(grid.seq,grid.seq)

## Estimate the copula (bw object obtained from npudistbw())

bw.cdf <- npudistbw(~x+y,data=mydat)
```

```
copula <- npcopula(bws=bw.cdf,data=mydat,u=grid.dat)

## Plot the copula


contour(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
        xlab="u1",
        ylab="u2",
        main="Copula Contour")

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
      ticktype="detailed",
      xlab="u1",
      ylab="u2",
      zlab="Copula",zlim=c(0,1))

## Plot the empirical copula

copula.emp <- npcopula(bws=bw.cdf,data=mydat)

plot(copula.emp$u1,copula.emp$u2,
     xlab="u1",
     ylab="u2",
     cex=.25,
     main="Empirical Copula")

## Estimate the copula density (bw object obtained from npudensbw())

bw.pdf <- npudensbw(~x+y,data=mydat)
copula <- npcopula(bws=bw.pdf,data=mydat,u=grid.dat)

## Plot the copula density

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
      ticktype="detailed",
      xlab="u1",
      ylab="u2",
      zlab="Copula Density")

## Example 2: Bivariate Continuous Data

require(MASS)

set.seed(42)

## Simulate correlated Gaussian data (rho(x,y)=0.99)

n <- 1000
n.eval <- 100
rho <- 0.99
mu <- c(0,0)
Sigma <- matrix(c(1,rho,rho,1),2,2)
mydat <- mvrnorm(n=n, mu, Sigma)
```

```
mydat <- data.frame(x=mydat[,1],y=mydat[,2])

q.min <- 0.0
q.max <- 1.0
grid.seq <- seq(q.min,q.max,length=n.eval)
grid.dat <- cbind(grid.seq,grid.seq)

## Estimate the copula (bw object obtained from npudistbw())

bw.cdf <- npudistbw(~x+y,data=mydat)
copula <- npcopula(bws=bw.cdf,data=mydat,u=grid.dat)

## Plot the copula

contour(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
        xlab="u1",
        ylab="u2",
        main="Copula Contour")

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
      ticktype="detailed",
      xlab="u1",
      ylab="u2",
      zlab="Copula",
      zlim=c(0,1))

## Plot the empirical copula

copula.emp <- npcopula(bws=bw.cdf,data=mydat)

plot(copula.emp$u1,copula.emp$u2,
     xlab="u1",
     ylab="u2",
     cex=.25,
     main="Empirical Copula")

## Estimate the copula density (bw object obtained from npudensbw())

bw.pdf <- npudensbw(~x+y,data=mydat)
copula <- npcopula(bws=bw.pdf,data=mydat,u=grid.dat)

## Plot the copula density

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
      ticktype="detailed",
      xlab="u1",
      ylab="u2",
      zlab="Copula Density")

## 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

x, y	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).
bw.x, bw.y	optional bandwidth objects for x, y
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.
...	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 bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype.

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

Tn	the (standardized) statistic Tn
In	the (unstandardized) statistic In
Tn.bootstrap	contains the bootstrap replications of Tn

In.bootstrap	contains the bootstrap replications of In
Tn.P	the P-value of the Tn statistic
In.P	the P-value of the In statistic
boot.num	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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
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)
```

```
## 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 `integration` or `summation` (see below for details). Defaults to `integration`.
`bootstrap`	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`.
`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

Srho	the statistic Srho
Srho.bootstrap.vec	contains the bootstrap replications of Srho
P	the P-value of the Srho statistic
bootstrap	a logical value indicating whether bootstrapping was performed
boot.num	number of bootstrap replications
bw.data.x	the numeric bandwidth for data.x marginal density
bw.data.y	the numeric bandwidth for data.y marginal density
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 <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly nonlinear 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

```
## Not run:
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(y,y.fit,boot.num=99,method="summation")

## 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(y,y.fit,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,
        y.eval = FALSE,
        ...)

## 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

bws	a bandwidth specification. This can be set as a sibandwidth object returned from an invocation of npindexbw, or as a vector of parameters (beta) with each element i corresponding to the coefficient for column i in txdat where the first element is normalized to 1, and a scalar bandwidth (h).
gradients	a logical value indicating that you want gradients and the asymptotic covariance matrix for beta computed and returned in the resulting singleindex object. Defaults to FALSE.
residuals	a logical value indicating that you want residuals computed and returned in the resulting singleindex object. Defaults to FALSE.
errors	a logical value indicating that you want (bootstrapped) standard errors for the conditional mean, gradients (when gradients=TRUE is set), and average gradients (when gradients=TRUE is set), computed and returned in the resulting singleindex object. Defaults to FALSE.
boot.num	an integer specifying the number of bootstrap replications to use when performing standard error calculations. Defaults to 399.
...	additional arguments supplied to specify the parameters to the sibandwidth S3 method, which is called during estimation.
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(bws), typically the environment from which npindexbw was called.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
y.eval	If newdata contains dependent data and y.eval = TRUE, np will compute goodness of fit statistics on these data and return them. Defaults to FALSE.
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 (Doksum and Samarov (1995))

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 <tristen.hayfield@gmail.com>, 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.

Doksum, K. and A. Samarov (1995), "Nonparametric estimation of global functionals and a measure of the explanatory power of covariates regression," The Annals of Statistics, 23 1443-1473.

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

```
## Not run:
# 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)

# 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="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)

# 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 $Y = G(X\beta) + \epsilon$. For continuous Y, the approach is that of Hardle, Hall and Ichimura (1993) which jointly minimizes a least-squares cross-validation function with respect to the parameters and bandwidth. For binary Y, a likelihood-based cross-validation approach is employed which jointly maximizes a likelihood cross-validation function with respect to the parameters and bandwidth. 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 environment(formula), 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.
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 xdat.
bws	a bandwidth specification. This can be set as a singleindexbandwidth object returned from an invocation of npindexbw, or as a vector of parameters (beta) with each element i corresponding to the coefficient for column i in xdat 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" (Ichimura (1993)) or "kleinspady" (Klein and Spady (1993)). Defaults to ichimura.
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,ncol(xdat)).
random.seed	an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.

bandwidth.compute
: 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 bws. Defaults to TRUE.

optim.method
: method used by optim for minimization of the objective function. See ?optim 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 optim. Defaults to 100.

optim.abstol
: the absolute convergence tolerance used by optim. Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to .Machine$double.eps.

optim.reltol
: relative convergence tolerance used by optim. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step. Defaults to sqrt(.Machine$double.eps), typically about 1e-8.

optim.maxit
: maximum number of iterations used by optim. Defaults to 500.

only.optimize.beta
: signals the routine to only minimize the objective function with respect to beta

...
: additional arguments supplied to specify the parameters to the sibandwidth S3 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:

bw	bandwidth(s), scale factor(s) or nearest neighbours for the data, xdat
beta	coefficients of the model
fval	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 kth 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 ith 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 <tristen.hayfield@gmail.com>, 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.

Hardle, W. and P. Hall and H. Ichimura (1993), "Optimal Smoothing in Single-Index Models," The Annals of Statistics, 21, 157-178.

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

```
## Not run:
# 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)

# 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 = names(ALL_OPERATORS),
       permutation.operator = names(PERMUTATION_OPERATORS),
       compute.score = FALSE,
       compute.ocg = FALSE,
       return.kernel.weights = 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,
        permutation.operator,
        compute.score,
        compute.ocg,
        return.kernel.weights,
        ...)
```

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 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.
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 na.action setting of options, and is na.fail if that is unset. The (recommended) default is na.omit.
...	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 ith kernel weight is applied to the ith 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 tydat 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 exdat 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. Operators scale results by factors of h or $1/h$ where appropriate. Defaults to `normal` and applies to all elements in a multivariate object. See details.

`permutation.operator` a string which can have a value of `none`, `normal`, `derivative`, or `integral`. If set to something other than `none` (the default), then a separate result will be returned for each term in the product kernel, with the operator applied to that term. Permutation operators scale results by factors of h or $1/h$ where appropriate. This is useful for computing gradients, for example.

`compute.score` a logical specifying whether or not to return the score (the 'grad h' terms) for each dimension in addition to the kernel sum. Cannot be `TRUE` if a permutation operator other than `"none"` is selected.

`compute.ocg` a logical specifying whether or not to return a separate result for each unordered and ordered dimension, where the product kernel term for that dimension is evaluated at an appropriate reference category. This is used primarily in `np` to compute ordered and unordered categorical gradients. See details.

`return.kernel.weights` a logical specifying whether or not to return the matrix of generalized product kernel weights. Defaults to `FALSE`. See details.

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 W_j represents a row vector extracted from W. 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).

The option operator= can be used to 'mix and match' operator strings to create a 'hybrid' kernel provided they match the dimension of the data. For example, for a two-dimensional data frame of numeric datatypes, operator=c("normal","derivative") will use the normal (i.e. PDF) kernel for variable one and the derivative of the PDF kernel for variable two. Please note that applying operators will scale the results by factors of h or $1/h$ where appropriate.

The option permutation.operator= can be used to 'mix and match' operator strings to create a 'hybrid' kernel, in addition to the kernel sum with no operators applied, one for each continuous dimension in the data. For example, for a two-dimensional data frame of numeric datatypes, permutation.operator=c("derivative") will return the usual kernel sum as if operator = c("normal","normal") in the ksum member, and in the p.ksum member, it will return kernel sums for operator = c("derivative","normal"), and operator = c("normal","derivative"). This makes the computation of gradients much easier.

The option compute.score= can be used to compute the gradients with respect to h in addition to the normal kernel sum. Like permutations, the additional results are returned in the p.ksum. This option does not work in conjunction with permutation.operator.

The option compute.ocg= works much like permutation.operator, but for discrete variables. The kernel is evaluated at a reference category in each dimension: for ordered data, the next lowest category is selected, except in the case of the lowest category, where the second lowest category is selected; for unordered data, the first category is selected. These additional data are returned in the p.ksum member. This option can be set simultaneously with permutation.operator.

The option return.kernel.weights=TRUE returns a matrix of dimension 'number of training observations' by 'number of evaluation observations' and contains only the generalized product kernel weights ignoring all other objects and options that may be provided to npksum (e.g. bandwidth.divide=TRUE will be ignored, etc.). Summing the columns of the weight matrix and dividing by 'number of training observations' times the product of the bandwidths (i.e. colMeans(foo$kw)/prod(h)) would produce the kernel estimator of a (multivariate) density (operator="normal") or multivariate cumulative distribution (operator="integral").

Value

npksum returns a npkernelsum object with the following components:

eval the evaluation points

ksum	the sum at the evaluation points
kw	the kernel weights (when return.kernel.weights=TRUE is specified)

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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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))

# 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. 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)))

bw <- npreg(error~age)

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

ksum

# 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
# npcmstest() routine in this package is constructed purely from calls
# to npksum(). You can type npcmstest to see the npcmstest()
# 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 = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       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("band","bar"),
        plot.errors.bar = c("|","I"),
        plot.errors.bar.num = min(neval,25),
        plot.bxp = FALSE,
        plot.bxp.out = TRUE,
        plot.par.mfrow = 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,
       gradients = FALSE,
       common.scale = TRUE,
       perspective = TRUE,
       main = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       tau = 0.5,
```

```
        view = c("rotate","fixed"),
        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("band","bar"),
        plot.errors.bar = c("|","I"),
        plot.errors.bar.num = min(neval,25),
        plot.bxp = FALSE,
        plot.bxp.out = TRUE,
        plot.par.mfrow = 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 = NULL,
        type = NULL,
        border = NULL,
        cex.axis = NULL,
        cex.lab = NULL,
        cex.main = NULL,
        cex.sub = NULL,
        col = NULL,
        ylab = NULL,
        xlab = NULL,
        zlab = NULL,
        sub = NULL,
        ylim = NULL,
        xlim = NULL,
        zlim = NULL,
        lty = NULL,
```

```
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       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("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = 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 = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
```

```
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       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("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = 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 = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
```

```
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       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("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = TRUE,
       ...,
       random.seed)

## S3 method for class 'sibandwidth'
npplot(bws,
       xdat,
       ydat,
       data = NULL,
       common.scale = TRUE,
       gradients = FALSE,
       main = NULL,
       type = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       lty = NULL,
       lwd = 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("band","bar"),
plot.errors.bar = c("|","I"),
plot.errors.bar.num = NULL,
plot.par.mfrow = TRUE,
...,
random.seed)
```

Arguments

bws	a bandwidth specification. This should be a bandwidth object returned from an invocation of npudensbw, npcdensbw, npregbw, npplregbw, npindexbw, or npscoefbw.
...	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 as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(bws), 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 xq corresponds to the ith column of txdat. Defaults to the median (0.5). See details.
yq	a numeric q-vector of quantiles. Each element i of yq corresponds to the ith column of tydat. 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 zq corresponds to the ith column of tzdat. 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 xtrim corresponds to the ith column of txdat. Defaults to 0.0. See details.
ytrim	a numeric q-vector of quantiles. Each element i of ytrim corresponds to the ith column of tydat. Defaults to 0.0. See details.
ztrim	a numeric q-vector of quantiles. Each element i of ztrim corresponds to the ith column of tzdat. 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.
gradients	a logical value specifying whether gradients should be plotted (if possible). Defaults to FALSE.
main	optional title, see title. Defaults to NULL.
sub	optional subtitle, see sub. Defaults to NULL.
type	optional character indicating the type of plotting; actually any of the types as in plot.default. Defaults to NULL.
border	optional character indicating the border of plotting; actually any of the borders as in plot.default. Defaults to NULL.
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex.
cex.lab	The magnification to be used for x and y labels relative to the current setting of cex.
cex.main	The magnification to be used for main titles relative to the current setting of cex.
cex.sub	The magnification to be used for sub-titles relative to the current setting of cex.
col	optional character indicating the color of plotting; actually any of the colours as in plot.default. Defaults to NULL.
ylab	optional character indicating the y axis label of plotting; actually any of the ylabs as in plot.default. Defaults to NULL.
xlab	optional character indicating the x axis label of plotting; actually any of the xlabs as in plot.default. Defaults to NULL.
zlab	optional character indicating the z axis label of plotting; actually any of the zlabs as in plot.default. Defaults to NULL.
ylim	optional a two-element numeric vector of the minimum and maximum y plotting limits. Defaults to NULL.
xlim	a two-element numeric vector of the minimum and maximum x plotting limits. Defaults to NULL.
zlim	a two-element numeric vector of the minimum and maximum z plotting limits. Defaults to NULL.
lty	a numeric value indicating the line type of plotting; actually any of the ltys as in plot.default. Defaults to 1.
lwd	a numeric value indicating the width of the line of plotting; actually any of the lwds as in plot.default. Defaults to 1.
theta	a numeric value specifying the starting azimuthal angle of the perspective plot. Defaults to 0.0.
phi	a numeric value specifying the starting zenith angle of the perspective plot. Defaults to 10.0.
tau	a numeric value specifying the τth quantile is desired when plotting quantile regressions. Defaults to 0.5.

`view` a character string used to specify the viewing mode of the perspective plot. Can be set as `rotate` or `fixed`. Defaults to `rotate`.

`plot.behavior` a character string used to specify the net behavior of `npplot`. Can be set as `plot`, `plot-data` or `data`. Defaults to `plot`. See value.

`plot.errors.method`
a character string used to specify the method to calculate errors. Can be set as `none`, `bootstrap`, or `asymptotic`. Defaults to `none`.

`plot.errors.boot.method`
a character string used to specify the bootstrap method. Can be set as `inid`, `fixed`, or `geom` (see below for details). Defaults to `inid`.

`plot.errors.boot.blocklen`
an integer used to specify the block length b for the `fixed` or `geom` bootstrap (see below for details).

`plot.errors.boot.num`
an integer used to specify the number of bootstrap samples to use for the calculation of errors. Defaults to `399`.

`plot.errors.center`
a character string used to specify where to center the errors on the plot(s). Can be set as `estimate` or `bias-corrected`. Defaults to `estimate`.

`plot.errors.type`
a character string used to specify the type of error to calculate. Can be set as `standard` or `quantiles`. Defaults to `standard`.

`plot.errors.quantiles`
a numeric vector specifying the quantiles of the statistic to calculate for the purpose of error plotting. Defaults to `c(0.025,0.975)`.

`plot.errors.style`
a character string used to specify the style of error plotting. Can be set as `band` or `bar`. Defaults to `band`. Bands are not drawn for discrete variables.

`plot.errors.bar`
a character string used to specify the error bar shape. Can be set as `|` (vertical bar character) for a dashed vertical bar, or as `I` for an 'I' shaped error bar with horizontal bounding bars. Defaults to `|`.

`plot.errors.bar.num`
an integer specifying the number of error bars to plot. Defaults to `min(neval,25)`.

`plot.bxp` a logical value specifying whether boxplots should be produced when appropriate. Defaults to `FALSE`.

`plot.bxp.out` a logical value specifying whether outliers should be plotted on boxplots. Defaults to `TRUE`.

`plot.par.mfrow` a logical value specifying whether `par(mfrow=c(,))` should be called before plotting. Defaults to `TRUE`.

`random.seed` 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 jth 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 <tristen.hayfield@gmail.com>, 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="ll", 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 named, 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)
```

`npplreg` *Partially Linear Kernel Regression with Mixed Data Types*

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, y.eval =
FALSE, ...)
```

```
## 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

bws	a bandwidth specification. This can be set as a plbandwidth object returned from an invocation of npplregbw, or as a matrix of bandwidths, where 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.
...	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 regtype, bwmethod, bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype, as described in npregbw.
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(bws), typically the environment from which npplregbw was called.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
y.eval	If newdata contains dependent data and y.eval = TRUE, np will compute goodness of fit statistics on these data and return them. Defaults to FALSE.
txdat	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.
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.
tzdat	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 (Doksum and Samarov (1995))
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 <tristen.hayfield@gmail.com>, 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.

Doksum, K. and A. Samarov (1995), "Nonparametric estimation of global functionals and a measure of the explanatory power of covariates in regression," The Annals of Statistics, 23 1443-1473.

Gao, Q. and L. Liu and J.S. Racine (2015), "A partially linear kernel estimator for categorical data," Econometric Reviews, 34 (6-10), 958-977.

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.

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

```
## Not run:
# 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)

# 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(pl)
```

```
# 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")

## End(Not run)
```

---

npplregbw *Partially Linear Kernel Regression Bandwidth Selection with Mixed Data Types*

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.

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 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.

xdat — 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.

ydat — a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of xdat.

zdat — 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)

bws — a bandwidth specification. This can be set as a plbandwidth object returned from an invocation of npplregbw, or as a matrix of bandwidths, where 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, npplregbw will search for optimal bandwidths using npregbw in the course of calculations. If specified, npplregbw will use the given bandwidths as the starting point for the numerical search for optimal bandwidths, unless you specify bandwidth.compute = FALSE.

... — 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 regtype, bwmethod, bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype, as described in npregbw.

bandwidth.compute — a logical value which specifies whether to do a numerical search for bandwidths or not. If set to FALSE, a plbandwidth object will be returned with bandwidths set to those specified in bws. Defaults to TRUE.

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,ncol(zdat)).

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 1.19e-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

npplregbw 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.

npplregbw 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 y1, linear parametric regressors x1 and x2, and nonparametric regressor z1 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 kth 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 ith 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 ith 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 <tristen.hayfield@gmail.com>, 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.

Gao, Q. and L. Liu and J.S. Racine (2015), "A partially linear kernel estimator for categorical data," Econometric Reviews, 34 (6-10), 958-977.

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.

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

```
## Not run:
# 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)

# 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$yzbw
bw$bw[[1]]$bw <- c(0.17, 0.30)
```

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

summary(bw)

## End(Not run)
```

---

npqcmstest *Kernel Consistent Quantile Regression Model Specification Test with Mixed Data Types*

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 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 rq. Important: the call to rq must have the argument model=TRUE or npqcmstest will not work.
xdat	a p-variate data frame of explanatory data (training data) used to calculate the quantile regression estimators.

ydat	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of xdat.
tau	a numeric value specifying the τth quantile is desired
distribution	a character string used to specify the method of estimating the distribution of the statistic to be calculated. bootstrap will conduct bootstrapping. asymptotic will use the normal distribution. Defaults to bootstrap.
bwydat	a character string used to specify the left hand side variable used in bandwidth selection. "varepsilon" uses $1-\tau, -\tau$ for ydat while "y" will use y. Defaults to "y".
boot.method	a character string used to specify the bootstrap method. iid will generate independent identically distributed draws. wild will use a wild bootstrap. wild-rademacher will use a wild bootstrap with Rademacher variables. Defaults to iid.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
pivot	a logical value specifying whether the statistic should be normalised such that it approaches $N(0,1)$ in distribution. Defaults to TRUE.
density.weighted	a logical value specifying whether the statistic should be weighted by the density of xdat. Defaults to TRUE.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
...	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 bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype, as described in npregbw. This is necessary if you specify bws 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 Jn or In depending on the value of pivot:

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

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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# 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 methods of Li and Racine (2008) and Li, Lin and Racine (2013). A bandwidth specification can be a condbandwidth 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 'condbandwidth'
npqreg(bws,
```

```
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        exdat,
        tau = 0.5,
        gradients = FALSE,
        ftol = 1.490116e-07,
        tol = 1.490116e-04,
        small = 1.490116e-05,
        itmax = 10000,
        lbc.dir = 0.5,
        dfc.dir = 3,
        cfac.dir = 2.5*(3.0-sqrt(5)),
        initc.dir = 1.0,
        lbd.dir = 0.1,
        hbd.dir = 1,
        dfac.dir = 0.25*(3.0-sqrt(5)),
        initd.dir = 1.0,
        ...)

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

Arguments

bws	a bandwidth specification. This can be set as a condbandwidth object returned from an invocation of npcdistbw, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in txdat. 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 τth 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 bwmethod, bwscaling, bwtype, cxkertype, cxkerorder, cykertype, cykerorder, uxkertype, uykertype, oxkertype, oykertype, as described in npcdistbw.
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(bws), typically the environment from which npcdistbw 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 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.

gradients — [currently not supported] a logical value indicating that you want gradients computed and returned in the resulting npregression object. Defaults to FALSE.

itmax — integer number of iterations before failure in the numerical optimization routine. Defaults to 10000.

ftol — fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

tol — tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

small — a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

lbc.dir, dfc.dir, cfac.dir, initc.dir — lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbd.dir, hbd.dir, dfac.dir, initd.dir — lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

Details

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying nmulti). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

Value

npqreg returns a npqregression object. The generic functions fitted (or quantile), se, predict (when using predict you must add the argument tau= to generate predictions other than the median), 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 τth 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 <tristen.hayfield@gmail.com>, 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.

Li, Q. and J. Lin and J.S. Racine (2013), "Optimal Bandwidth Selection for Nonparametric Conditional Distribution and Quantile Functions", Journal of Business and Economic Statistics, 31, 57-65.

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

See Also

quantreg

Examples

```
## Not run:
# 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 cross-validated bandwidths.  Note - this may take a
# few minutes depending on the speed of your computer...

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

# Note - numerical search for computing the quantiles may 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)

# 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 <- npcdistbw(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)
```

`npquantile` *Kernel Univariate Quantile Estimation*

Description

`npquantile` computes smooth quantiles from a univariate unconditional kernel cumulative distribution estimate given data and, optionally, a bandwidth specification i.e. a `dbandwidth` object using the bandwidth selection method of Li, Li and Racine (2017).

Usage

```
npquantile(x = NULL,
           tau = c(0.01,0.05,0.25,0.50,0.75,0.95,0.99),
           num.eval = 10000,
           bws = NULL,
           f = 1,
           ...)
```

Arguments

`x`	a univariate vector of type `numeric` containing sample realizations (training data) used to estimate the cumulative distribution (must be the same training data used to compute the bandwidth object `bws` passed in).
`tau`	an optional vector containing the probabilities for quantile(s) to be estimated (must contain numbers in $[0, 1]$). Defaults to `c(0.01,0.05,0.25,0.50,0.75,0.95,0.99)`.
`num.eval`	an optional integer specifying the length of the grid on which the quasi-inverse is computed. Defaults to `10000`.
`bws`	an optional `dbandwidth` specification (if already computed avoid unnecessary computation inside `npquantile`). This must be set as a `dbandwidth` object returned from an invocation of `npudistbw`. If not provided `npudistbw` is invoked with optional arguments passed via `...`.

f — an optional argument fed to extendrange. Defaults to 1. See ?extendrange for details.

... — additional arguments supplied to specify the bandwidth type, kernel types, bandwidth selection methods, and so on. See ?npudistbw for details.

Details

Typical usage is

```
x <- rchisq(100,df=10)
npquantile(x)
```

The quantile function q_τ is defined to be the left-continuous inverse of the distribution function $F(x)$, i.e. $q_\tau = \inf\{x : F(x) \geq \tau\}$.

A traditional estimator of q_τ is the τth sample quantile. However, these estimates suffer from lack of efficiency arising from variability of individual order statistics; see Sheather and Marron (1990) and Hyndman and Fan (1996) for methods that interpolate/smooth the order statistics, each of which discussed in the latter can be invoked through quantile via type=j, j=1,...,9.

The function npquantile implements a method for estimating smooth quantiles based on the quasi-inverse of a npudist object where $F(x)$ is replaced with its kernel estimator and bandwidth selection is that appropriate for such objects; see Definition 2.3.6, page 21, Nelsen 2006 for a definition of the quasi-inverse of $F(x)$.

For construction of the quasi-inverse we create a grid of evaluation points based on the function extendrange along with the sample quantiles themselves computed from invocation of quantile. The coarseness of the grid defined by extendrange (which has been passed the option f=1) is controlled by num.eval.

Note that for any value of τ less/greater than the smallest/largest value of $F(x)$ computed for the evaluation data (i.e. that outlined in the paragraph above), the quantile returned for such values is that associated with the smallest/largest value of $F(x)$, respectively.

Value

npquantile returns a vector of quantiles corresponding to tau.

Usage Issues

Cross-validated bandwidth selection is used by default (npudistbw). For large datasets this can be computationally demanding. In such cases one might instead consider a rule-of-thumb bandwidth (bwmethod="normal-reference") or, alternatively, use kd-trees (options(np.tree=TRUE) along with a bounded kernel (ckertype="epanechnikov")), both of which will reduce the computational burden appreciably.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Cheng, M.-Y. and Sun, S. (2006), "Bandwidth selection for kernel quantile estimation," Journal of the Chinese Statistical Association, **44**, 271-295.

Hyndman, R.J. and Fan, Y. (1996), "Sample quantiles in statistical packages," American Statistician, **50**, 361-365.

Li, Q. and J.S. Racine (2017), "Smooth Unconditional Quantile Estimation," Manuscript.

Li, C. and H. Li and J.S. Racine (2017), "Cross-Validated Mixed Datatype Bandwidth Selection for Nonparametric Cumulative Distribution/Survivor Functions," Econometric Reviews, **36**, 970-987.

Nelsen, R.B. (2006), *An Introduction to Copulas,* Second Edition, Springer-Verlag.

Sheather, S. and J.S. Marron (1990), "Kernel quantile estimators," Journal of the American Statistical Association, Vol. 85, No. 410, 410-416.

Yang, S.-S. (1985), "A Smooth Nonparametric Estimator of a Quantile Function," Journal of the American Statistical Association, **80**, 1004-1011.

See Also

`quantile` for various types of sample quantiles; `ecdf` for empirical distributions of which `quantile` is an inverse; `boxplot.stats` and `fivenum` for computing other versions of quartiles; `qlogspline` for logspline density quantiles; `qkde` for alternative kernel quantiles, etc.

Examples

```
## Not run:
## Simulate data from a chi-square distribution
df <- 50
x <- rchisq(100,df=df)

## Vector of quantiles desired
tau <- c(0.01,0.05,0.25,0.50,0.75,0.95,0.99)

## Compute kernel smoothed sample quantiles
npquantile(x,tau)

## Compute sample quantiles using the default method in R (Type 7)
quantile(x,tau)

## True quantiles based on known distribution
qchisq(tau,df=df)

## 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, y.eval =
FALSE, ...)

## 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

bws	a bandwidth specification. This can be set as a rbandwidth object returned from an invocation of npregbw, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in txdat. 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.
...	additional arguments supplied to specify the regression type, bandwidth type, kernel types, training data, and so on, detailed 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(bws), typically the environment from which npregbw was called.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
y.eval	If newdata contains dependent data and y.eval = TRUE, np will compute goodness of fit statistics on these data and return them. Defaults to FALSE.

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.
gradients	a logical value indicating that you want gradients computed and returned in the resulting npregression object. Defaults to FALSE.
residuals	a logical value indicating that you want residuals computed and returned in the resulting npregression object. Defaults to FALSE.

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.

The use of compactly supported kernels or the occurrence of small bandwidths can lead to numerical problems for the local linear estimator when computing the locally weighted least squares solution. To overcome this problem we rely on a form or 'ridging' proposed by Cheng, Hall, and Titterington (1997), modified so that we solve the problem pointwise rather than globally (i.e. only when it is needed).

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 (Doksum and Samarov (1995))
`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 <tristen.hayfield@gmail.com>, 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.

Cheng, M.-Y. and P. Hall and D.M. Titterington (1997), "On the shrinkage of local linear curve estimators," Statistics and Computing, 7, 11-17.

Doksum, K. and A. Samarov (1995), "Nonparametric estimation of global functionals and a measure of the explanatory power of covariates in regression," The Annals of Statistics, 23 1443-1473.

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

```
## Not run:
# 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)
```

```
# 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,
        lbc.dir,
        dfc.dir,
        cfac.dir,
        initc.dir,
        lbd.dir,
        hbd.dir,
        dfac.dir,
```

```
          initd.dir,
          lbc.init,
          hbc.init,
          cfac.init,
          lbd.init,
          hbd.init,
          dfac.init,
          scale.init.categorical.sample,
          transform.bounds = FALSE,
          invalid.penalty = c("baseline","dbmax"),
          penalty.multiplier = 10,
          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 = 1.490116e-07,
        tol = 1.490116e-04,
        small = 1.490116e-05,
        lbc.dir = 0.5,
        dfc.dir = 3,
        cfac.dir = 2.5*(3.0-sqrt(5)),
        initc.dir = 1.0,
        lbd.dir = 0.1,
        hbd.dir = 1,
        dfac.dir = 0.25*(3.0-sqrt(5)),
        initd.dir = 1.0,
        lbc.init = 0.1,
        hbc.init = 2.0,
        cfac.init = 0.5,
        lbd.init = 0.1,
        hbd.init = 0.9,
        dfac.init = 0.375,
        scale.init.categorical.sample = FALSE,
        transform.bounds = FALSE,
```

```
invalid.penalty = c("baseline","dbmax"),
penalty.multiplier = 10,
...)
```

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 environment(formula), 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.
xdat	a p-variate data frame of regressors on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
ydat	a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of xdat.
bws	a bandwidth specification. This can be set as a rbandwidth object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in xdat. 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.
regtype	a character string specifying which type of kernel regression estimator to use. lc specifies a local-constant estimator (Nadaraya-Watson) and ll specifies a local-linear estimator. Defaults to lc.
bwmethod	which method to use to select bandwidths. cv.aic specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and cv.ls specifies least-squares cross-validation. Defaults to cv.ls.
bwscaling	a logical value that when set to TRUE the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is FALSE 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(standard deviation, mean absolute deviation/1.4826, 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 FALSE.

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 fixed. Option summary:
fixed: compute fixed bandwidths
generalized_nn: compute generalized nearest neighbors
adaptive_nn: compute adaptive nearest neighbors

bandwidth.compute — a logical value which specifies whether to do a numerical search for bandwidths or not. If set to FALSE, a rbandwidth object will be returned with bandwidths set to those specified in bws. Defaults to TRUE.

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 liracine.

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,ncol(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 — fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

tol — tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

small — a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

lbc.dir, dfc.dir, cfac.dir, initc.dir — lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbd.dir, hbd.dir, dfac.dir, initd.dir — lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbc.init`, `hbc.init`, `cfac.init`
: lower bound, upper bound, and non-random initial values for scale factors for `numeric` variables for Powell's algorithm. See Details

`lbd.init`, `hbd.init`, `dfac.init`
: lower bound, upper bound, and non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

`scale.init.categorical.sample`
: a logical value that when set to `TRUE` scales `lbd.dir`, `hbd.dir`, `dfac.dir`, and `initd.dir` by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of `numeric` variables. See Details

`transform.bounds`
: a logical value that when set to `TRUE` applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to `FALSE`.

`invalid.penalty`
: a character string specifying the penalty used when the optimizer encounters invalid bandwidths. `"baseline"` returns a finite penalty based on a baseline objective; `"dbmax"` returns `DBL\_MAX`. Defaults to `"baseline"`.

`penalty.multiplier`
: a numeric multiplier applied to the baseline penalty when `invalid.penalty="baseline"`. Defaults to `10`.

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, `y1 ~ x1 + x2` specifies that the bandwidths for the regression

of response y1 and nonparametric regressors 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.

The use of compactly supported kernels or the occurrence of small bandwidths during cross-validation can lead to numerical problems for the local linear estimator when computing the locally weighted least squares solution. To overcome this problem we rely on a form or 'ridging' proposed by Cheng, Hall, and Titterington (1997), modified so that we solve the problem pointwise rather than globally (i.e. only when it is needed).

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying nmulti). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

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 kth 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 ith 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 <tristen.hayfield@gmail.com>, 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.

Cheng, M.-Y. and P. Hall and D.M. Titterington (1997), "On the shrinkage of local linear curve estimators," Statistics and Computing, 7, 11-17.

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

```
## Not run:
# 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)

# 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,
        xeval = NULL,
        alpha = NULL,
        alpha.iter = NULL,
        alpha.max = 1e-01,
        alpha.min = 1e-10,
        alpha.tol = .Machine$double.eps^0.25,
        bw = NULL,
        constant = 0.5,
        iterate.diff.tol = 1.0e-08,
        iterate.max = 1000,
        iterate.Tikhonov = TRUE,
        iterate.Tikhonov.num = 1,
        method = c("Landweber-Fridman","Tikhonov"),
        nmulti = NULL,
        optim.abstol = .Machine$double.eps,
        optim.maxattempts = 10,
        optim.maxit = 500,
        optim.method = c("Nelder-Mead", "BFGS", "CG"),
        optim.reltol = sqrt(.Machine$double.eps),
        p = 1,
        penalize.iteration = TRUE,
        random.seed = 42,
        return.weights.phi = FALSE,
        return.weights.phi.deriv.1 = FALSE,
        return.weights.phi.deriv.2 = FALSE,
        smooth.residuals = TRUE,
        start.from = c("Eyz","EEywz"),
        starting.values  = NULL,
        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.

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.

alpha — a numeric scalar that, if supplied, is used rather than numerically solving for alpha, when using method="Tikhonov".

alpha.iter — a numeric scalar that, if supplied, is used for iterated Tikhonov rather than numerically solving for alpha, when using method="Tikhonov".

alpha.max — maximum of search range for α, the Tikhonov regularization parameter, when using method="Tikhonov".

alpha.min — minimum of search range for α, the Tikhonov regularization parameter, when using method="Tikhonov".

alpha.tol — the search tolerance for optimize when solving for α, the Tikhonov regularization parameter, when using method="Tikhonov".

bw — an object which, if provided, contains bandwidths and parameters (obtained from a previous invocation of npregiv) required to re-compute the estimator without having to re-run cross-validation and/or numerical optimization which is particularly costly in this setting (see details below for an illustration of its use)

constant — the constant to use when using method="Landweber-Fridman".

iterate.diff.tol — the search tolerance for the difference in the stopping rule from iteration to iteration when using method="Landweber-Fridman" (disable by setting to zero).

iterate.max — an integer indicating the maximum number of iterations permitted before termination occurs when using method="Landweber-Fridman".

iterate.Tikhonov — a logical value indicating whether to use iterated Tikhonov (one iteration) or not when using method="Tikhonov".

iterate.Tikhonov.num — an integer indicating the number of iterations to conduct when using method="Tikhonov".

method — the regularization method employed (defaults to "Landweber-Fridman", see Horowitz (2011); see Darolles, Fan, Florens and Renault (2011) for details for "Tikhonov").

nmulti — integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.

optim.abstol — the absolute convergence tolerance used by optim. Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to .Machine$double.eps.

optim.maxattempts — maximum number of attempts taken trying to achieve successful convergence in optim. Defaults to 100.

optim.maxit — maximum number of iterations used by optim. Defaults to 500.

optim.method — method used by optim for minimization of the objective function. See ?optim 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 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.reltol — relative convergence tolerance used by optim. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step. Defaults to sqrt(.Machine$double.eps), typically about 1e-8.

p — the order of the local polynomial regression (defaults to p=1, i.e. local linear).

penalize.iteration — a logical value indicating whether to penalize the norm by the number of iterations or not (default TRUE)

random.seed — an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.

return.weights.phi — a logical value (defaults to FALSE) indicating whether to return the weight matrix which when postmultiplied by the response y delivers the instrumental regression

return.weights.phi.deriv.1 — a logical value (defaults to FALSE) indicating whether to return the weight matrix which when postmultiplied by the response y delivers the first partial derivative of the instrumental regression with respect to z

return.weights.phi.deriv.2 — a logical value (defaults to FALSE) indicating whether to return the weight matrix which when postmultiplied by the response y delivers the second partial derivative of the instrumental regression with respect to z

smooth.residuals — a logical value indicating whether to optimize bandwidths for the regression of $(y - \varphi(z))$ on w (defaults to TRUE) or for the regression of $\varphi(z)$ on w during iteration

start.from — a character string indicating whether to start from $E(Y|z)$ (default, "Eyz") or from $E(E(Y|z)|z)$ (this can be overridden by providing starting.values below)

starting.values — a value indicating whether to commence Landweber-Fridman assuming $\varphi_{-1} = starting.values$ (proper Landweber-Fridman) or instead begin from $E(y|z)$ (defaults to NULL, see details below)

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)

...
: additional arguments supplied to npksum.

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 local rather than global optima are encountered the resulting estimates can be overly noisy. To best guard against this eventuality set nmulti to a larger number than the default nmulti=5 for the first iteration.

Note that for subsequent Landweber-Fridman iterations, a "warm start" strategy is employed. The optimal bandwidths from the previous iteration are used as starting values for the current iteration. The user-supplied nmulti is respected for all iterations. For iterations after the first successful one, these optimal bandwidths serve as the first of the multiple initial points (a warm start), while any remaining restarts are cold starts. If nmulti is not explicitly supplied by the user, it defaults to 5 for the first iteration and to 1 for all subsequent iterations. This strategy provides a balance between computational efficiency and robustness, allowing the numerical optimizer to refine the structural bandwidths as the residuals evolve incrementally while still guarding against local optima.

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.diff.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.

The option bw= would be useful, say, when bootstrapping is necessary. Note that when passing bw, it must be obtained from a previous invocation of npregiv. For instance, if model.iv was obtained from an invocation of npregiv with method="Landweber-Fridman", then the following needs to be fed to the subsequent invocation of npregiv:

```
model.iv <- npregiv(\dots)

bw <- NULL
bw$bw.E.y.w <- model.iv$bw.E.y.w
bw$bw.E.y.z <- model.iv$bw.E.y.z
bw$bw.resid.w <- model.iv$bw.resid.w
bw$bw.resid.fitted.w.z <- model.iv$bw.resid.fitted.w.z
bw$norm.index <- model.iv$norm.index

foo <- npregiv(\dots,bw=bw)
```

If, on the other hand model.iv was obtained from an invocation of npregiv with method="Tikhonov", then the following needs to be fed to the subsequent invocation of npregiv:

```
model.iv <- npregiv(\dots)

bw <- NULL
bw$alpha <- model.iv$alpha
bw$alpha.iter <- model.iv$alpha.iter
bw$bw.E.y.w <- model.iv$bw.E.y.w
bw$bw.E.E.y.w.z <- model.iv$bw.E.E.y.w.z
bw$bw.E.phi.w <- model.iv$bw.E.phi.w
bw$bw.E.E.phi.w.z <- model.iv$bw.E.E.phi.w.z

foo <- npregiv(\dots,bw=bw)
```

Or, if model.iv was obtained from an invocation of npregiv with either method="Landweber-Fridman" or method="Tikhonov", then the following would also work:

```
model.iv <- npregiv(\dots)

foo <- npregiv(\dots,bw=model.iv)
```

When exogenous predictors x (xeval) are passed, they are appended to both the endogenous predictors z and the instruments w as additional columns. If this is not desired, one can manually append the exogenous variables to z (or w) prior to passing z (or w), and then they will only appear among the z or w as desired.

Value

npregiv returns a npregiv object. The generic functions print, summary, and plot support objects of this type.

npregiv returns a list with components phi, phi.mat and either alpha when method="Tikhonov" or norm.index, norm.stop and convergence when method="Landweber-Fridman", among others.

In addition, if any of return.weights.* are invoked (*=1,2), then phi.weights and phi.deriv.*.weights return weight matrices for computing the instrumental regression and its partial derivatives. Note that these weights, post multiplied by the response vector y, will deliver the estimates returned in phi, phi.deriv.1, and phi.deriv.2 (the latter only being produced when p is 2 or greater). When invoked with evaluation data, similar matrices are returned but named phi.eval.weights and phi.deriv.eval.*.weights. These weights can be used for constrained estimation, among others.

When method="Landweber-Fridman" is invoked, bandwidth objects are returned in bw.E.y.w (scalar/vector), bw.E.y.z (scalar/vector), and bw.resid.w (matrix) and bw.resid.fitted.w.z, the latter matrices containing bandwidths for each iteration stored as rows. When method="Tikhonov" is invoked, bandwidth objects are returned in bw.E.y.w, bw.E.E.y.w.z, and bw.E.phi.w and bw.E.E.phi.w.z.

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.

Feve, 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 and S. Centorrino (2018), "Nonparametric instrumental derivatives," Journal of Nonparametric Statistics, 30 (2), 368-391.

Fridman, V. M. (1956), "A method of successive approximations for Fredholm integral equations of the first kind," Uspeskhi, 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 <- 500

## The DGP is as follows:

## 1) y = phi(z) + u

## 2) E(u|z) != 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 + eps)

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(-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("topright",
       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),
       bty="n")


## End(Not run)
```

---

npregivderiv *Nonparametric Instrumental Derivatives*

Description

npregivderiv uses the approach of Florens, Racine and Centorrino (2018) 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,
constant = 0.5,
iterate.break = TRUE,
iterate.max = 1000,
nmulti = NULL,
random.seed = 42,
smooth.residuals = TRUE,
start.from = c("Eyz","EEywz"),
starting.values = NULL,
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.
constant	the constant to use for Landweber-Fridman iteration.
iterate.break	a logical value indicating whether to compute all objects up to iterate.max or to break when a potential optimum arises (useful for inspecting full stopping rule profile up to iterate.max)
iterate.max	an integer indicating the maximum number of iterations permitted before termination occurs for Landweber-Fridman iteration.
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.

`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.

`start.from`
: a character string indicating whether to start from $E(Y|z)$ (default, `"Eyz"`) or from $E(E(Y|z)|z)$ (this can be overridden by providing `starting.values` below)

`starting.values`
: a value indicating whether to commence Landweber-Fridman assuming $\varphi'_{-1} = starting.values$ (proper Landweber-Fridman) or instead begin from $E(y|z)$ (defaults to `NULL`, see details below)

`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).

`...`
: additional arguments supplied to `npreg` and `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 local rather than global optima are encountered the resulting estimates can be overly noisy. To best guard against this eventuality set `nmulti` to a larger number than the default `nmulti=5` for the first iteration.

Note that for subsequent Landweber-Fridman iterations, a "warm start" strategy is employed. The optimal bandwidths from the previous iteration are used as starting values for the current iteration. The user-supplied `nmulti` is respected for all iterations. For iterations after the first successful one, these optimal bandwidths serve as the first of the multiple initial points (a warm start), while any remaining restarts are cold starts. If `nmulti` is not explicitly supplied by the user, it defaults to 5 for the first iteration and to 1 for all subsequent iterations. This strategy provides a balance between computational efficiency and robustness, allowing the numerical optimizer to refine the structural bandwidths as the residuals evolve incrementally while still guarding against local optima.

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.diff.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 `npregivderiv` object. The generic functions `print`, `summary`, and `plot` support objects of this type.

npregivderiv returns a list with components phi.prime, phi, num.iterations, norm.stop and convergence.

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.

Feve, 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 and S. Centorrino (2018), "Nonparametric instrumental derivatives," Journal of Nonparametric Statistics, 30 (2), 368-391.

Fridman, V. M. (1956), "A method of successive approximations for Fredholm integral equations of the first kind," Uspeskhi, 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) != 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 + eps)

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(-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)
```

---

npscoef *Smooth Coefficient Kernel Regression*

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, y.eval =
FALSE, ...)

## 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

bws	a bandwidth specification. This can be set as a scbandwidth object returned from an invocation of npscoefbw, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in tzdat. 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.
...	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 bwscaling, bwtype, ckertype, ckerorder, as described in npscoefbw.
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(bws), typically the environment from which npscoefbw was called.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
y.eval	If newdata contains dependent data and y.eval = TRUE, np will compute goodness of fit statistics on these data and return them. Defaults to FALSE.
txdat	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 zdat, will also correspond to Z from the model equation. 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.
tzdat	an 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.
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.
ezdat	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 txdat.
errors	a logical value indicating whether or not asymptotic standard errors should be computed and returned in the resulting smoothcoefficient object. Defaults to TRUE.

residuals	a logical value indicating that you want residuals computed and returned in the resulting smoothcoefficient object. Defaults to FALSE.
iterate	a logical value indicating whether or not backfitted estimates should be iterated for self-consistency. Defaults to TRUE.
maxiter	integer specifying the maximum number of times to iterate the backfitted estimates while attempting to make the backfitted estimates converge to the desired tolerance. Defaults to 100.
tol	desired tolerance on the relative convergence of backfit estimates. Defaults to .Machine$double.eps.
leave.one.out	a logical value to specify whether or not to compute the leave one out estimates. Will not work if e[xyz]dat is specified. Defaults to FALSE.
betas	a logical value indicating whether or not estimates of the components of γ should be returned in the smoothcoefficient 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:

eval	evaluation points
mean	estimation of the regression function (conditional mean) at the evaluation points
merr	if errors = TRUE, 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 (Doksum and Samarov (1995))
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 <tristen.hayfield@gmail.com>, 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.

Doksum, K. and A. Samarov (1995), "Nonparametric estimation of global functionals and a measure of the explanatory power of covariates in regression," The Annals of Statistics, 23 1443-1473.

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.

Li, Q. and D. Ouyang and J.S. Racine (2013), "Categorical semiparametric varying-coefficient models," Journal of Applied Econometrics, 28, 551-589.

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

```
## Not run:
# 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)

# 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 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 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.
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 zdat, 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 xdat.

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 xdat.

bws — a bandwidth specification. This can be set as a scbandwidth object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in xdat. 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 FALSE, a scbandwidth object will be returned with bandwidths set to those specified in bws. Defaults to TRUE.

bwmethod — which method was used to select bandwidths. cv.ls specifies least-squares cross-validation, which is all that is currently supported. Defaults to cv.ls.

bwscaling — a logical value that when set to TRUE the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is FALSE 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(standard deviation, mean absolute 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 FALSE.

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 liracine.

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,ncol(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 `optim` for minimization of the objective function. See `?optim` 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 `optim`. Defaults to `100`.

`optim.abstol` the absolute convergence tolerance used by `optim`. Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to `.Machine$double.eps`.

`optim.reltol` relative convergence tolerance used by `optim`. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step. Defaults to `sqrt(.Machine$double.eps)`, typically about `1e-8`.

`optim.maxit` maximum number of iterations used by `optim`. 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`.

`cv.num.iterations` integer specifying the number of times to iterate the backfitting process over all covariates. Defaults to `1`.

`backfit.iterate` 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`.

`backfit.maxiter` 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`.

`backfit.tol` tolerance to determine convergence of iterated evaluations of the smooth coefficient estimator. Defaults to `.Machine$double.eps`.

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 ~ 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 kth 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 ith 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 <tristen.hayfield@gmail.com>, 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.

Li, Q. and D. Ouyang and J.S. Racine (2013), "Categorical semiparametric varying-coefficient models," Journal of Applied Econometrics, 28, 551-589.

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

```
## Not run:
# 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)

# 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 `numeric` or `ts`.
`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 `integration` or `summation` (see below for details). Defaults to `integration`.
`bootstrap`	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`.
`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 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

Srho	the statistic vector Srho
Srho.cumulant	the cumulant statistic vector Srho.cumulant
Srho.bootstrap.mat	contains the bootstrap replications of Srho
Srho.cumulant.bootstrap.mat	contains the bootstrap replications of Srho.cumulant
P	the P-value vector of the Srho statistic vector
P.cumulant	the P-value vector of the cumulant Srho statistic vector
bootstrap	a logical value indicating whether bootstrapping was performed
boot.num	number of bootstrap replications
lag.num	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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
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")

## 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 <tristen.hayfield@gmail.com>, 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(s) in a nonparametric regression setting that is analogous to a simple t-test (F-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","pairwise"),
          boot.type = c("I","II"),
          pivot=TRUE,
          joint=FALSE,
          index = seq(1,ncol(xdat)),
          random.seed = 42,
          ...)
```

Arguments

bws	a bandwidth specification. This can be set as a rbandwidth object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in xdat. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths when using boot.type="II". 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.
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(bws), typically the environment from which npregbw was called.
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 xdat.
boot.method	a character string used to specify the bootstrap method for determining the null distribution. pairwise resamples pairwise, while the remaining methods use a residual bootstrap procedure. iid will generate independent identically distributed draws. wild will use a wild bootstrap. wild-rademacher will use a wild bootstrap with Rademacher variables. Defaults to iid.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
boot.type	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 boot.type="I".
pivot	a logical value which specifies whether to bootstrap a pivotal statistic or not (pivoting is achieved by dividing gradient estimates by their asymptotic standard errors). Defaults to TRUE.
joint	a logical value which specifies whether to conduct a joint test or individual test. This is to be used in conjunction with index where index contains two or more integers corresponding to the variables being tested, where the integers correspond to the variables in the order in which they appear among the set of explanatory variables in the function call to npreg/npregbw. Defaults to FALSE.
index	a vector of indices for the columns of xdat for which the test of significance is to be conducted. Defaults to (1,2,...,p) where p is the number of columns in xdat.
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.

Details

npsigtest implements a variety of methods for computing the null distribution of the test statistic and allows the user to investigate the impact of a variety of default settings including whether or not to pivot the statistic (pivot), whether pairwise or residual resampling is to be used (boot.method), and whether or not to recompute the bandwidths for the variables being tested (boot.type), among others.

Defaults are chosen so as to provide reasonable behaviour in a broad range of settings and this involves a trade-off between computational expense and finite-sample performance. However, the default boot.type="I", though computationally expedient, can deliver a test that can be slightly over-sized in small sample settings (e.g. at the 5% level the test might reject 8% of the time for samples of size $n = 100$ for some data generating processes). If the default setting (boot.type="I") delivers a P-value that is in the neighborhood (i.e. slightly smaller) of any classical level (e.g. 0.05) and you only have a modest amount of data, it might be prudent to re-run the test using the more computationally intensive boot.type="II" setting to confirm the original result. Note also that boot.method="pairwise" is not recommended for the multivariate local linear estimator due to substantial size distortions that may arise in certain cases.

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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# 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).
boot.method	a character string used to specify the bootstrap method. Can be set as iid or geom (see below for details). Defaults to iid.
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, and so on. This is used since we specify bw 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 bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype.

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, $D[f(y), f(\tilde{y})]$. 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

Srho	the statistic Srho
Srho.bootstrap	contains the bootstrap replications of Srho
P	the P-value of the statistic
boot.num	number of bootstrap replications
data.rotate	the rotated data series
bw	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 <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly nonlinear 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

```
## Not run:
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)

## 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)
```

---

nptgauss *Truncated Second-order Gaussian Kernels*

Description

nptgauss provides an interface for setting the truncation radius of the truncated second-order Gaussian kernel used by **np**.

Usage

```
nptgauss(b)
```

Arguments

b	Truncation radius of the kernel.

Details

nptgauss allows one to set the truncation radius of the truncated Gaussian kernel used by **np**, which defaults to 3. It automatically computes the constants describing the truncated gaussian kernel for the user.

We define the truncated gaussion kernel on the interval $[-b, b]$ as:

$$K = \frac{\alpha}{\sqrt{2\pi}} \left(e^{-z^2/2} - e^{-b^2/2} \right)$$

The constant α is computed as:

$$\alpha = \left[\int_{-b}^{b} \frac{1}{\sqrt{2\pi}} \left(e^{-z^2/2} - e^{-b^2/2} \right) \right]^{-1}$$

Given these definitions, the derivative kernel is simply:

$$K' = (-z) \frac{\alpha}{\sqrt{2\pi}} e^{-z^2/2}$$

The CDF kernel is:

$$G = \frac{\alpha}{2} \mathrm{erf}(z/\sqrt{2}) + \frac{1}{2} - c_0 z$$

The convolution kernel on $[-2b, 0]$ has the general form:

$$H_- = a_0 \, \mathrm{erf}(z/2 + b) e^{-z^2/4} + a_1 z + a_2 \, \mathrm{erf}((z + b)/\sqrt{2}) - c_0$$

and on $[0, 2b]$ it is:

$$H_+ = -a_0 \, \mathrm{erf}(z/2 - b) e^{-z^2/4} - a_1 z - a_2 \, \mathrm{erf}((z - b)/\sqrt{2}) - c_0$$

where a_0 is determined by the normalisation condition on H, a_2 is determined by considering the value of the kernel at $z = 0$ and a_1 is determined by the requirement that $H = 0$ at $[-2b, 2b]$.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

Examples

```
## The default kernel, a gaussian truncated at +- 3
nptgauss(b = 3.0)
```

npudens *Kernel Density 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).

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 npudensbw, 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 bws 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 bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype, as described in npudensbw.
tdat	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.
edat	a p-variate data frame of density evaluation points. By default, evaluation takes place on the data provided by tdat.

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(bws), typically the environment from which npudensbw 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 implements a variety of methods for estimating multivariate density functions (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. 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:

eval	the evaluation points.
dens	estimation of the density at the evaluation points
derr	standard errors of the density estimates
log_likelihood	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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# 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)

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)

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 = 1.490116e-07,
          tol = 1.490116e-04,
          small = 1.490116e-05,
          lbc.dir = 0.5,
          dfc.dir = 3,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          initc.dir = 1.0,
          lbd.dir = 0.1,
          hbd.dir = 1,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          initd.dir = 1.0,
          lbc.init = 0.1,
          hbc.init = 2.0,
          cfac.init = 0.5,
          lbd.init = 0.1,
          hbd.init = 0.9,
          dfac.init = 0.375,
          scale.init.categorical.sample = FALSE,
          transform.bounds = FALSE,
          invalid.penalty = c("baseline","dbmax"),
          penalty.multiplier = 10,
          ...)

## Default S3 method:
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
```

```
            ftol,
            tol,
            small,
            lbc.dir,
            dfc.dir,
            cfac.dir,
            initc.dir,
            lbd.dir,
            hbd.dir,
            dfac.dir,
            initd.dir,
            lbc.init,
            hbc.init,
            cfac.init,
            lbd.init,
            hbd.init,
            dfac.init,
            scale.init.categorical.sample,
            transform.bounds,
            invalid.penalty,
            penalty.multiplier,
            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 environment(formula), 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 dat. 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. cv.ml specifies likelihood cross-validation, cv.ls specifies least-squares cross-validation, and normal-reference 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 jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, 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 cv.ml.

bwscaling — a logical value that when set to TRUE the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is FALSE 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 jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, 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 FALSE.

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 fixed. Option summary:
fixed: compute fixed bandwidths
generalized_nn: compute generalized nearest neighbors
adaptive_nn: compute adaptive nearest neighbors

bandwidth.compute — 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 bws. Defaults to TRUE.

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.

okertype character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin or liracine.

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 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 fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

tol tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

small a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

lbc.dir, dfc.dir, cfac.dir, initc.dir
lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbd.dir, hbd.dir, dfac.dir, initd.dir
lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbc.init, hbc.init, cfac.init
lower bound, upper bound, and non-random initial values for scale factors for numeric variables for Powell's algorithm. See Details

lbd.init, hbd.init, dfac.init
lower bound, upper bound, and non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

scale.init.categorical.sample
a logical value that when set to TRUE scales lbd.dir, hbd.dir, dfac.dir, and initd.dir by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of numeric variables. See Details

transform.bounds
a logical value that when set to TRUE applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to FALSE.

invalid.penalty
a character string specifying the penalty used when the optimizer encounters invalid bandwidths. "baseline" returns a finite penalty based on a baseline objective; "dbmax" returns DBL_MAX. Defaults to "baseline".

penalty.multiplier
a numeric multiplier applied to the baseline penalty when invalid.penalty="baseline". Defaults to 10.

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 ~ data, where data is a series of variables specified by name, separated by the separation character '+'. For example, ~ 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.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures

of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying nmulti). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

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 kth 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 ith 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 <tristen.hayfield@gmail.com>, 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

```
## Not run:
# 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)

# 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)
```

---

npudist *Kernel Distribution Estimation with Mixed Data Types*

Description

npudist computes kernel unconditional cumulative distribution estimates on evaluation data, given a set of training data and a bandwidth specification (a dbandwidth object or a bandwidth vector, bandwidth type, and kernel type) using the method of Li, Li and Racine (2017).

Usage

```
npudist(bws, ...)

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

## S3 method for class 'dbandwidth'
npudist(bws,
        tdat = stop("invoked without training data 'tdat'"),
        edat,
        ...)

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

```
## Default S3 method:
npudist(bws, tdat, ...)
```

Arguments

bws	a dbandwidth specification. This can be set as a dbandwidth object returned from an invocation of npudistbw, 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 bws as a p-vector and not a dbandwidth object, and you do not desire the default behaviours. To do this, you may specify any of bwscaling, bwtype, ckertype, ckerorder, okertype, as described in npudistbw.
tdat	a p-variate data frame of sample realizations (training data) used to estimate the cumulative distribution. Defaults to the training data used to compute the bandwidth object.
edat	a p-variate data frame of cumulative distribution evaluation points. By default, evaluation takes place on the data provided by tdat.
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(bws), typically the environment from which npudistbw 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 npudistbw and then
compute the cumulative distribution:

bw <- npudistbw(~y)
Fhat <- npudist(bw)

Usage 2: alternatively, compute the bandwidth object indirectly:

Fhat <- npudist(~y)

Usage 3: modify the default kernel and order:

Fhat <- npudist(~y, ckertype="epanechnikov", ckerorder=4)

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

```
interface:

Fhat <- npudist(tdat = y, ckertype="epanechnikov", ckerorder=4)
```

npudist implements a variety of methods for estimating multivariate cumulative distributions (p-variate) defined over a set of possibly continuous and/or discrete (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 cumulative distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative distribution 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) 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. Ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

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:

eval	the evaluation points.
dist	estimate of the cumulative distribution at the evaluation points
derr	standard errors of the cumulative distribution 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 <tristen.hayfield@gmail.com>, 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.

Li, C. and H. Li and J.S. Racine (2017), "Cross-Validated Mixed Datatype Bandwidth Selection for Nonparametric Cumulative Distribution/Survivor Functions," Econometric Reviews, **36**, 970-987.

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

npudistbw , density

Examples

```
## Not run:
# 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 cross-validation, then create a grid of data
# on which the cumulative distribution will be evaluated for plotting
# purposes.

data("Italy")
attach(Italy)

# Compute bandwidths using cross-validation (default).

bw <- npudistbw(formula=~ordered(year)+gdp)

# At this stage you could use npudist() to do a variety of things. Here
# we compute the npudist() object and place it in Fhat.

Fhat <- npudist(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 cumulative distribution computed for the
# evaluation data

Fhat <- fitted(npudist(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 CDF 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 = "Cumulative Distribution 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 cross-validation, then create a grid of data
# on which the cumulative distribution will be evaluated for plotting
# purposes.
```

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

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

# Compute bandwidths using cross-validation (default).

bw <- npudistbw(dat=data)

# At this stage you could use npudist() to do a variety of
# things. Here we compute the npudist() object and place it in Fhat.

Fhat <- npudist(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 cumulative distribution computed for the
# evaluation data

Fhat <- fitted(npudist(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 CDF F, and a 2D
# contour plot.

persp(as.integer(levels(year.seq)), gdp.seq, F, col="lightblue",
      ticktype="detailed", ylab="GDP", xlab="Year",
      zlab="Cumulative Distribution",
      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 = "Cumulative Distribution 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 cumulative distribution function.

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

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

bw <- npudistbw(formula=~eruptions+waiting)

summary(bw)

# Plot the cumulative distribution 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)

detach(faithful)

# EXAMPLE 2 (INTERFACE=DATA FRAME): For this example, we load the old
# faithful geyser data and compute the cumulative distribution function.

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

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

bw <- npudistbw(dat=faithful)

summary(bw)
```

```
# Plot the cumulative distribution 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)

detach(faithful)

## End(Not run)
```

---

npudistbw *Kernel Distribution Bandwidth Selection with Mixed Data Types*

Description

npudistbw computes a bandwidth object for a p-variate kernel cumulative distribution estimator defined over mixed continuous and discrete (ordered) data using either the normal reference rule-of-thumb or least-squares cross validation using the method of Li, Li and Racine (2017).

Usage

```
npudistbw(...)

## S3 method for class 'formula'
npudistbw(formula, data, subset, na.action, call, gdata = NULL,...)

## S3 method for class 'NULL'
npudistbw(dat = stop("invoked without input data 'dat'"),
          bws,
          ...)

## S3 method for class 'dbandwidth'
npudistbw(dat = stop("invoked without input data 'dat'"),
          bws,
          gdat = NULL,
          bandwidth.compute = TRUE,
          nmulti,
          remin = TRUE,
          itmax = 10000,
          do.full.integral = FALSE,
          ngrid = 100,
          ftol = 1.490116e-07,
          tol = 1.490116e-04,
          small = 1.490116e-05,
          lbc.dir = 0.5,
          dfc.dir = 3,
          cfac.dir = 2.5*(3.0-sqrt(5)),
```

```
            initc.dir = 1.0,
            lbd.dir = 0.1,
            hbd.dir = 1,
            dfac.dir = 0.25*(3.0-sqrt(5)),
            initd.dir = 1.0,
            lbc.init = 0.1,
            hbc.init = 2.0,
            cfac.init = 0.5,
            lbd.init = 0.1,
            hbd.init = 0.9,
            dfac.init = 0.375,
            scale.init.categorical.sample = FALSE,
            memfac = 500.0,
            transform.bounds = FALSE,
            invalid.penalty = c("baseline","dbmax"),
            penalty.multiplier = 10,
            ...)

## Default S3 method:
npudistbw(dat = stop("invoked without input data 'dat'"),
            bws,
            gdat,
            bandwidth.compute = TRUE,
            nmulti,
            remin,
            itmax,
            do.full.integral,
            ngrid,
            ftol,
            tol,
            small,
            lbc.dir,
            dfc.dir,
            cfac.dir,
            initc.dir,
            lbd.dir,
            hbd.dir,
            dfac.dir,
            initd.dir,
            lbc.init,
            hbc.init,
            cfac.init,
            lbd.init,
            hbd.init,
            dfac.init,
            scale.init.categorical.sample,
            memfac,
            transform.bounds,
```

```
            invalid.penalty,
            penalty.multiplier,
            bwmethod,
            bwscaling,
            bwtype,
            ckertype,
            ckerorder,
            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 environment(formula), 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.
gdata	a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles).
dat	a p-variate data frame on which bandwidth selection will be performed. The data types may be continuous, discrete (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 dat. 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.
gdat	a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles).
...	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. cv.cdf specifies least-squares cross-validation for cumulative distribution functions (Li, Li and Racine (2017)), and normal-reference just computes the 'rule-of-thumb'

bandwidth h_j using the standard formula $h_j = 1.587\sigma_j n^{-1/(P+l)}$, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, 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 `cv.cdf`.

`bwscaling` a logical value that when set to `TRUE` the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is `FALSE` 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 $h_j = c_j\sigma_j n^{-1/(P+l)}$, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, 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 $h_j = c_j n^{-2/(P+l)}$, where here j denotes discrete variable j. Defaults to `FALSE`.

`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 `fixed`. Option summary:
`fixed`: compute fixed bandwidths
`generalized_nn`: compute generalized nearest neighbors
`adaptive_nn`: compute adaptive nearest neighbors

`bandwidth.compute`
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 `bws`. Defaults to `TRUE`.

`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`.

`okertype` character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin`.

`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 `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`.

`do.full.integral`
a logical value which when set as `TRUE` evaluates the moment-based integral on the entire sample. Defaults to `FALSE`.

`ngrid` integer number of grid points to use when computing the moment-based integral. Defaults to `100`.

ftol fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

tol tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

small a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

lbc.dir, dfc.dir, cfac.dir, initc.dir
lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbd.dir, hbd.dir, dfac.dir, initd.dir
lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbc.init, hbc.init, cfac.init
lower bound, upper bound, and non-random initial values for scale factors for numeric variables for Powell's algorithm. See Details

lbd.init, hbd.init, dfac.init
lower bound, upper bound, and non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

scale.init.categorical.sample
a logical value that when set to TRUE scales lbd.dir, hbd.dir, dfac.dir, and initd.dir by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of numeric variables. See Details

transform.bounds
a logical value that when set to TRUE applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to FALSE.

invalid.penalty
a character string specifying the penalty used when the optimizer encounters invalid bandwidths. "baseline" returns a finite penalty based on a baseline objective; "dbmax" returns DBL_MAX. Defaults to "baseline".

penalty.multiplier
a numeric multiplier applied to the baseline penalty when invalid.penalty="baseline". Defaults to 10.

memfac The algorithm to compute the least-squares objective function uses a block-based algorithm to eliminate or minimize redundant kernel evaluations. Due to memory, hardware and software constraints, a maximum block size must be imposed by the algorithm. This block size is roughly equal to memfac*10^5 elements. Empirical tests on modern hardware find that a memfac of 500 performs well. If you experience out of memory errors, or strange behaviour for large data sets (>100k elements) setting memfac to a lower value may fix the problem.

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 <- npudistbw(~y)

Usage 2: compute a bandwidth object using the data frame interface
and change the default kernel and order:

Fhat <- npudistbw(tdat = y, ckertype="epanechnikov", ckerorder=4)
```

npudistbw implements a variety of methods for choosing bandwidths for multivariate (p-variate) distributions defined over a set of possibly continuous and/or discrete (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 cumulative distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative distribution is estimated, x. Fixed bandwidths are constant over the support of x.

npudistbw 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) 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 ~ data, where data is a series of variables specified by name, separated by the separation character '+'. For example, ~ 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. Ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters

themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying `nmulti`). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user them self.

Value

`npudistbw` 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 kth 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 ith 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 <tristen.hayfield@gmail.com>, 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.

Bowman, A. and P. Hall and T. Prvan (1998), "Bandwidth selection for the smoothing of distribution functions," Biometrika, 85, 799-808.

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.

Li, C. and H. Li and J.S. Racine (2017), "Cross-Validated Mixed Datatype Bandwidth Selection for Nonparametric Cumulative Distribution/Survivor Functions," Econometric Reviews, **36**, 970-987.

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 Cumulative Distribution 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, npudist, npudist

Examples

```
## Not run:
# 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 cumulative distribution 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 npudist()
# function.

bw <- npudistbw(formula=~ordered(year)+gdp, 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 <- npudistbw(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 1.587 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 1.587 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 <- npudistbw(formula=~ordered(year)+gdp, bws=c(0, 1.587),
                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 <- npudistbw(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 <- npudistbw(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 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 <- npudistbw(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 <- npudistbw(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 cumulative distribution 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 npudist()
# function.

bw <- npudistbw(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 <- npudistbw(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 1.587 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 1.587 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 <- npudistbw(dat=data, bws=c(0, 1.587),
                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 <- npudistbw(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 <- npudistbw(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 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 <- npudistbw(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 <- npudistbw(dat=data, bws=c(1, 1))

summary(bw)

detach(Italy)

## End(Not run)
```

---

npuniden.boundary *Kernel Bounded Univariate Density Estimation Via Boundary Kernel Functions*

Description

npuniden.boundary computes kernel univariate unconditional density estimates given a vector of continuously distributed training data and, optionally, a bandwidth (otherwise least squares cross-validation is used for its selection). Lower and upper bounds [a,b] can be supplied (default is the empirical support $[\min(X), \max(X)]$) and if a is set to -Inf there is only one bound on the right, while if b is set to Inf there is only one bound on the left. If a is set to -Inf and b to Inf and the Gaussian type 1 kernel function is used, this will deliver the standard unadjusted kernel density estimate.

Usage

```
npuniden.boundary(X = NULL,
                  Y = NULL,
```

```
                    h = NULL,
                    a = min(X),
                    b = max(X),
                    bwmethod = c("cv.ls","cv.ml"),
                    cv = c("grid-hybrid","numeric"),
                    grid = NULL,
                    kertype = c("gaussian1","gaussian2",
                                "beta1","beta2",
                                "fb","fbl","fbu",
                                "rigaussian","gamma"),
                    nmulti = 5,
                    proper = FALSE)
```

Arguments

X	a required numeric vector of training data lying in $[a, b]$
Y	an optional numeric vector of evaluation data lying in $[a, b]$
h	an optional bandwidth (>0)
a	an optional lower bound (defaults to lower bound of empirical support $\min(X)$)
b	an optional upper bound (defaults to upper bound of empirical support $\max(X)$)
bwmethod	whether to conduct bandwidth search via least squares cross-validation ("cv.ls") or likelihood cross-validation ("cv.ml")
cv	an optional argument for search (default is likely more reliable in the presence of local maxima)
grid	an optional grid used for the initial grid search when cv="grid-hybrid"
kertype	an optional kernel specification (defaults to "gaussian1")
nmulti	number of multi-starts used when cv="numeric" (defaults to 5)
proper	an optional logical value indicating whether to enforce proper density and distribution function estimates over the range $[a, b]$

Details

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

```
model <- npuniden.boundary(X,a=-2,b=3)
```

npuniden.boundary implements a variety of methods for estimating a univariate density function defined over a continuous random variable in the presence of bounds via the use of so-called boundary or edge kernel functions.

The kernel functions "beta1" and "beta2" are Chen's (1999) type 1 and 2 kernel functions with biases of $O(h)$, the "gamma" kernel function is from Chen (2000) with a bias of $O(h)$, "rigaussian" is the reciprocal inverse Gaussian kernel function (Scaillet (2004), Igarashi & Kakizawa (2014)) with bias of $O(h)$, and "gaussian1" and "gaussian2" are truncated Gaussian kernel functions with biases of $O(h)$ and $O(h^2)$, respectively. The kernel functions "fb", "fbl" and "fbu" are

floating boundary polynomial biweight kernels with biases of $O(h^2)$ (Scott (1992), Page 146). Without exception, these kernel functions are asymmetric in general with shape that changes depending on where the density is being estimated (i.e., how close the estimation point x in $\hat{f}(x)$ is to a boundary). This function is written purely in R, so to see the exact form for each of these kernel functions, simply enter the name of this function in R (i.e., enter `npuniden.boundary` after loading this package) and scroll up for their definitions.

The kernel functions `"gamma"`, `"rigaussian"`, and `"fbl"` have support $[a, \infty]$. The kernel function `"fbu"` has support $[-\infty, b]$. The rest have support on $[a, b]$. Note that the two sided support default values are `a=min(X)` and `b=max(X)`.

Note that data-driven bandwidth selection is more nuanced in bounded settings, therefore it would be prudent to manually select a bandwidth that is, say, 1/25th of the range of the data and manually inspect the estimate (say `h=0.05` when $X \in [0, 1]$). Also, it may be wise to compare the density estimate with that from a histogram with the option `breaks=25`. Note also that the kernel functions `"gaussian2"`, `"fb"`, `"fbl"` and `"fbu"` can assume negative values leading to potentially negative density estimates, and must be trimmed when conducting likelihood cross-validation which can lead to oversmoothing. Least squares cross-validation is unaffected and appears to be more reliable in such instances hence is the default here.

Scott (1992, Page 149) writes "While boundary kernels can be very useful, there are potentially serious problems with real data. There are an infinite number of boundary kernels reflecting the spectrum of possible design constraints, and these kernels are not interchangeable. Severe artifacts can be introduced by any one of them in inappropriate situations. Very careful examination is required to avoid being victimized by the particular boundary kernel chosen. Artifacts can unfortunately be introduced by the choice of the support interval for the boundary kernel."

Note that since some kernel functions can assume negative values, this can lead to improper density estimates. The estimated distribution function is obtained via numerical integration of the estimated density function and may itself not be proper even when evaluated on the full range of the data $[a, b]$. Setting the option `proper=TRUE` will render the density and distribution estimates proper over the full range of the data, though this may not in general be a mean square error optimal strategy.

Finally, note that this function is pretty bare-bones relative to other functions in this package. For one, at this time there is no automatic print support so kindly see the examples for illustrations of its use, among other differences.

Value

`npuniden.boundary` returns the following components:

`f`	estimated density at the points X
`F`	estimated distribution at the points X (numeric integral of f)
`sd.f`	asymptotic standard error of the estimated density at the points X
`sd.F`	asymptotic standard error of the estimated distribution at the points X
`h`	bandwidth used
`nmulti`	number of multi-starts used

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Bouezmarni, T. and Rolin, J.-M. (2003). "Consistency of the beta kernel density function estimator," The Canadian Journal of Statistics / La Revue Canadienne de Statistique, 31(1):89-98.

Chen, S. X. (1999). "Beta kernel estimators for density functions," Computational Statistics & Data Analysis, 31(2):131-145.

Chen, S. X. (2000). "Probability density function estimation using gamma kernels," Annals of the Institute of Statistical Mathematics, 52(3):471-480.

Diggle, P. (1985). "A kernel method for smoothing point process data," Journal of the Royal Statistical Society. Series C (Applied Statistics), 34(2):138-147.

Igarashi, G. and Y. Kakizawa (2014). "Re-formulation of inverse Gaussian, reciprocal inverse Gaussian, and Birnbaum-Saunders kernel estimators," Statistics & Probability Letters, 84:235-246.

Igarashi, G. and Y. Kakizawa (2015). "Bias corrections for some asymmetric kernel estimators," Journal of Statistical Planning and Inference, 159:37-63.

Igarashi, G. (2016). "Bias reductions for beta kernel estimation," Journal of Nonparametric Statistics, 28(1):1-30.

Racine, J. S. and Q. Li and Q. Wang, "Boundary-adaptive kernel density estimation: the case of (near) uniform density", Journal of Nonparametric Statistics, 2024, 36 (1), 146-164, https://doi.org/10.1080/10485252.2023.2

Scaillet, O. (2004). "Density estimation using inverse and reciprocal inverse Gaussian kernels," Journal of Nonparametric Statistics, 16(1-2):217-226.

Scott, D. W. (1992). "Multivariate density estimation: Theory, practice, and visualization," New York: Wiley.

Zhang, S. and R. J. Karunamuni (2010). "Boundary performance of the beta kernel estimators," Journal of Nonparametric Statistics, 22(1):81-104.

See Also

The **Ake**, **bde**, and **Conake** packages and the function `npuniden.reflect`.

Examples

```
## Not run:
## Example 1: f(0)=0, f(1)=1, plot boundary corrected density,
## unadjusted density, and DGP
set.seed(42)
n <- 100
X <- sort(rbeta(n,5,1))
dgp <- dbeta(X,5,1)
model.g1 <- npuniden.boundary(X,kertype="gaussian1")
model.g2 <- npuniden.boundary(X,kertype="gaussian2")
model.b1 <- npuniden.boundary(X,kertype="beta1")
model.b2 <- npuniden.boundary(X,kertype="beta2")
model.fb <- npuniden.boundary(X,kertype="fb")
model.unadjusted <- npuniden.boundary(X,a=-Inf,b=Inf)
ylim <- c(0,max(c(dgp,model.g1$f,model.g2$f,model.b1$f,model.b2$f,model.fb$f)))
plot(X,dgp,ylab="Density",ylim=ylim,type="l")
lines(X,model.g1$f,lty=2,col=2)
lines(X,model.g2$f,lty=3,col=3)
```

```
lines(X,model.b1$f,lty=4,col=4)
lines(X,model.b2$f,lty=5,col=5)
lines(X,model.fb$f,lty=6,col=6)
lines(X,model.unadjusted$f,lty=7,col=7)
rug(X)
legend("topleft",c("DGP",
                   "Boundary Kernel (gaussian1)",
                   "Boundary Kernel (gaussian2)",
                   "Boundary Kernel (beta1)",
                   "Boundary Kernel (beta2)",
                   "Boundary Kernel (floating boundary)",
                   "Unadjusted"),col=1:7,lty=1:7,bty="n")

## Example 2: f(0)=0, f(1)=0, plot density, distribution, DGP, and
## asymptotic point-wise confidence intervals
set.seed(42)
X <- sort(rbeta(100,5,3))
model <- npuniden.boundary(X)
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
ylim=range(c(model$f,model$f+1.96*model$sd.f,model$f-1.96*model$sd.f,dbeta(X,5,3)))
plot(X,model$f,ylim=ylim,ylab="Density",type="l",)
lines(X,model$f+1.96*model$sd.f,lty=2)
lines(X,model$f-1.96*model$sd.f,lty=2)
lines(X,dbeta(X,5,3),col=2)
rug(X)
legend("topleft",c("Density","DGP"),lty=c(1,1),col=1:2,bty="n")

plot(X,model$F,ylab="Distribution",type="l")
lines(X,model$F+1.96*model$sd.F,lty=2)
lines(X,model$F-1.96*model$sd.F,lty=2)
lines(X,pbeta(X,5,3),col=2)
rug(X)
legend("topleft",c("Distribution","DGP"),lty=c(1,1),col=1:2,bty="n")

## Example 3: Age for working age males in the cps71 data set bounded
## below by 21 and above by 65
data(cps71)
attach(cps71)
model <- npuniden.boundary(age,a=21,b=65)
par(mfrow=c(1,1))
hist(age,prob=TRUE,main="")
lines(age,model$f)
lines(density(age,bw=model$h),col=2)
legend("topright",c("Boundary Kernel","Unadjusted"),lty=c(1,1),col=1:2,bty="n")
detach(cps71)
par(oldpar)

## End(Not run)
```

npuniden.reflect *Kernel Bounded Univariate Density Estimation Via Data-Reflection*

Description

npuniden.reflect computes kernel univariate unconditional density estimates given a vector of continuously distributed training data and, optionally, a bandwidth (otherwise likelihood cross-validation is used for its selection). Lower and upper bounds [a,b] can be supplied (default is [0,1]) and if a is set to -Inf there is only one bound on the right, while if b is set to Inf there is only one bound on the left.

Usage

```
npuniden.reflect(X = NULL,
                 Y = NULL,
                 h = NULL,
                 a = 0,
                 b = 1,
                 ...)
```

Arguments

X	a required numeric vector of training data lying in $[a, b]$
Y	an optional numeric vector of evaluation data lying in $[a, b]$
h	an optional bandwidth (>0)
a	an optional lower bound (defaults to 0)
b	an optional upper bound (defaults to 1)
...	optional arguments passed to npudensbw and npudens

Details

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

```
model <- npuniden.reflect(X,a=-2,b=3)
```

npuniden.reflect implements the data-reflection method for estimating a univariate density function defined over a continuous random variable in the presence of bounds.

Note that data-reflection imposes a zero derivative at the boundary, i.e., $f'(a) = f'(b) = 0$.

Value

npuniden.reflect returns the following components:

f	estimated density at the points X
F	estimated distribution at the points X (numeric integral of f)

sd.f	asymptotic standard error of the estimated density at the points X
sd.F	asymptotic standard error of the estimated distribution at the points X
h	bandwidth used
nmulti	number of multi-starts used

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Boneva, L. I., Kendall, D., and Stefanov, I. (1971). "Spline transformations: Three new diagnostic aids for the statistical data- analyst," Journal of the Royal Statistical Society. Series B (Methodological), 33(1):1-71.

Cline, D. B. H. and Hart, J. D. (1991). "Kernel estimation of densities with discontinuities or discontinuous derivatives," Statistics, 22(1):69-84.

Hall, P. and Wehrly, T. E. (1991). "A geometrical method for removing edge effects from kernel-type nonparametric regression estimators," Journal of the American Statistical Association, 86(415):665-672.

See Also

The **Ake**, **bde**, and **Conake** packages and the function npuniden.boundary.

Examples

```
## Not run:
## Example 1: f(0)=0, f(1)=1, plot boundary corrected density,
## unadjusted density, and DGP
set.seed(42)
n <- 100
X <- sort(rbeta(n,5,1))
dgp <- dbeta(X,5,1)
model <- npuniden.reflect(X)
model.unadjusted <- npuniden.boundary(X,a=-Inf,b=Inf)
ylim <- c(0,max(c(dgp,model$f,model.unadjusted$f)))
plot(X,model$f,ylab="Density",ylim=ylim,type="l")
lines(X,model.unadjusted$f,lty=2,col=2)
lines(X,dgp,lty=3,col=3)
rug(X)
legend("topleft",c("Data-Reflection","Unadjusted","DGP"),col=1:3,lty=1:3,bty="n")

## Example 2: f(0)=0, f(1)=0, plot density, distribution, DGP, and
## asymptotic point-wise confidence intervals
set.seed(42)
X <- sort(rbeta(100,5,3))
model <- npuniden.reflect(X)
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
ylim=range(c(model$f,model$f+1.96*model$sd.f,model$f-1.96*model$sd.f,dbeta(X,5,3)))
```

```
plot(X,model$f,ylim=ylim,ylab="Density",type="l",)
lines(X,model$f+1.96*model$sd.f,lty=2)
lines(X,model$f-1.96*model$sd.f,lty=2)
lines(X,dbeta(X,5,3),col=2)
rug(X)
legend("topleft",c("Density","DGP"),lty=c(1,1),col=1:2,bty="n")

plot(X,model$F,ylab="Distribution",type="l")
lines(X,model$F+1.96*model$sd.F,lty=2)
lines(X,model$F-1.96*model$sd.F,lty=2)
lines(X,pbeta(X,5,3),col=2)
rug(X)
legend("topleft",c("Distribution","DGP"),lty=c(1,1),col=1:2,bty="n")


## Example 3: Age for working age males in the cps71 data set bounded
## below by 21 and above by 65
data(cps71)
attach(cps71)
model <- npuniden.reflect(age,a=21,b=65)
par(mfrow=c(1,1))
hist(age,prob=TRUE,main="",ylim=c(0,max(model$f)))
lines(age,model$f)
lines(density(age,bw=model$h),col=2)
legend("topright",c("Data-Reflection","Unadjusted"),lty=c(1,1),col=1:2,bty="n")
detach(cps71)
par(oldpar)

## End(Not run)
```

---

npuniden.sc *Kernel Shape Constrained Bounded Univariate Density Estimation*

Description

npuniden.sc computes shape constrained kernel univariate unconditional density estimates given a vector of continuously distributed training data and a bandwidth. Lower and upper bounds [a,b] can be supplied (default is [0,1]) and if a is set to -Inf there is only one bound on the right, while if b is set to Inf there is only one bound on the left.

Usage

```
npuniden.sc(X = NULL,
            Y = NULL,
            h = NULL,
            a = 0,
            b = 1,
            lb = NULL,
            ub = NULL,
```

```
              extend.range = 0,
              num.grid = 0,
              function.distance = TRUE,
              integral.equal = FALSE,
              constraint = c("density",
                             "mono.incr",
                             "mono.decr",
                             "concave",
                             "convex",
                             "log-concave",
                             "log-convex"))
```

Arguments

X	a required numeric vector of training data lying in $[a, b]$
Y	an optional numeric vector of evaluation data lying in $[a, b]$
h	a bandwidth (> 0)
a	an optional lower bound on the support of X or Y (defaults to 0)
b	an optional upper bound on the support of X or Y (defaults to 1)
lb	a scalar lower bound (≥ 0) to be used in conjunction with constraint="density"
ub	a scalar upper bound (≥ 0 and $\geq$ lb) to be used in conjunction with constraint="density"
extend.range	number specifying the fraction by which the range of the training data should be extended for the additional grid points (passed to the function extendrange)
num.grid	number of additional grid points (in addition to X and Y) placed on an equispaced grid spanning extendrange(c(X,Y),f=extend.range) (if num.grid=0 no additional grid points will be used regardless of the value of extend.range)
function.distance	a logical value that, if TRUE, minimizes the squared deviation between the constrained and unconstrained estimates, otherwise, minimizes the squared deviation between the constrained and unconstrained weights
integral.equal	a logical value, that, if TRUE, adjusts the constrained estimate to have the same probability mass over the range X, Y, and the additional grid points
constraint	a character string indicating whether the estimate is to be constrained to be monotonically increasing (constraint="mono.incr"), decreasing (constraint="mono.decr"), convex (constraint="convex"), concave (constraint="concave"), log-convex (constraint="log-convex"), or log-concave (constraint="log-concave")

Details

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

```
model <- npuniden.sc(X,a=-2,b=3)
```

npuniden.sc implements a methods for estimating a univariate density function defined over a continuous random variable in the presence of bounds subject to a variety of shape constraints. The bounded estimates use the truncated Gaussian kernel function.

Note that for the log-constrained estimates, the derivative estimate returned is that for the log-constrained estimate not the non-log value of the estimate returned by the function. See Example 5 below hat manually plots the log-density and returned derivative (no transformation is needed when plotting the density estimate itself).

If the quadratic program solver fails to find a solution, the unconstrained estimate is returned with an immediate warning. Possible causes to be investigated are undersmoothing, sparsity, and the presence of non-sample grid points. To investigate the possibility of undersmoothing try using a larger bandwidth, to investigate sparsity try decreasing extend.range, and to investigate non-sample grid points try setting num.grid to 0.

Mean square error performance seems to improve generally when using additional grid points in the empirical support of X and Y (i.e., in the observed range of the data sample) but appears to deteriorate when imposing constraints beyond the empirical support (i.e., when extend.range is positive). Increasing the number of additional points beyond a hundred or so appears to have a limited impact.

The option function.distance=TRUE appears to perform better for imposing convexity, concavity, log-convexity and log-concavity, while function.distance=FALSE appears to perform better for imposing monotonicity, whether increasing or decreasing (based on simulations for the Beta(s1,s2) distribution with sample size $n = 100$).

Value

A list with the following elements:

f	unconstrained density estimate
f.sc	shape constrained density estimate
se.f	asymptotic standard error of the unconstrained density estimate
se.f.sc	asymptotic standard error of the shape constrained density estimate
f.deriv	unconstrained derivative estimate (of order 1 or 2 or log thereof)
f.sc.deriv	shape constrained derivative estimate (of order 1 or 2 or log thereof)
F	unconstrained distribution estimate
F.sc	shape constrained distribution estimate
integral.f	the integral of the unconstrained estimate over X, Y, and the additional grid points
integral.f.sc	the integral of the constrained estimate over X, Y, and the additional grid points
solve.QP	logical, if TRUE solve.QP succeeded, otherwise failed
attempts	number of attempts when solve.QP fails (max = 9)

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Du, P. and C. Parmeter and J. Racine (2024), "Shape Constrained Kernel PDF and PMF Estimation", Statistica Sinica, 34 (1), 257-289, doi:10.5705/ss.202021.0112

See Also

The **logcondens**, **LogConDEAD**, and **scdensity** packages, and the function `npuniden.boundary`.

Examples

```
## Not run:
n <- 100
set.seed(42)

## Example 1: N(0,1), constrain the density to lie within lb=.1 and ub=.2

X <- sort(rnorm(n))
h <- npuniden.boundary(X,a=-Inf,b=Inf)$h
foo <- npuniden.sc(X,h=h,constraint="density",a=-Inf,b=Inf,lb=.1,ub=.2)
ylim <- range(c(foo$f.sc,foo$f))
plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 2: Beta(5,1), DGP is monotone increasing, impose valid
## restriction

X <- sort(rbeta(n,5,1))
h <- npuniden.boundary(X)$h

foo <- npuniden.sc(X=X,h=h,constraint=c("mono.incr"))

oldpar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="First Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 3: Beta(1,5), DGP is monotone decreasing, impose valid
## restriction

X <- sort(rbeta(n,1,5))
```

```
h <- npuniden.boundary(X)$h

foo <- npuniden.sc(X=X,h=h,constraint=c("mono.decr"))

par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="First Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 4: N(0,1), DGP is log-concave, impose invalid concavity
## restriction

X <- sort(rnorm(n))
h <- npuniden.boundary(X,a=-Inf,b=Inf)$h

foo <- npuniden.sc(X=X,h=h,a=-Inf,b=Inf,constraint=c("concave"))

par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")
ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))

plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="Second Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 45: Beta(3/4,3/4), DGP is convex, impose valid restriction

X <- sort(rbeta(n,3/4,3/4))
h <- npuniden.boundary(X)$h

foo <- npuniden.sc(X=X,h=h,constraint=c("convex"))

par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")
```

```
ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="Second Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 6: N(0,1), DGP is log-concave, impose log-concavity
## restriction

X <- sort(rnorm(n))
h <- npuniden.boundary(X,a=-Inf,b=Inf)$h

foo <- npuniden.sc(X=X,h=h,a=-Inf,b=Inf,constraint=c("log-concave"))

par(mfrow=c(1,2))

ylim <- range(c(log(foo$f.sc),log(foo$f)))
plot(X,log(foo$f.sc),type="l",ylim=ylim,xlab="X",ylab="Log-Density")
lines(X,log(foo$f),col=2,lty=2)
rug(X)
legend("topleft",c("Constrained-log","Unconstrained-log"),lty=1:2,col=1:2,bty="n")

ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="Second Derivative of Log-Density")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained-log","Unconstrained-log"),lty=1:2,col=1:2,bty="n")
par(oldpar)

## End(Not run)
```

---

npunitest *Kernel Consistent Univariate Density Equality Test with Mixed Data Types*

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.
bootstrap	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.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
bw.x, bw.y	numeric (scalar) bandwidths. Defaults to plug-in (see details below).
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, and so on. This is used since we specify bw 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 bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype.

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

Srho	the statistic Srho
Srho.bootstrap	contains the bootstrap replications of Srho
P	the P-value of the statistic
boot.num	number of bootstrap replications
bw.x, bw.y	scalar bandwidths for data.x, data.y

summary supports object of type unitest.

Usage Issues

See the example below for proper usage.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly nonlinear 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

```
## Not run:
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)

## 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)

## 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)

## End(Not run)
```

`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 enrolment 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.

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 <tristen.hayfield@gmail.com>, 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

x	an ordered, unordered or continuous variable.
prob	quantile to compute.

Details

uocquantile is a function which computes quantiles of an unordered, ordered or continuous variable x. If x is unordered, the mode is returned. If x is ordered, the level for which the cumulative distribution is >= prob is returned. If x is continuous, quantile is invoked and the result returned.

Value

A quantile computed from x.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, 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 dependants

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 construction industry

ndurman column 14, of type `numeric`, =1 if in non-durable manufacturing industry

trcommpu column 15, of type `numeric`, =1 if in transportation, communications, public utility

trade column 16, of type `numeric`, =1 if in wholesale or retail

services column 17, of type `numeric`, =1 if in services industry

profserv column 18, of type `numeric`, =1 if in professional services industry

profocc column 19, of type `numeric`, =1 if in professional 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(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