Package: np (via r-universe)

October 26, 2026

Version 0.80-1

Date 2026-10-17

Depends R (>= 3.5.0)

Imports compiler, boot, methods, quadprog, quantreg, stats, crs (>= 0.15-46)

Suggests MASS, logspline, ks, testthat, withr, knitr, rmarkdown, rgl, htmlwidgets

LinkingTo crs (>= 0.15-46)

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 0.0.0

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://sshrc-crsh.canada.ca/en.aspx>), 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

Encoding UTF-8

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

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

NeedsCompilation yes

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

Repository https://cran.r-universe.dev

Date/Publication 2026-10-26 14:20:02 UTC

RemoteUrl https://github.com/cran/np

RemoteRef HEAD

RemoteSha 3c84604ccc17e894b97418a8aa3d8653fad0d1cc

Contents

b.star . . . 3
cps71 . . . 5
dimBS . . . 6
Engel95 . . . 7
gradients . . . 9
Italy . . . 11
np . . . 12
np-data-interfaces . . . 16
np.kernels . . . 17
np.options . . . 25
np.pairs . . . 28
np_plot_controls . . . 29
npcdens . . . 31
npcdensbw . . . 46
npcdenshat . . . 63
npcdist . . . 65
npcdistbw . . . 78
npcdisthat . . . 93
npcmstest . . . 95
npconmode . . . 100
npcopula . . . 113
npdeneqtest . . . 123
npdeptest . . . 127
npindex . . . 131
npindexbw . . . 144
npksum . . . 157
nplsqreg . . . 169
nplsqregbw . . . 175
npplreg . . . 178
npplregbw . . . 187
npqcmstest . . . 198
npqreg . . . 202
npquantile . . . 212
npreg . . . 215
npregbw . . . 231
npreghat . . . 247

npregiv 252
npregivderiv 262
npscoef 269
npscoefbw 275
npsdeptest 288
npseed 291
npsemihat 293
npsigtest 296
npsymtest 303
npudens 307
npudensbw 317
npudenshat 332
npudist 333
npudistbw 342
npudisthat 358
npuniden.boundary 359
npuniden.reflect 364
npuniden.sc 367
npunitest 373
oecdpanel 378
plot.np 379
se 399
uocquantile 400
wage1 401

Index **403**

`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 Input: Time-series data used for automatic block-length selection.

`data`	data, an n x k matrix, each column being a data series.

Block-Length Selection Controls: Tuning constants from Politis and White (2004) and Patton, Politis, and White (2009).

`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)`.

Output Rounding: Control for rounding the selected block lengths.

`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", package = "np")

if (interactive()) with(cps71, plot(age, logwage, xlab="Age", ylab="log(wage)"))
```

---

dimBS *Local-Polynomial Basis Dimension Helper*

Description

`dimBS` returns the number of columns implied by an additive, generalized local-polynomial, or tensor-product basis specification. It is a compatibility wrapper around the internal `dim_basis()` helper used by **np**.

Usage

```
dimBS(basis = "additive",
      kernel = TRUE,
      degree = NULL,
      segments = NULL,
      include = NULL,
      categories = NULL)
```

Arguments

Basis Specification: Basis family, continuous-kernel counting mode, polynomial degree, and segment controls.

`basis`	basis family. One of `"additive"`, `"glp"`, or `"tensor"`.
`kernel`	logical indicating whether only the continuous-kernel basis should be counted. When `FALSE`, optional categorical augmentation controlled by `include` and `categories` is included.
`degree`	non-negative integer vector of local-polynomial degrees.
`segments`	positive integer vector giving the number of segments for each continuous predictor. Defaults to one segment per degree entry.

Categorical Augmentation: Optional categorical-component controls used when `kernel = FALSE`.

`include`	non-negative integer vector indicating which categorical components are included when `kernel = FALSE`.
`categories`	non-negative integer vector giving category counts for included categorical components when `kernel = FALSE`.

Details

dimBS() is provided for compatibility with **crs**. In **np**, the underlying implementation lives in dim_basis(), which is used internally for LP basis-dimension checks and safe NOMAD restart initialization. For kernel=TRUE, basis="glp", and one segment per continuous predictor, the count is the number of non-intercept multi-indices satisfying $0 \leq \alpha_j \leq d_j$ and $\sum_j \alpha_j \leq \max_j d_j$. This unit-segment rule is the local-polynomial contract; non-unit segment and categorical augmentation semantics are unchanged. Counts beyond the supported 100,000-term native GLP capacity fail before basis allocation.

Value

A numeric scalar giving the implied basis dimension.

Examples

```
dimBS(basis = "tensor", degree = c(2, 2))
dimBS(basis = "glp", degree = c(3, 1, 0))
```

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),]
Engel95 <- Engel95[seq_len(min(120, nrow(Engel95))), ]

with(Engel95, {

model.iv <- npregiv(y=food,
                    z=logexp,
                    w=logwages,
                    method="Landweber-Fridman",
                    iterate.max=3,
                    iterate.diff.tol=1.0e-04)
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, se = FALSE, gradient.order = NULL, ...)

## S3 method for class 'condistribution'
gradients(x, se = FALSE, gradient.order = NULL, ...)

## S3 method for class 'lsqregression'
gradients(x, se = FALSE, gradient.order = NULL, ...)

## S3 method for class 'npregression'
gradients(x, se = FALSE, gradient.order = NULL, ...)

## S3 method for class 'qregression'
gradients(x, se = FALSE, ...)
```

```
## S3 method for class 'conmode'
gradients(x, level = NULL, se = FALSE, ...)

## S3 method for class 'singleindex'
gradients(x, se = FALSE, ...)
```

Arguments

Object And Output Controls: Object to interrogate and whether gradient standard errors are requested.

x	an object for which the extraction of gradients is meaningful.
se	a logical value specifying whether or not standard errors of gradients are desired. Defaults to FALSE.
level	for conmode objects, an optional dependent-variable class level identifying the stored class-probability gradients to extract. If supplied, it must match the level requested when the object was fitted.

Derivative Order Controls: Optional local-polynomial derivative order controls.

gradient.order	for npregression, condensity, condistribution, and lsqregression objects fitted with regtype="lp", optional derivative order request (scalar or one entry per continuous predictor). For npregression, condensity, and condistribution, unavailable components whose order exceeds the fitted polynomial degree are returned as NA when another requested continuous component or categorical first-difference effect remains available. Although the formal derivative of the fitted truncated polynomial is zero above its degree, that zero is not an estimate of the corresponding derivative of the unknown target; NA records that the derivative was not estimated. For lsqregression, a supplied order must exactly identify the stored order; unavailable stored components remain NA, while any different requested order is rejected. If another derivative order is desired, compute the fit, prediction, or evaluation with gradients=TRUE and that gradient.order; gradients() does not recompute derivatives after fitting. Gradient standard errors additionally require the object to have been created with se=TRUE.

Additional Arguments: Further method-specific arguments.

...	other arguments.

Details

This function provides a generic interface for extraction of gradients from objects. For npregression, condensity, condistribution, and lsqregression objects fitted with regtype="lp", gradient.order identifies the stored continuous-predictor derivative order to extract; it is not a post-fit recomputation control. For lsqregression, the requested order must exactly match the stored order. For qregression objects, se=TRUE returns asymptotic standard errors for the quantile gradients when the object was fitted with gradients=TRUE.

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")
with(Italy, {

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

})
```

---

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 getting-started vignette containing a short introduction to the **np** package can be accessed via `vignette("np_getting_started", package = "np")`. It includes prediction, no-search uncertainty/gradient refitting, and a migration guide for users updating from 0.70-5. An installed vignette for the entropy-based test family can be accessed via `vignette("np_entropy_tests", 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` (via class-specific S3 methods) can work with the bandwidth object directly without having to do the subsequent companion function evaluation.

The user may also combine these steps. 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 can 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.

Bandwidth objects for regression, unconditional and conditional density/distribution, single-index, partially linear and smooth-coefficient methods retain their training values. Formula objects retain the prepared model frame and NA policy; native objects retain the supplied training values. Fitting with replacement training data records that sample in the returned fit's bandwidth object, without modifying the original bandwidth object. Evaluation or prediction grids do not replace training data. Objects created without retained samples still require their original data bindings or explicit training inputs. New evaluation data involving custom formula transformations still require those transformation functions.

Estimator formulas do not support right-hand-side `offset()` special terms; these are rejected rather than silently omitted. A variable named `offset` is not an offset special. This restriction does not change offset handling by a separately supplied parametric model in a specification test. Conditional density/distribution formulas require response variable names (separated by + for multiple responses). To transform a conditional response, first create an explicit transformed variable in `data`, or supply the transformed values through the native data interface. Supported response transformations in other estimator families, such as `npreg(log(y) ~ x)`, retain their existing meaning.

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.

Estimators support multivariate data subject to each method's documented requirements and available computational resources. Runtime and memory depend on sample size, evaluation points, dimension, polynomial basis, kernel support, bandwidth type and search settings. Large datasets and high-dimensional fits can be expensive; there is no single exponential sample-size cost rule shared by all routines.

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, via class-specific S3 `plot` methods, facilitates graphing of multivariate objects. An example for creating postscript graphs 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. For MPI-oriented workflows on larger jobs, see the companion package **npRmpi**, which extends the same mixed-data kernel methodology to clustered computing environments.

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

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options. 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 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.

Racine, Li, and Yan (x is a (discrete) ordered factor) $l(x_i, x, \lambda) = \lambda^{|x_i - x|} / \sum_{z \in D} \lambda^{|x_i - z|}$, where D is the ordered support.

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.

Racine, J.S., Q. Li, and K.X. Yan (2020), "Kernel Smoothed Probability Mass Functions for Ordered Datatypes," Journal of Nonparametric Statistics, 32(3), 563-586.

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

`np.kernels`, `np.options`, `plot np.options`

`np-data-interfaces` *Formula Data and Explicit Native Inputs*

Description

The role of `data` depends on the selected interface. It is never a request to silently ignore a supplied training sample.

Details

Formula calls use `data` for model-frame construction. Supported refits through a bandwidth object retaining a formula also accept replacement `data`; their established omitted-data and `data = NULL` behavior is unchanged.

Native interfaces use explicitly supplied vectors, matrices or data frames. Where `data` is not a native input, even explicitly supplying `data = NULL` is an error. Omit it, supply inputs such as `xdat = d["x"]` and `ydat = d$y`, or evaluate the native call inside `with(d, ...)`. Native inputs use ordinary R evaluation, not a data mask. Directly supplied, unbound native observation names receive this interface guidance at their existing argument-resolution boundary. No data-mask lookup is attempted. Existing argument evaluation and method dispatch are preserved. An error in a supplied

expression may therefore occur before the native-data diagnostic; MPI methods retain their normal error context. This includes the native IV interfaces: use `with(d, npregiv(y, z, w))` or the supported formula interface.

Some functions genuinely consume a native argument called `data`. In particular, `npcopula(bws, data = observations)`, `b.star`, `npsdeptest` and `npsymtest` retain their documented data inputs. This policy does not change `newdata`, native evaluation arguments, or which input takes precedence during prediction.

Plotting does not generally provide a replacement-data interface. A supplied `data` argument is rejected when the selected plot route does not consume it, including explicit `NULL`. Refit with the intended data and plot the result, or use supported explicit plotting inputs. The existing partially linear formula-bandwidth plot route retains its replacement-data support when its explicit training inputs are omitted.

The diagnostic for unsupported native data is:

```
`data=` is supported only with the formula interface. For native calls, supply vectors, matrices or data
```

See Also

npreg, npregiv, npcopula

`np.kernels` *Kernel Functions Used In* **np**

Description

Summary of continuous, unordered-categorical, and ordered-categorical kernels used by **np** (including higher-order continuous kernels and compact-support variants used in C-level code paths).

Details

Documentation guide: see `np.options` for global options and `plot` for plotting options.

Kernel option names used in **np**:

- Continuous kernels: `ckertype` (and `ckerorder`, `ckerbound` where applicable).
- Unordered kernels: `ukertype`.
- Ordered kernels: `okertype`.
- Conditional density/distribution bandwidth objects split kernel choices by response and regressor blocks: `cykertype`/`cxkertype`, `uykertype`/`uxkertype`, `oykertype`/`oxkertype` (with matching order/bound options for continuous kernels).

Let $u = (x_i - x)/h$ for continuous variables.

Continuous kernels (called via `ckertype`):

$$K_{G,2}(u) = \phi(u)$$

$$K_{G,4}(u) = \left(\frac{3}{2} - \frac{1}{2}u^2\right)\phi(u)$$

$$K_{G,6}(u) = \left(\frac{15}{8} - \frac{5}{4}u^2 + \frac{1}{8}u^4\right)\phi(u)$$

$$K_{G,8}(u) = \left(\frac{35}{16} - \frac{35}{16}u^2 + \frac{7}{16}u^4 - \frac{1}{48}u^6\right)\phi(u)$$

where $\phi(u)$ is the standard normal density.

`ckertype="gaussian"` with `ckerorder=2,4,6,8`.

The compact-support Epanechnikov-family kernels implemented in C use support $|u| < \sqrt{5}$:

$$K_{E,2}(u) = \frac{3}{4\sqrt{5}}\left(1 - \frac{u^2}{5}\right)\mathbf{1}(|u| < \sqrt{5})$$

$$K_{E,4}(u) = 0.008385254916(-15 + 7u^2)(-5 + u^2)\mathbf{1}(u^2 < 5)$$

$$K_{E,6}(u) = 0.33541019662496845446\left(2.734375 - 3.28125u^2 + 0.721875u^4\right)\left(1 - 0.2u^2\right)\mathbf{1}(u^2 < 5)$$

$$K_{E,8}(u) = 0.33541019662496845446\left(3.5888671875 - 7.8955078125u^2 + 4.1056640625u^4 - 0.5865234375u^6\right)\left(1 - 0.\right.$$

`ckertype="epanechnikov"` with `ckerorder=2,4,6,8`.

Uniform (rectangular) kernel:

$$K_U(u) = \frac{1}{2}\mathbf{1}(|u| < 1)$$

via `ckertype="uniform"` (order ignored).

With nearest-neighbor bandwidths, the uniform kernel compares the absolute distance directly with the computed radius: points at the radius are excluded. Tree search uses a conservative candidate box and the same final distance comparison, so rounding of standardized distances or tree endpoints cannot change this boundary convention.

For generalized- and adaptive-nearest-neighbor modes, each continuous bandwidth is the literal kth admitted distance after any occurrence exclusion required by the route. Exact ties can therefore make the radius zero. A zero continuous radius does not define a positive metric bandwidth and is rejected rather than widened to a more distant observation or replaced by a numerical floor. Fixed bandwidths (`bwtype="fixed"`, the default) avoid this nearest-neighbor radius limitation. Alternatively, choose a larger k or correct an unintended continuously coded tie. Repeated values alone are not an error: only a requested radius that is actually zero is rejected. Training fits and external evaluations can differ at a tie boundary because their occurrence exclusions differ. Any represented strictly positive radius, including a positive subnormal value, is distinct from this zero-radius condition.

The beta associated-kernel family is available for continuous data with finite support. For an evaluation point x, observation x_i, fixed support $[a, b]$, $L = b - a$, and public metric bandwidth h, define

$$t = \frac{x - a}{L}, \quad z_i = \frac{x_i - a}{L}, \quad \tau = \left(\frac{L}{h}\right)^2.$$

The implemented PDF contribution is Chen's simple beta kernel

$$K_{Beta,2}(x_i; x, h, a, b) = \frac{1}{L}\frac{z_i^{t\tau}(1 - z_i)^{(1-t)\tau}}{B(1 + t\tau, 1 + (1 - t)\tau)}.$$

Thus ckertype="beta" uses the evaluation-centered associated-kernel orientation: the beta shape parameters depend on x, and the density is evaluated at x_i. The bandwidth convention is deliberately metric-compatible with other continuous kernels: Chen's smoothing parameter is $\rho = (h/L)^2$. The factor $1/L$ is already included, so callers must not apply either an additional $1/h$ factor or the generic bounded-kernel normalization below.

Orders 4, 6, and 8 use the additive higher-order construction of Igarashi and Kakizawa (2020). For public order $2m$,

$$K_{Beta,2m}(x_i; x, h) = \sum_{s=1}^{m} c_{m,s} K_{Beta,2}(x_i; x, h\sqrt{s}), \qquad c_{m,s} = (-1)^{s+1} \binom{m}{s}.$$

The coefficient vectors for orders 2, 4, 6, and 8 are respectively (1), $(2, -1)$, $(3, -3, 1)$, and $(4, -6, 4, -1)$. The construction preserves unit mass and the original support while deliberately producing signed kernels above order 2. The Igarashi–Kakizawa higher-order bias result is a deterministic/fixed- bandwidth result. The generalized- and adaptive-nearest-neighbour routes below implement the same validated multiscale construction conditional on their realized bandwidths; their availability is an implementation and numerical-validation contract, not a claim that the fixed-bandwidth bias theorem automatically extends to random NN bandwidths.

In the present implementation orders 2, 4, 6, and 8 are available through npksum, npudens, npudist, npreg, and the conditional density/distribution families. Regression and the conditional families support local-constant, local-linear and local-polynomial fits; categorical coordinates retain their categorical kernels alongside continuous beta coordinates. Manual and cross-validated bandwidth objects support bwtype="fixed", "generalized_nn", and "adaptive_nn". Bounds may be explicit (ckerbound="fixed") or inferred from the training sample (ckerbound="range"); all active beta bounds must be finite. For sorted distinct values $u_1 < \cdots < u_m$, beta range bounds are

$$A^* = u_1 - (u_2 - u_1)/2, \qquad B^* = u_m + (u_m - u_{m-1})/2.$$

Consequently, beta ckerbound="range" is not equivalent to ckerbound="fixed" at the raw sample minima and maxima; its exact fixed-bound equivalent uses the coordinatewise padded bounds A^* and B^*. Thus every training value is strictly inside the fitted support, including tied extrema, and raw-extremum PDF derivatives are finite. At least two distinct finite values are required per continuous coordinate. This is an empirical continuity convention, not an estimate of physical support: it may extend beyond a known population bound. Use ckerbound="fixed" when bounds are known; fixed bounds are always used literally and retain the documented extended-real derivative convention when observations equal an endpoint. Non-beta ckerbound="range" continues to use exact sample minima and maxima. Rounded or atomic variables should be modelled with the package's discrete kernels rather than made continuous by range padding.

Automatic univariate order-2 fixed-bandwidth beta range searches have an additional basin certificate for density CVLS and distribution CDF criteria when Powell is the selected solver. If $L^* = B^* - A^*$, the official criterion is evaluated once at $h_{tail} = L^* 2^{26}$, where the beta concentration $(L^*/h_{tail})^2$ equals .Machine$double.eps. If this uniform- limit candidate materially improves the ordinary result, one deterministic Powell refinement starts at $h = L^*$; the best materially distinct official objective among the ordinary, tail, and refined candidates is returned. The ordinary search uses the user's unchanged nmulti; the certification and any refinement are separately counted in the bandwidth summary. Cold starts and automatic restarts from a beta range bandwidth object use this same single-certification contract. This safeguard does not apply to explicit fixed bounds, CVML, orders 4–8, nearest-neighbour or multivariate searches, or non-Powell solvers.

Beta continuous coordinates can be combined with unordered and ordered categorical coordinates in kernel sums and supported estimators. Automatic mixed-data bandwidth selection is available for unconditional density, conditional density/distribution, and regression. Regression supports automatic `cv.ls`/`cv.aic` selection with fixed, generalized-NN, and adaptive-NN bandwidths; see `npregbw`. Consult each family's bandwidth help page for its supported beta selection criteria and configurations. Regression and conditional fitting support the canonical scalar and general local-polynomial engines. With `regtype="ll"` or `regtype="lp"`, regression `gradients=TRUE` reports the fitted local-polynomial slope coefficient, as for the other supported kernels; it is not the derivative of the entire fitted smoother with respect to the evaluation point. `npksum` supports categorical products, integer powers, normal, integral, convolution, and one-coordinate PDF-derivative operators; `permutation.operator="derivative"` returns every continuous first derivative. Unsupported score, OCG, or mixed derivative requests fail explicitly rather than entering a legacy kernel code.

For scale component s, put $\tau_s = \tau/s$, $\alpha_s = 1 + t\tau_s$, and $\beta_s = 1 + (1-t)\tau_s$. At an interior observation $0 < z_i < 1$, its evaluation-coordinate derivative is

$$\frac{\partial K_s}{\partial x} = \frac{\tau_s}{L} K_s \left\{ \log z_i - \log(1 - z_i) - \psi(\alpha_s) + \psi(\beta_s) \right\},$$

where ψ is the digamma function. Higher-order derivatives use the same signed multiscale coefficients as the PDF. Bandwidths are held fixed while differentiating: this includes the evaluation bandwidth selected by a generalized-NN rule and each observation bandwidth selected by an adaptive-NN rule. These are derivatives of the fitted smoother conditional on its realized bandwidths, not derivatives of the NN selection map.

Density CVLS quadrature integrates with respect to the continuous support measure. At a boundary observation, it therefore uses the interior-limit PDF contribution rather than assigning integration mass to an isolated endpoint value. This is applied separately to each multiscale component; zero-concentration uniform components remain included. Ordinary PDF point evaluations and the leave-one-out cross term retain their point values. This integration convention alone does not guarantee a finite optimum when an automatic search permits bandwidths tending to zero and the data contain repeated boundary observations. Existing positive search bounds are unchanged.

An observation exactly equal to the lower (upper) support endpoint makes its matching scalar PDF contribution jump downward (upward) when the target crosses that endpoint. The scalar derivative is consequently reported as $-\infty$ at a matching lower endpoint and $+\infty$ at a matching upper endpoint. An observation at the opposite endpoint contributes zero; observations strictly inside the support have finite one-sided derivatives. Local-constant regression and conditional estimators cancel numerator and denominator jumps before classifying their gradient, so a normalized ratio may still have a finite derivative (and a constant response or constant Y factor gives exactly zero). A genuine ratio jump is returned as a signed infinity. Numerically indeterminate cancellation is returned as `NA` with a warning. Derivative hat matrices cannot encode response-specific cancellation and therefore mark an affected endpoint row `NA`. These distinctions follow R's numeric semantics: signed infinity denotes a proper mathematical limit, whereas `NA_real_` denotes an unavailable value. Plot methods preserve the returned values and replace non-finite coordinates only in their temporary drawing geometry; see `plot.np`.

For nearest-neighbor modes, the existing observation-support nearest-neighbor engine supplies the evaluation-specific metric bandwidth $h(x)$ or observation-specific metric bandwidth h_i, after which the same beta concentration mapping is used. For unconditional density and distribution fits, a generalized-NN query identified as a training observation excludes that occurrence when selecting

its radius, for beta as for other continuous kernels. The full fit still averages all donor contributions. Explicit external evaluation data remain external even when their values equal training data. For overlap, generalized bandwidths are evaluated at both centers; adaptive bandwidths remain attached to training centers, while a genuinely new evaluation center receives the established generalized-NN auxiliary bandwidth. PDF, CDF, and overlap operators support both NN modes. See `npksum` for the exact kernel-sum normalization and `npudens` and `npudist` for estimator-specific standard errors and examples.

The implemented beta CDF factor intentionally uses the complementary observation-centered orientation. With the notation above,

$$H_{Beta,2}(x; x_i, h, a, b) = I_t(1 + z_i\tau, 1 + (1 - z_i)\tau),$$

where $I_t(\alpha, \beta)$ is the regularized incomplete beta function. It is evaluated as an analytic incomplete-beta probability, with a symmetry transformation in the upper tail. Thus every order-2 contribution is in $[0, 1]$, and with fixed or adaptive bandwidth is nondecreasing in x; it is exactly zero at a, exactly one at b, and tends to $(x - a)/(b - a)$ as $h \to \infty$. Multivariate CDFs use products of these factors; `npksum` may also mix beta PDF and CDF factors across dimensions.

This observation-centered beta CDF is not the integral, with respect to x, of the evaluation-centered Chen PDF contribution. It is also not the target-centered bounded-CDF estimator of Mansouri, Rastin, and Mombeni (2024). The orientation was selected because it preserves the essential empirical-CDF contribution limit as $h \to 0$, exact endpoint behavior, monotonicity for order 2, and a direct analytic implementation suitable for the existing kernel-sum architecture. Orders 4, 6, and 8 apply the same signed multiscale coefficients to these incomplete-beta factors. The Igarashi–Kakizawa density theorem does not establish a corresponding CDF bias order; the public order identifies the construction. Signed higher-order beta CDFs may overshoot $[0, 1]$ or be locally non-monotone, just as existing higher-order distribution kernels may. Values are not silently clipped or monotonized.

With `bwtype="generalized_nn"`, the CDF bandwidth changes with the target. Every returned order-2 contribution remains an incomplete-beta probability in $[0, 1]$ with exact support endpoints, but the resulting curve need not be monotone. A deterministic boundary-heavy comparison was monotone across the recorded chi-square(1), beta-boundary, U-shaped, and uniform designs, while a seeded 300-design adversarial search found local decreases as large as approximately 8.6×10^{-5}. The implementation neither clips nor monotonizes those values. Inversion-based uses therefore remain outside this generalized-NN CDF contract.

The beta derivative operator differentiates the PDF contribution with respect to its evaluation coordinate. It does not differentiate the observation-centered CDF factor with respect to Y, and it is not a bandwidth score. As with existing higher-order kernels and local-polynomial derivatives, signed kernels can yield non-proper density or distribution estimates; derivative output is not clipped, monotonized, or otherwise repaired.

The beta `"convolution"` operator is the analytic overlap of two associated beta density weights in their common observation coordinate. If the two centers and bandwidths give shapes (α_1, β_1) and (α_2, β_2), then

$$\mathcal{O} = \frac{1}{L} \frac{B(\alpha_1 + \alpha_2 - 1, \beta_1 + \beta_2 - 1)}{B(\alpha_1, \beta_1) B(\alpha_2, \beta_2)}.$$

The diagonal is the exact location- and bandwidth-dependent roughness. This is an L^2 overlap, is symmetric, has units L^{-1}, and is evaluated analytically in C with stable log-gamma ratios. It is not a translation convolution and is not Chen's target-coordinate integrated square. In particular,

bounded `cv.ls` density and conditional-density objectives continue to integrate the actual bounded estimator by their existing quadrature route; they do not substitute this public overlap.

Bounded continuous-kernel normalization (`ckerbound` and, for conditional objects, `cxkerbound`/`cykerbound`) reuses the selected continuous kernel and renormalizes it on the declared support. For a base kernel K and support $[a, b]$, the bounded kernel is

$$K_{[a,b]}(u; x, h) = \frac{K(u)}{\int_{(a-x)/h}^{(b-x)/h} K(t)dt}$$

with $u = (x_i - x)/h$. For bounded adaptive-nearest-neighbor regression, each donor x_j carries its own radius h_j and normalization mass $D(x_j, h_j)$. Its weight at evaluation point x is $K((x-x_j)/h_j)/(h_j D(x_j, h_j))$. Regression fitting and bandwidth objectives use this donor-centred convention, as do the corresponding adaptive density and conditional density/distribution kernel factors. Delete-one objectives use radii from the corresponding deleted sample. Fixed and generalized-neighbor regression use the query-centred mass associated with their query bandwidth. Option `ckerbound="range"` uses sample bounds for a, b; `ckerbound="fixed"` uses user-supplied bounds via `ckerlb`/`ckerub` (or the corresponding `cx*`/`cy*` arguments). Infinite bounds recover the unbounded kernel. This support-normalization strategy follows the same Racine-Li-Yan finite-support normalization principle and is useful when data exhibit non-negligible probability mass near boundaries.

Computationally, let $H_K(r) = \int_0^r K(t)dt$ and $S_K(r) = H_K(r)/r$, with the continuous value $S_K(0) = K(0)$. For finite bounds the implementation evaluates the scaled normalization mass directly as

$$hD(x, h) = (x - a)S_K((x - a)/h) + (b - x)S_K((b - x)/h).$$

Each supported kernel uses its analytic centered primitive. This avoids subtracting nearly equal whole-kernel CDF values and avoids constructing a normalization mass that vanishes numerically as h grows. Consequently, on finite $[a, b]$ the bounded density kernel converges to the uniform density $1/(b - a)$ as $h \to \infty$. One-sided infinite bounds use the corresponding exact half-mass term; two infinite bounds retain the ordinary unbounded kernel path.

For distribution estimation, finite bounds use the distinct observation-centered truncated-kernel CDF

$$B_i(y; h) = \frac{G((y - Y_i)/h) - G((a - Y_i)/h)}{G((b - Y_i)/h) - G((a - Y_i)/h)}, \quad y \in [a, b].$$

Thus $B_i(a; h) = 0$, $B_i(b; h) = 1$, and $B_i(y; h) \to (y - a)/(b - a)$ as $h \to \infty$. The implementation evaluates centered analytic interval masses rather than subtracting nearly equal whole-kernel CDF values. This distribution operator is not the integral of the evaluation-centered bounded density kernel above. For nonnegative second-order kernels each contribution is monotone and lies in $[0, 1]$; higher-order signed kernels retain their usual possible interior nonmonotonicity or overshoot.

Typical bounded-kernel calls:

```
## Unconditional density on [0,1]
bw <- npudensbw(dat=data.frame(x),
                ckertype="gaussian",
                ckerbound="fixed", ckerlb=0, ckerub=1)

## Regression with automatic sample-range bounds
bw <- npregbw(xdat=data.frame(x), ydat=y, ckerbound="range")
```

```
## Conditional density with separate x/y support controls
bw <- npcdensbw(xdat=data.frame(x), ydat=data.frame(y),
                cxkerbound="fixed", cxkerlb=0, cxkerub=1,
                cykerbound="range")
```

Unordered-categorical kernels (called via `ukertype`; for category count c):

$$L_{AA}(x_i, x; \lambda) = \mathbf{1}(x_i = x)(1 - \lambda) + \mathbf{1}(x_i \neq x)\frac{\lambda}{c - 1}$$

(Aitchison-Aitken) via `ukertype="aitchisonaitken"`.

$$L_{LR,u}(x_i, x; \lambda) = \frac{\mathbf{1}(x_i = x) + \mathbf{1}(x_i \neq x)\lambda}{1 + (c - 1)\lambda}$$

(normalized Li-Racine unordered kernel returned by `npksum`) via `ukertype="liracine"`. Ratio estimators may equivalently use the proportional unnormalized weights, since this common factor cancels in their numerator and denominator.

Ordered-categorical kernels (called via `okertype`): Numeric ordered-factor levels retain their numeric distances, including gaps. They must be finite and increasing with a finite numeric span. Li-Racine weights and Racine-Li-Yan kernels accept fractional distances in the original units; there is no inferred lattice spacing, rescaling, or arbitrary denominator cap. Wang-van Ryzin and normalized Li-Racine retain unit-lattice definitions: they require integer distances and total span less than `.Machine$integer.max`. Nonnumeric ordered labels use their declared ordinal ranks: for `ordered(c("a","b","c","d","e"))`, the distance from `a` to `e` is four, from `c` to `d` one. Use explicit nonnumeric labels when rank distance, not numeric distance, is intended. Lattice admission is deliberately conservative: decimal labels at large magnitudes can lose enough precision that integer distances cannot be established reliably. Such lattice-dependent requests report an error; they do not round the labels or substitute another kernel. Select `okertype="racineliyan"` explicitly for retained-support smoothing.

$$L_{WvR}(x_i, x; \lambda) = \begin{cases} 1 - \lambda, & x_i = x \\ \frac{1-\lambda}{2}\lambda^{|x_i - x|}, & x_i \neq x \end{cases}$$

(Wang-van Ryzin) via `okertype="wangvanryzin"`.

$$L_{LR,o}(x_i, x; \lambda) = \lambda^{|x_i - x|}$$

(Li-Racine ordered kernel) via `okertype="liracine"`.

$$L_{NLR,o}(x_i, x; \lambda) = \lambda^{|x_i - x|}\frac{1 - \lambda}{1 + \lambda}$$

(normalized Li-Racine ordered kernel; used internally)

$$L_{RLY}(x_i, x; \lambda) = \frac{\lambda^{|x_i - x|}}{\sum_{z \in \mathcal{S}(x)} \lambda^{|x_i - z|}}$$

(Racine-Li-Yan ordered kernel, normalized on support $\mathcal{S}(x)$). exposed as `okertype="racineliyan"`. Here the support is the retained set of factor levels, including unused levels, not an integer interval filling its gaps. The normalizer is indexed by the training value x_i, so weights need not be symmetric in training and evaluation values. Cumulative kernels, overlap integrals, scores and paired regression computations use that same support and direction.

For density/distribution roles, the public spelling `"liracine"` denotes normalized Li-Racine and therefore still requires a unit lattice; select `"racineliyan"` for fractional response/density support. Conditional explanatory roles and regression use raw Li-Racine weights. On nonlattice support, the raw Li-Racine cumulative operator sums these unnormalized weights over retained levels at or below the evaluation point; it is not a probability CDF. On a unit lattice, its cumulative operator instead sums over every lattice point between the smallest and largest retained levels up to the evaluation cutoff, including points in gaps. Its convolution sums products over the retained levels, not the gap-filled lattice interval. These are distinct incumbent operator conventions. A fractional evaluation cutoff does not change the retained integration support or turn a lattice cumulative into a nonlattice sum. If a near-integer numeric support is ambiguous at its floating-point precision, the raw Li-Racine cumulative operator reports an error rather than silently selecting a different integration convention. Clearly fractional support remains supported. Racine-Li-Yan provides an explicit retained-support cumulative definition in either case. At $\lambda = 0$, bandwidth scores with positive distances below one have non-finite elementwise one-sided derivatives and report a condition; use an interior bandwidth when requesting such scores. Kernel values at zero and one remain supported.

These univariate kernels are combined as generalized product kernels over mixed data types in the estimators and cross-validation criteria.

References

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

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

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

Racine, J. S. and Li, Q. (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," *Journal of Econometrics*, **119**, 99–130.

Racine, J. S., Li, Q., and Yan, K. X. (2020), "Kernel Smoothed Probability Mass Functions for Ordered Datatypes," *Journal of Nonparametric Statistics*, **32**(3), 563–586. doi:10.1080/10485252.2020.1759595

Chen, S. X. (1999), "Beta kernel estimators for density functions," *Computational Statistics & Data Analysis*, **31**(2), 131–145. doi:10.1016/S0167-9473(99)00010-9

Jones, M. C. and Henderson, D. A. (2007), "Kernel-type density estimation on the unit interval," *Biometrika*, **94**(4), 977–984. doi:10.1093/biomet/asm068

Igarashi, G. and Kakizawa, Y. (2020), "Higher-order bias corrections for asymmetric kernel density estimators," *Journal of Nonparametric Statistics*, **32**(3), 716–752. doi:10.1080/10485252.2020.1770754

Mansouri, B., Rastin, A., and Allah Mombeni, H. (2024), "Beta Kernel Estimator for a Cumulative Distribution Function with Bounded Support," *Journal of Sciences, Islamic Republic of Iran*, **34**(4), 349–361 (volume dated 2023; accepted and DOI registered in 2024). doi:10.22059/jsciences.2024.367933.1007836

Hall, P., Racine, J. S., and Li, Q. (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, **99**, 1015–1026.

See Also

np.options, plot npregbw, npudensbw, npudistbw, npcdensbw, npcdistbw, npksum, np.options.

---

np.options *Global Package Options for* **np**

Description

Global options controlling selected computational and display behavior for the **np** package.

Details

Documentation guide: see np.kernels for kernels and plot for plotting options.

The following options are recognized by **np**.

Output and Plot Progress:

- np.messages (logical): controls console/progress output. Default is TRUE.
- np.plot.progress (logical): controls bounded plot/bootstrap progress heartbeats. Default is TRUE.
- np.progress.interval.sec (numeric): package-wide minimum elapsed time between visible progress heartbeats. The default is 2.0 seconds. When set, this canonical option takes precedence over the older surface-specific interval options.
- np.plot.progress.start.grace.sec (numeric): delay before the first plot/bootstrap progress line is shown. Default is 0.75.
- np.plot.progress.interval.sec (numeric): minimum elapsed time between plot/bootstrap heartbeat lines once progress reporting has started when np.progress.interval.sec is unset. Default is 2.0.
- np.plot.progress.max.intermediate (integer): maximum number of mid-run plot/bootstrap heartbeat lines emitted between the initial start notice and final completion line. Default is 3.

Kernel Routing and Acceleration:

- np.tree (logical or "auto"): controls kd-tree acceleration for continuous variables. TRUE forces eligible tree routes on, FALSE disables them, and "auto" enables trees only when active continuous kernels are bounded support ("epanechnikov" or "uniform"). Default is "auto". Conditional-distribution generalized-neighbor bandwidth selection admits explicit TRUE for second-order Epanechnikov and uniform kernels without boundary normalization. The same explicit selection is available for local-constant and linear/polynomial conditional density/distribution fitting from generalized-neighbor CVLS bandwidth objects. Linear/polynomial adaptive-neighbor fitting reuses bounded evaluation-tree batches. Automatic-selection policies are unchanged. The same explicit compact-kernel selection is available for linear and polynomial regression generalized- and adaptive-neighbor cv.ls/cv.aic bandwidth selection, and for generalized-neighbor linear/polynomial regression fitting and its hat/apply operators. Adaptive-neighbor linear/polynomial regression fitting also supports this explicit selection, using donor-radius evaluation-tree batches. GNN fitting may use dense accumulation when an exact support certificate proves that tree traversal cannot prune

any donor; kernel weights remain varying, and no occupancy or conditioning threshold is used. Fixed-bandwidth regression fitting and regression automatic-selection policies are unchanged. Tree and dense accumulation can differ numerically on nearly singular local-polynomial designs; enabling trees does not change the basis, ridge or rank policies.

- `np.categorical.compress` (logical): enables exact unique-profile compression for eligible all-categorical routes. This is distinct from `np.tree`: no continuous kd-tree is requested merely because categorical compression is enabled. The default is `TRUE`. Set `options(np.categorical.compress = FALSE, np.tree = FALSE)` to force the historical dense categorical route for debugging or timing comparisons.
- `np.macMseries.accelerate` (logical or `"auto"`): controls macOS arm64 Accelerate/vDSP/vForce fast paths for eligible Gaussian-kernel and weighted-reduction computations. `"auto"` enables these paths only when they were compiled for macOS arm64 with Accelerate support; `TRUE` requests them when available; and `FALSE` disables them exactly. Default is `"auto"`. On non-macOS, non-arm64, or non-Accelerate builds, this option falls back to the ordinary portable code path. Accelerated and portable paths may use different math-library kernels or summation orders, so platform or option comparisons should use numerical tolerances; reserve exact equality for controlled runs using the same acceleration setting.

Search and Objective Caching:

- `np.objective.cache` (logical): enables package-side evaluation caching for eligible repeated bandwidth-search candidates. For all Powell routes using the shared C optimizer wrapper, exact repeated raw candidate vectors within one bandwidth solve reuse the previous objective value, including repeated candidates reached by later Powell restarts. For eligible continuous-only generalized/adaptive nearest-neighbor routes, this also enables the existing integer nearest-neighbor objective cache. Mixed continuous/categorical rows use only the exact full-candidate cache, so categorical smoothing parameters remain part of the cache key. Default is `TRUE`. Eligible continuous `npscoefbw` and `npindexbw` routes also use package-side caches around their family-native R `optim` evaluations. Set `options(np.objective.cache = FALSE)` to force repeated eligible candidates through the evaluator for diagnostic timing comparisons. Each native Powell-side cache has an independent 64 MiB peak workspace ceiling, including replacement storage during growth. If the ceiling or an allocation prevents growth, existing entries remain available and the selected search continues without inserting new entries; bandwidth cache diagnostics report this through `allocation.failed`. MADS/NOMAD solver caching is separate and is not controlled by this option, and the saturated extended-NN distance cache is governed by `np.extendednn`. Bandwidth summaries keep `Number of Function Evaluations` as the established cumulative optimizer evaluation count and report NOMAD, Powell, R-`optim`, nearest-neighbor, degree-search, guarded-evaluation, and other fast-CV diagnostics on separate lines when the corresponding counters are available. These diagnostics describe their own optimizer/cache layer and are not intended to decompose the function-evaluation count arithmetically.
- `np.nomad.degree.start.policy` (character): controls automatic polynomial-degree starting values for NOMAD local-polynomial searches when `degree.start` is not supplied. The default, `"low_first_full_random"`, uses `degree.min` for the first start and draws later starts over the full user-specified `degree.min`/`degree.max` range. Other accepted values are `"mid_first_full_random"`, `"anchor_then_random"`, `"spread_then_random"`, and `"random_full_only"`. Explicit `degree.start` values remain the first start.

Bandwidth Shortcuts:

- `np.largeh` (logical): enables continuous large-h shortcut evaluations. Default is `TRUE`. Set `options(np.largeh = FALSE)` to keep tree/dense routing unchanged while forcing continuous large-bandwidth objective rows through the ordinary kernel-summation machinery for diagnostic timing comparisons.
- `np.extendednn` (logical): enables extended nearest-neighbor bandwidth search and manually supplied nearest-neighbor bandwidths above the ordinary $n-1$ neighbor-count limit for eligible generalized/adaptive nearest-neighbor routes. When `TRUE`, a nearest-neighbor search parameter above $n-1$ scales the saturated nearest-neighbor bandwidth. Default is `TRUE`.
- `np.largeh.rel.tol` (numeric): relative tolerance used by the continuous large-h shortcut. When all standardized distances for a continuous predictor are sufficiently close to zero, the corresponding kernel factor is approximated by $K(0)$ to reduce repeated kernel evaluations. Default is `1e-3`. Valid range is `(0, 0.1)`.
- `np.largelambda` (logical): enables discrete near-upper bandwidth shortcut evaluations. Default is `TRUE`. Set `options(np.largelambda = FALSE)` to keep categorical routing unchanged while forcing near-upper discrete bandwidth rows through the ordinary category-comparison machinery for diagnostic timing comparisons.
- `np.disc.upper.rel.tol` (numeric): relative tolerance used by the discrete upper-bound shortcut for bandwidths near their feasible upper bounds. The near-upper check is applied relative to each kernel's own feasible upper bound (e.g., Aitchison-Aitken depends on category cardinality), with a tiny machine-precision floor for numerical robustness. When same/different-category kernel values are numerically close, the corresponding discrete kernel factor is treated as constant to reduce repeated category comparisons. Default is `1e-2`. Valid range is `(0, 0.5)`.

Plot Layout:

- `plot.par.mfrow` (logical): used by `plot` to determine whether plotting layout is automatically managed via `par(mfrow=...)`. If `NULL` (default behavior), **np** uses its internal plotting defaults.

Option values can be set globally via `options` and restored with `on.exit` in scripts/functions for reproducibility.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

`np.kernels`, `plot np`, `plot`, `options`

Examples

```
## Not run:
old <- options(
  np.tree = "auto",
  np.categorical.compress = TRUE,
  np.macMseries.accelerate = "auto",
  np.objective.cache = TRUE,
  np.largeh = TRUE,
```

```
  np.extendednn = TRUE,
  np.largelambda = TRUE,
  np.messages = FALSE,
  np.largeh.rel.tol = 1e-3,
  np.disc.upper.rel.tol = 1e-2
)
on.exit(options(old), add = TRUE)

## ... run bandwidth selection / estimation ...

## End(Not run)
```

`np.pairs` *Cross-Validated Pairs Plot (Helper Functions)*

Description

Compute pairwise nonparametric regressions and densities for a set of variables, then plot a pairs-style display with fitted smoothers.

Usage

```
np.pairs(y.vars, y.dat, ...)
np.pairs.plot(pair.list)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the variables, data, and pair specifications to plot.

`pair.list`	list returned by `np.pairs`.
`y.dat`	data frame containing the variables listed in `y.vars`.
`y.vars`	character vector of column names in `y.dat`. If `y.vars` is named, the names are used as plot labels.

Additional Arguments: Further graphical arguments are passed through to plotting methods.

`...`	additional arguments passed to `npudens` and `npreg`.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

On the diagonal, `npudens` is used to compute kernel density estimates. Off-diagonal panels use `npreg` with residuals to draw scatterplots and smoothers.

Value

np.pairs returns a list with components y_vars, pair_names, and pair_kerns. np.pairs.plot returns NULL (invisibly).

See Also

np.kernels, np.options, plotnpudens, npreg

Examples

```
## Not run:
data("USArrests")
y.vars <- c("Murder", "UrbanPop")
names(y.vars) <- c("Murder Arrests per 100K", "Pop. Percent Urban")

pair.list <- np.pairs(
  y.vars = y.vars,
  y.dat = USArrests,
  ckertype = "epanechnikov",
  bwscaling = TRUE
)

np.pairs.plot(pair.list)

## End(Not run)
```

---

np_plot_controls *Validated plot control helpers*

Description

Constructors for detailed plot controls used by the redesigned **np** plotting interface. Common choices such as errors, band, alpha, bootstrap, and B remain direct plot arguments; these helpers collect less common options.

Usage

```
np_boot_control(nonfixed = c("exact", "frozen"),
                wild = c("rademacher", "mammen"),
                blocklen = NULL)

np_grid_control(xtrim = NULL, xq = NULL, slices = NULL)

np_render_control(style = c("band", "bar"),
                  bar = c("|", "I"),
                  bar.num = NULL)
```

Arguments

nonfixed	Bootstrap behavior for non-fixed bandwidth routes. "exact" reconstructs each duplicate-row resample and recomputes nearest-neighbour bandwidths; "frozen" reuses the bandwidths from the fitted object.
wild	Wild-bootstrap multiplier distribution.
blocklen	Optional block length for block bootstrap methods.
xtrim	Optional two-element trim range for evaluation grids.
xq	Optional evaluation quantiles.
slices	Optional slice specification for routes that support it.
style	Interval display style.
bar	Bar style when style = "bar".
bar.num	Optional number of bars.

Details

Gaussian simultaneous variability bands use the Gaussian-multiplier framework of Chernozhukov, Chetverikov and Kato (2013, 2015). For the density/distribution families, related simultaneous-inference research is due to Cattaneo, Jansson and Ma (2024) and Cattaneo, Chandak, Jansson and Ma (2024). See plot.np for the full published references, implementation scope and estimator-specific qualifications.

These helpers validate names and values early. Raw lists are deliberately not accepted as substitutes, so misspelled detailed controls fail before plot computation begins.

Compound public plot arguments use dots, consistently with the package API. Constructor function names and returned object fields are unchanged. Common rendering controls such as data.overlay, data.rug, layout, and output are supplied directly to plot(...) rather than through np_render_control().

Gaussian simultaneous variability bands use the direct plot argument gaussian.draws (default 9999), including supported density and distribution plots. This controls Gaussian calibration only; B and np_boot_control() continue to control bootstrap resampling. See plot.np for supported targets, covariance contracts and the references to Chernozhukov, Chetverikov and Kato (2013, 2015).

Conditional density and distribution level bootstraps permit different supported continuous kernel families and orders on the X and Y sides, including beta on either side. Beta-containing routes remain limited to continuous X and Y with local-constant fitting and retain the resolved support bounds stored in the bandwidth object. The nonfixed choice changes nearest-neighbour bandwidth replay, not those fitted support bounds.

Value

A small classed list consumed by plot methods.

Examples

```
## Not run:
plot(fit,
     errors = "bootstrap",
```

```
        bootstrap = "inid",
        B = 9999,
        band = "pointwise",
        boot.control = np_boot_control(nonfixed = "exact"))

## End(Not run)
```

---

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 'conbandwidth'
npcdens(bws,
        txdat = stop("invoked without training data 'txdat'"),
        tydat = stop("invoked without training data 'tydat'"),
        exdat,
        eydat,
        gradients = FALSE,
        gradient.order = 1L,
        proper = FALSE,
        proper.method = c("project"),
        proper.control = list(),
        ...,
        se = FALSE)

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

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

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.

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

txdat — a p-variate data frame of sample realizations of explanatory data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

tydat — a q-variate data frame of sample realizations of dependent data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside npcdens.

nomad — logical shortcut passed through to npcdensbw when bandwidths are computed inside npcdens. When TRUE, the bandwidth route fills any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npcdensbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the fitted conditional density is evaluated and which estimates are returned.

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.

se — logical scalar, default FALSE. If TRUE, compute and retain available asymptotic standard errors of fitted values and, when gradients = TRUE, of the requested

gradients. If FALSE, skip uncertainty-only computation; point estimates and gradients are unchanged. Unavailable derivative errors remain NA when requested. To obtain errors without repeating bandwidth selection, refit using npcdens(bws = fit$bws, se = TRUE); also supply gradients = TRUE for gradient errors. Explicit training data may be supplied if they are no longer recoverable from the bandwidth object. The argument must be named.

gradients — a logical value specifying whether to return estimates of the gradients at the evaluation points. Defaults to FALSE.

gradient.order — derivative order for continuous explanatory-variable gradients when gradients = TRUE and regtype = "lp". A scalar is recycled across continuous explanatory variables, or one value may be supplied per continuous explanatory variable. The default 1L returns first derivatives. Higher orders are available for continuous explanatory variables only and must not exceed the corresponding local-polynomial degree, apart from the established all-degree-zero first-derivative compatibility route. An unavailable continuous component is returned as NA; plotting retains it as an empty panel when another continuous derivative or categorical effect is available. A request for which no component is available is rejected. Unordered and ordered predictors retain their usual first-order discrete effects.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Fit Properization Controls: These arguments control optional post-estimation properization of the fitted conditional density.

proper — a logical value specifying whether to apply post-estimation properization to the conditional density estimate. Defaults to FALSE. Local-constant or degree-zero fits bypass correction only when their kernels and bandwidth geometry guarantee nonnegative, unit-mass response components and nonnegative predictor weights. Signed kernels, generalized-nearest-neighbour response bandwidths and target-centred bounded or beta response densities do not have this guarantee. Requested correction then uses the existing supported grid or slice procedure, with its additional evaluation cost and support requirements.

proper.control — a list of controls for properization. Supported entries are tol, grid.check, store.raw, and fail.on.unsupported. Resolved controls are retained in fitted objects and inherited by predict() and plot() unless explicitly replaced. An explicit control list replaces the retained list; unspecified entries then use their defaults. Prediction still errors on unsupported correction unless fail.on.unsupported=FALSE is explicitly supplied.

proper.method — a character string specifying the properization method. Currently "project" is supported.

Additional Arguments: Further arguments are passed to npcdensbw when bandwidths are computed internally, or used to interpret a numeric bws vector.

... — additional arguments supplied to npcdensbw when npcdens computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth selection controls such as bwmethod, bwtype, kernel/support

controls such as cxkertype, cykertype, cxkerbound, and cykerbound, search controls such as nmulti, scale.factor.search.lower, and nomad.nmulti, and local-polynomial controls such as regtype, degree, basis, and bernstein.basis are supplied. See npcdensbw for the complete bandwidth-selection argument surface.

Details

Conditional fitting and replay use the literal categorical kernels for both explanatory and response variables. Near-upper categorical bandwidths are not replaced by approximate constant factors. The same target is used for levels, gradients, standard errors, conditional hats, and reconstructed plot uncertainty. This conditional-family policy does not change direct regression or unconditional kernel-sum evaluation and does not modify the caller's np.largelambda option.

With generalized nearest-neighbor bandwidths, query identity matters when selecting a radius: a training observation excludes its own occurrence, whereas explicit evaluation data remain external queries even if their values equal training observations. Consequently, predict(fit, newdata = training.data) can differ from fitted(fit). This does not omit the training contribution from the fitted estimator. See np.kernels for the radius convention.

With a formula-derived bandwidth object, supplying only tydat replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective na.action is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use data for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named newdata columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over newdata.

For a native bandwidth object, newdata is split into exdat and eydat using the retained predictor and response column names. Names must identify both roles unambiguously; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

For explicit external evaluation rows, a completed explanatory-kernel row whose weights are all zero has no conditional estimate. Such rows return NA for the fitted value and requested uncertainty or derivatives, with one notice. A missing categorical counterfactual affects only its contrast; it does not invalidate an otherwise supported base estimate. A zero response-kernel contribution is not a missing explanatory row. Supported rows retain their usual computation. Required internal evaluations remain strict. With proper = TRUE, complete unsupported fixed-X slices retain NA; projection still applies to supported slices.

Documentation guide: see npcdensbw for bandwidth selection and search controls, np.kernels for kernels, np.options for global options, and plot, plot.np for plotting options.

When bws is omitted, the formula and default methods call npcdensbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a conbandwidth object, npcdens estimates with the stored bandwidth metadata in that object.

Beta associated kernels may be selected independently for explanatory X coordinates (`cxkertype="beta"`) and response Y coordinates (`cykertype="beta"`). Orders 2, 4, 6, and 8, fixed/generalized-NN/adaptive-NN bandwidths, explicit finite bounds, sample-range bounds, and automatic `cv.ml`/`cv.ls` bandwidth selection are supported. Beta kernels act on continuous coordinates and can coexist with supported categorical X or Y coordinates using their respective categorical kernels. Local-constant, local-linear and local-polynomial fits are supported. For local-constant fits, analytic first derivatives with respect to the continuous X coordinates are available with `gradients=TRUE`, including beta-X/legacy-Y, legacy-X/beta-Y, fixed, generalized-NN, and adaptive-NN combinations. NN bandwidths are held fixed while differentiating. Exact beta-X endpoint jumps are cancelled in the normalized ratio before a gradient is classified; genuine jumps are signed infinities, while indeterminate cancellation is `NA`, both with an explanatory warning. Returned infinities retain that extended-real meaning. Plot methods convert them to missing geometry only in a temporary drawing copy, leaving fitted and plot-data objects unchanged. For ordinary conditional-density level plots, bootstrap intervals support different continuous X- and Y-side kernel families or orders, including beta on either side. Beta-containing bootstrap levels use the same signed log-domain conditional ratio as the fitted estimator. With non-fixed bandwidths, `np_boot_control(nonfixed="exact")` reconstructs each duplicate-row resample and recomputes its nearest-neighbour bandwidths, while `nonfixed="frozen"` keeps the original fitted bandwidths. In both modes, resolved beta `"range"` support bounds are replayed from the bandwidth object rather than recomputed from bootstrap counts. This mixed-family level-bootstrap support does not alter the separate restrictions on gradient, quantile, or bias-corrected centers. Higher-order fits are signed and need not be proper densities; no positivity repair is applied. Bounded beta-response `cv.ls` integrates the actual conditional estimator over the Y support and does not replace that integral by the public observation-coordinate beta overlap.

Argument groups for bandwidth selection are documented on `npcdensbw`. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls, numerical search controls, bounded `cv.ls` quadrature controls if relevant, and finally local-polynomial/NOMAD controls for polynomial-adaptive fits.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

`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.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, `npcdens(..., nomad=TRUE)` and `npcdensbw(..., nomad=TRUE)` expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

With a raw polynomial basis, high powers of poorly scaled continuous predictors (such as uncentred calendar years) can produce severely ill-conditioned systems. Conditional local-polynomial fits do not apply the design-conditioning rejection used by `npreg`; they may return inaccurate finite values without a warning, with or without a ridge adjustment. Centre or scale such predictors before fitting and apply the same transformation to all evaluation or prediction data. The explicit `bernstein.basis = TRUE` representation is another option for improving conditioning. Check numerical stability for the chosen data, degree and bandwidths. `proper = TRUE` enforces probability-shape constraints; it does not repair an ill-conditioned polynomial solve.

With `regtype = "lp"`, `gradients = TRUE` and `gradient.order` greater than one expose higher-order derivative estimates with respect to continuous explanatory variables. These derivatives use the same local-polynomial basis, degree, Bernstein option, bandwidths, and kernels as the fitted conditional density. See `npreg` for the complete heterogeneous GLP multi-index and Bernstein-representation definitions shared by these routes. With `se = TRUE`, available higher-order derivative columns for fixed and adaptive nearest-neighbour bandwidths also have standard errors. These combine the existing local pseudo-response variance with the squared norm of the actual derivative weighting row, including its accepted ridge adjustment. They condition on the selected bandwidths, neighbour radii and local-polynomial map; they do not account for their selection uncertainty. Generalized nearest-neighbour higher-order point derivatives remain subject to the existing hat-operator restriction. Bootstrap inference remains recommended.

With positive-degree local-polynomial fits (including local-linear fits), numerical ridging follows the policies described in `npreg` (Seifert and Gasser, 2000). The conditional pseudo-response uncertainty formulas here are distinct from ordinary regression's residual normalization.

With positive-degree local-polynomial fits (including local-linear fits), categorical first-difference standard errors use paired endpoint influence calculations. The covariance between the two category endpoints is retained, not replaced by the sum of their marginal variances. These calculations condition on the selected basis, smoothing parameters and realized neighbour radii for all three bandwidth types. They run only when both `gradients = TRUE` and `se = TRUE` request them. A non-smooth regularization boundary or nonfinite influence gives `NA` uncertainty with a notice, without changing the point contrast.

For ordinary scalar fits, leading interior standard errors use the squared X-kernel moment; conditional density errors additionally use the independent Y-kernel moment. Analytic first-derivative errors for fixed and generalized nearest-neighbour bandwidths with Gaussian or Epanechnikov kernels use the squared derivative-kernel moment and the X bandwidth. These formulas do not establish uniform-kernel derivative inference or full adaptive-NN inference with estimated random radii. Positive-degree local-polynomial fits use their separate covariance calculation.

For fixed and generalized nearest-neighbour scalar fits with categorical variables, these leading moments retain squared categorical kernel factors and the joint training-category frequencies. Conditional distributions, and conditional densities with no continuous response, retain the numerator–denominator covariance of the ratio estimator. In particular, a constant response-kernel contribution has zero conditional variance. Purely categorical fits use this ratio covariance without a continuous bandwidth factor, including nearest-neighbour-labelled bandwidth objects. Beta-kernel and positive-degree local-polynomial covariance calculations retain their separate targets.

For scalar adaptive nearest-neighbour fits with any continuous variable, requested level and available continuous first-derivative standard errors use the complete empirical influence of the conditional ratio. They retain the actual normalized X weights and continuous/categorical Y contributions, including numerator and denominator derivatives. This working covariance holds the selected model, observed X geometry and realized response nearest-neighbour radii fixed; it does not account for their sampling or selection uncertainty. Local-constant-equivalent degree-zero fits share this calculation, including continuous responses with categorical-only X. Positive-degree local-polynomial, beta-kernel and paired categorical- contrast errors retain their separate covariance calculations. With `se = FALSE`, the additional uncertainty pass is not performed.

Value

The `se` component records whether uncertainty was requested. `conderr` and `congerr` are `NULL` when `se = FALSE`. `se()` and `gradients(..., se = TRUE)` only extract stored results and give a no-search refit message when the errors were omitted. `predict(..., se.fit = TRUE)` explicitly requests the needed errors.

`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. An evaluation row whose required explanatory normalization is exactly zero or non-finite is returned as `NA`; other rows remain available.
`conderr`	when `se = TRUE`, standard errors of the conditional density estimates. Undefined normalization rows are returned as `NA`.
`congrad`	if invoked with `gradients = TRUE`, estimates of the gradients at the evaluation points. For local-polynomial fits, continuous-coordinate derivative orders are controlled by `gradient.order`. Undefined normalization rows are returned as `NA`.
`congerr`	if invoked with `gradients = TRUE` and `se = TRUE`, standard errors of the gradients at the evaluation points, including available higher-order continuous derivatives under the working-covariance assumptions described above. Undefined normalization or uncertainty rows are returned as `NA`.
`log_likelihood`	aggregate fitted conditional-density log likelihood. Every represented strictly positive estimate, including a positive subnormal value, contributes its literal logarithm. An exact zero or negative estimate has no mathematical logarithm and is represented in this aggregate by the established computational placeholder `log(DBL_MIN)`. Consequently this component is not a literal log likelihood if any fitted density is nonpositive.

Book And Method Pointers

The conditional density target is $f(y \mid x)$. In the product-kernel formulation this can be viewed as estimating the joint mixed-data density of (Y, X) and normalizing by the marginal density of X, with local-polynomial routes using the selected continuous-coordinate polynomial degree in the conditional fit. The requested standard errors are asymptotic standard errors for the fitted conditional density values, and gradient standard errors are returned when gradients are requested and defined.

Scalar conditional fits involving beta kernels use the empirical observation-influence covariance. For normalized explanatory weights a_i, response-kernel contributions z_i, and fitted value m, the level variance is $\{n/(n-1)\} \sum_i [a_i(z_i - m)]^2$, using the full training count n. Supported continuous-derivative errors use the corresponding differentiated influence. Categorical contrasts use the difference of the two endpoints' observation influences inside the same squared norm, thereby retaining their covariance; self-contrasts have zero variance. These calculations hold the selected bandwidth state fixed; they do not add bandwidth-selection or random-NN-radius uncertainty, and do not replace positive-degree LP covariance.

For book-length derivations, see Li and Racine (2007), Chapter 5 *Conditional Density Estimation*, especially Sections 5.1-5.4, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

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.

Chen, S.X. (1999), "Beta kernel estimators for density functions," Computational Statistics & Data Analysis, 31(2), 131–145. doi:10.1016/S0167-9473(99)00010-9.

Igarashi, G. and Kakizawa, Y. (2020), "Higher-order bias corrections for asymmetric kernel density estimators," Journal of Nonparametric Statistics, 32(3), 716–752. doi:10.1080/10485252.2020.1770754.

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

np.kernels, np.options, plot, plot.np 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(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), nmulti=1)

# 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).

if (interactive()) plot(bw)
```

```
})

# 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(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, nmulti=1)

# 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).

if (interactive()) plot(bw)

})

# 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")
with(faithful, {

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

bw <- npcdensbw(formula=eruptions~waiting, nmulti=1)
```

```
summary(bw)

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

if (interactive()) plot(bw)

})

# 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")
with(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, nmulti=1)

summary(bw)

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

if (interactive()) plot(bw)

})

# 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 <- 300

# 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)

data.train <- data.frame(y = factor(y), x1 = x1, x2 = x2)

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

bw <- npcdensbw(formula=y~x1+x2, data=data.train, nmulti=1)

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

x1.seq <- seq(min(x1), max(x1), length=30)
x2.seq <- seq(min(x2), max(x2), length=30)
X.eval <- expand.grid(x1=x1.seq,x2=x2.seq)

data.eval <- data.frame(
  y=factor(rep(1, nrow(X.eval)), levels=levels(data.train$y)),
  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, 30, 30)

# 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 <- 300

# 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)
y.factor <- factor(y)

# 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=y.factor, nmulti=1)

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

x1.seq <- seq(min(x1), max(x1), length=30)
x2.seq <- seq(min(x2), max(x2), length=30)
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)), levels=levels(y.factor)),
               bws=bw))

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

fit.mat <- matrix(fit, 30, 30)

# 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 4: Variations on local polynomial conditional density
# estimation with proper = TRUE.

data("Italy")

# Work in years relative to 1975 to condition the raw polynomial basis.
# Calendar year = model year + year.offset, also for prediction data.
year.offset <- 1975
Italy2 <- within(Italy, {
  year <- as.numeric(as.character(year)) - year.offset
})

# Plot only: make the plotted surface proper on the plot evaluation grid.

fhat <- npcdens(gdp ~ year, data = Italy2,
                regtype = "lp", degree = 3, nmulti = 1)

plot(fhat, proper = TRUE)

# Fit an object whose fitted values are themselves proper.

ctrl_fit <- list(
  mode = "slice",
  apply = "fitted",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_fit <- npcdens(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_fit
)

fit_proper <- fitted(fhat_fit)
```

```
fit_raw <- fhat_fit$condens.raw

# Display the repaired and raw fitted values for cases where the raw
# fitted density is negative.

head(cbind(fit_proper, fit_raw)[which(fit_raw < 0), ])

# Predict on a common explicit y-grid for several years, and render
# those predictions proper.

g.grid <- seq(min(Italy2$gdp), max(Italy2$gdp), length.out = 200)

nd_grid <- expand.grid(
  gdp = g.grid,
  year = c(1955, 1975, 1995) - year.offset
)

pred_grid <- predict(fhat, newdata = nd_grid, proper = TRUE)

# Predict on paired rows with different gdp grids by year, and still
# make the predictions proper via slice mode.

g1 <- seq(quantile(Italy2$gdp, 0.10),
          quantile(Italy2$gdp, 0.60), length.out = 60)
g2 <- seq(quantile(Italy2$gdp, 0.30),
          quantile(Italy2$gdp, 0.90), length.out = 35)

nd_slice <- rbind(
  data.frame(gdp = g1, year = rep(1960 - year.offset, length(g1))),
  data.frame(gdp = g2, year = rep(1985 - year.offset, length(g2)))
)

pred_slice <- predict(
  fhat,
  newdata = nd_slice,
  proper = TRUE,
  proper.control = list(mode = "slice")
)

# One object that carries properization for fitted values and for later
# predict() calls.

ctrl_both <- list(
  mode = "slice",
  apply = "both",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_both <- npcdens(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
```

```
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_both
)

fit_both <- fitted(fhat_both)
pred_both <- predict(
  fhat_both,
  newdata = nd_slice,
  proper.control = ctrl_both
)

## 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 'conbandwidth'
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
```

```
            initc.dir = 1.0,
            initd.dir = 1.0,
            invalid.penalty = c("baseline","dbmax"),
            itmax = 10000,
            lbc.dir = 0.5,
            scale.factor.init.lower = 0.1,
            lbd.dir = 0.1,
            lbd.init = 0.1,
            memfac = 500,
            nmulti,
            penalty.multiplier = 10,
            nomad.remin = FALSE,
            powell.remin = TRUE,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample = FALSE,
            scale.factor.search.lower = NULL,
            cvls.quadrature.grid = NULL,
            cvls.quadrature.extend.factor = NULL,
            cvls.quadrature.points = NULL,
            cvls.quadrature.ratios = NULL,
            small = 1.490116e-05,
            tol = 1.490116e-04,
            transform.bounds = FALSE,
            ...,
            nomad.opts = list())

## Default S3 method:
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws,
          bandwidth.compute = TRUE,
          bwmethod,
          bwscaling,
          bwtype,
          cfac.dir,
          scale.factor.init,
          cxkerbound,
          cxkerlb,
          cxkerorder,
          cxkertype,
          cxkerub,
          cykerbound,
          cykerlb,
          cykerorder,
          cykertype,
          cykerub,
          dfac.dir,
          dfac.init,
```

```
dfc.dir,
ftol,
scale.factor.init.upper,
hbd.dir,
hbd.init,
initc.dir,
initd.dir,
invalid.penalty,
itmax,
lbc.dir,
scale.factor.init.lower,
lbd.dir,
lbd.init,
memfac,
nmulti,
oxkertype,
oykertype,
penalty.multiplier,
nomad.remin = FALSE,
powell.remin,
bwsolver = c("powell", "mads", "mads+powell"),
scale.init.categorical.sample,
scale.factor.search.lower = NULL,
cvls.quadrature.grid = c("hybrid", "uniform", "sample"),
cvls.quadrature.extend.factor = 1,
cvls.quadrature.points = c(100L, 50L),
cvls.quadrature.ratios = c(0.20, 0.55, 0.25),
small,
tol,
transform.bounds,
uxkertype,
uykertype,
regtype = c("lc", "ll", "lp"),
basis = c("glp", "additive", "tensor"),
degree = NULL,
degree.select = c("manual", "coordinate", "exhaustive"),
search.engine = c("nomad+powell", "cell", "nomad"),
nomad = FALSE,
nomad.nmulti = 0L,
degree.min = NULL,
degree.max = NULL,
degree.start = NULL,
degree.restarts = 0L,
degree.max.cycles = 20L,
degree.verify = FALSE,
bernstein.basis = FALSE,
...,
nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, and whether bandwidths are supplied or computed.

`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`.

`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.

`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.

`data`
: This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

  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.

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

`na.action`
: a function which indicates what should happen when the data contain `NA`s. 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`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

`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.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search when `regtype="lp"`.

degree.max optional scalar or integer vector giving upper bounds for automatic degree search over continuous xdat predictors when degree.select != "manual".

degree.max.cycles positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for "manual" and "exhaustive" degree selection.

degree.min optional scalar or integer vector giving lower bounds for automatic degree search over continuous xdat predictors when degree.select != "manual".

degree.restarts non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for "manual" and "exhaustive" degree selection.

degree.select character string controlling local-polynomial degree handling when regtype="lp". "manual" (default) treats degree as fixed. "coordinate" performs cached coordinate-wise search over admissible degree vectors for the continuous xdat predictors. "exhaustive" evaluates the full admissible degree grid when search.engine="cell". For NOMAD-based search engines, any non-"manual" value requests direct joint search over degree and bandwidth coordinates.

degree.start optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous xdat predictors.

degree.verify logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for search.engine="cell".

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

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, normal-consistent median absolute deviation, interquartile range divided by $2\Phi^{-1}(0.75)$), 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, normal-consistent median absolute deviation, interquartile range divided by $2\Phi^{-1}(0.75)$), 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

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

dfac.dir — stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

dfac.init — non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

hbd.dir — upper bound for direction set search for Powell's algorithm for categorical variables. See Details

hbd.init — upper bound for scale factors for categorical variables for Powell's algorithm. See Details

initd.dir — initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbd.dir — lower bound for direction set search for Powell's algorithm for categorical variables. See Details

lbd.init — lower bound 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

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

cfac.dir — stretch factor for direction set search for Powell's algorithm for numeric variables. See Details

dfc.dir — chi-square degrees of freedom for direction set search for Powell's algorithm for numeric variables. See Details

initc.dir — initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbc.dir — lower bound for direction set search for Powell's algorithm for numeric variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

cxkerbound — character string controlling continuous-kernel support handling for xdat. Can be set as none (default kernel on full support), range (use sample min/max, except that beta kernels use the outer half-spacing rule in np.kernels), or fixed (use cxkerlb/cxkerub). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see np.kernels.

cxkerlb numeric scalar/vector of lower bounds for continuous xdat variables used when cxkerbound="fixed". Must satisfy lower-bound validity for each variable (e.g., <= min(variable)). Use -Inf for unbounded below. See np.kernels for bounded-kernel normalization details.

cxkerub numeric scalar/vector of upper bounds for continuous xdat variables used when cxkerbound="fixed". Must satisfy upper-bound validity for each variable (e.g., >= max(variable)). Use Inf for unbounded above. See np.kernels for bounded-kernel normalization details.

cykerbound character string controlling continuous-kernel support handling for ydat. Can be set as none (default kernel on full support), range (use sample min/max, except that beta kernels use the outer half-spacing rule in np.kernels), or fixed (use cykerlb/cykerub). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see np.kernels.

cykerlb numeric scalar/vector of lower bounds for continuous ydat variables used when cykerbound="fixed". Must satisfy lower-bound validity for each variable (e.g., <= min(variable)). Use -Inf for unbounded below. See np.kernels for bounded-kernel normalization details.

cykerub numeric scalar/vector of upper bounds for continuous ydat variables used when cykerbound="fixed". Must satisfy upper-bound validity for each variable (e.g., >= max(variable)). Use Inf for unbounded above. See np.kernels for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

scale.factor.init
deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to 0.5. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is max(scale.factor.init, scale.factor.search.lower). See Details.

scale.factor.init.lower
lower endpoint for random continuous scale-factor starts. Defaults to 0.1. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). See Details.

scale.factor.init.upper
upper endpoint for random continuous scale-factor starts. Defaults to 2.0. It must be greater than or equal to the effective lower endpoint, max(scale.factor.init.lower, scale.factor.search.lower); otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels for xdat and ydat.

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.

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

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

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

oykertype character string used to specify the ordered categorical kernel type for ydat. Can be set as wangvanryzin, liracine, or racineliyan. Defaults to liracine.

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

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

Least-Squares Quadrature Controls: These controls tune quadrature for bounded continuous-response least-squares cross-validation.

cvls.quadrature.extend.factor
a positive finite scalar controlling the finite numerical integration window used by bounded conditional-density cv.ls quadrature when one or both continuous response-side bounds are infinite. Finite bounds are used literally. Defaults to 1.

cvls.quadrature.grid
character string specifying the one-dimensional bounded cv.ls I_1 quadrature grid. "uniform" uses only evenly spaced nodes over the finite quadrature window, "sample" uses deterministic ranked sample-y nodes, and "hybrid" uses a fixed-size merge controlled by cvls.quadrature.ratios. The default is "hybrid" for scalar continuous responses and "uniform" for two continuous responses.

cvls.quadrature.points
a two-element integer vector giving the bounded cv.ls quadrature point counts for one and two continuous response variables, respectively. Defaults to c(100L, 50L). For two continuous response variables, the second entry is used per dimension.

cvls.quadrature.ratios
a three-element non-negative numeric vector summing to one, giving the uniform, ranked sample-y, and composite Gauss-Legendre proportions used by the scalar bounded cv.ls "hybrid" quadrature grid. Defaults to c(0.20, 0.55, 0.25), which gives an 20/55/25 split when cvls.quadrature.points = c(100L, 50L) and the nearest deterministic exact-count split at other scalar grid sizes. The setting is ignored by explicit "uniform" and "sample" grid modes.

When response-side bounds are set explicitly to fixed infinite endpoints, bounded cv.ls uses a finite numerical quadrature surrogate over the data range extended by cvls.quadrature.extend.factor. In that edge case, callers who want tighter agreement with the ordinary unbounded convolution route should set cvls.quadrature.points explicitly.

Local-Polynomial Model Specification: These arguments control the local-polynomial estimator, basis, and fixed degree specification.

basis — character string specifying the polynomial basis used when regtype="lp". Options are "glp", "additive", and "tensor".

bernstein.basis — logical value controlling the stabilized representation for regtype="lp". For basis="glp", TRUE selects the complete degree-graded, orthonormal shifted-Legendre definition given in npreg; additive and tensor bases retain their fixed-coordinate-degree representation. When automatic degree search is requested and bernstein.basis is not explicitly supplied, the search route defaults to TRUE for numerical stability. Explicit bernstein.basis=FALSE is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

degree — integer scalar or integer vector of polynomial degrees for continuous xdat variables when regtype="lp". If scalar, the value is recycled to all continuous xdat variables. With no continuous xdat predictors, regtype="lp" is only admissible when degree=0, the local-constant equivalent.

regtype — character string specifying the conditional local method used for the xdat regression weight operator. Options are "lc", "ll", and "lp". For npc* methods, "ll" is implemented via the canonical local polynomial engine with degree = 1 and basis = "glp". "ll" and positive-degree "lp" require at least one continuous xdat predictor; for categorical-only xdat, use "lc" or "lp" with degree=0. If local-linear cv.ls search fails while using this canonical raw basis, retrying explicitly with regtype="lp", degree=1, and bernstein.basis=TRUE, or centering/scaling the continuous regressors, can improve numerical conditioning without changing package defaults or invoking an automatic fallback.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

nomad — logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When TRUE, any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype are filled with regtype="lp", search.engine="nomad+powell", degree.select="coordinate", bernstein.basis=TRUE, degree.min=0L, degree.max=10L, degree.verify=FALSE, and bwtype="fixed". Explicit incompatible settings error immediately; in particular, explicit nomad=TRUE currently requires regtype="lp", bwtype="fixed", automatic degree search, no explicit degree, at least one continuous xdat predictor, and search.engine %in% c("nomad", "nomad+powell"). This shortcut does not change the meaning of nmulti or nomad.nmulti: nmulti remains the outer restart count, while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in bw$nomad.shortcut for a returned object bw; when available, nomad.time and powell.time record the direct-search and Powell-polish timing components.

nomad.nmulti non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when regtype="lp" and automatic degree search uses search.engine="nomad" or "nomad+powell". Defaults to 0L, which preserves the current one-start-per- restart behavior. This does not replace nmulti: nmulti controls outer restarts, while nomad.nmulti controls inner NOMAD multistarts within each outer restart.

nomad.opts optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. NULL is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as NB_THREADS_PARALLEL_EVAL > 1.

nomad.remin logical flag controlling the optional second NOMAD hot start. When TRUE, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to FALSE; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

search.engine character string controlling the automatic local-polynomial search backend when regtype="lp" and degree.select != "manual". "nomad+powell" (default) performs direct joint search over the xdat-side continuous bandwidth coordinates and degree vector using the native **crs** NOMAD C API, then applies one Powell hot start from the NOMAD solution. "nomad" omits the Powell refinement. "cell" profiles the criterion over the admissible degree grid using repeated fixed-degree bandwidth solves. NOMAD-based search supports bwtype="fixed", bwtype="generalized_nn", and bwtype="adaptive_nn"; it currently requires degree.verify=FALSE and the suggested package **crs** to be installed.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, memory blocking, and bounded search transformations.

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)).

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".

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

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.

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

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

powell.remin logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, TRUE restarts the local search from the located minimum. For search.engine="nomad+powell", this controls only the final Powell bandwidth-polish step. The default is TRUE for ordinary Powell routes and FALSE for the Powell polish after NOMAD unless explicitly supplied.

bwsolver bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Use search.engine rather than bwsolver for automatic degree search. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

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)).

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)).

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.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

... additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via newdata does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state.

Beta associated kernels may be selected independently with cxkertype="beta" and cykertype="beta". Orders 2, 4, 6, and 8, fixed/generalized-NN/adaptive-NN bandwidths, cxkerbound/cykerbound equal to "fixed" or "range", explicit bandwidth objects, and automatic cv.ml/cv.ls search are

supported. Normal-reference selection is not. Beta kernels act on continuous coordinates alongside the kernels for categorical coordinates. Local-constant, local-linear and local-polynomial fits are supported. Local-constant gradients with respect to X are available: a beta X coordinate uses its analytic evaluation-coordinate PDF derivative, while the beta or legacy Y density factor is held fixed. Exact matching beta-X endpoints use structural cancellation in the normalized ratio and can yield signed infinity or NA when a finite derivative does not exist. Bounded beta-response `cv.ls` uses Y-support quadrature of the actual estimator with cached X weights. Signed orders 4, 6, and 8 are not positivity-repaired.

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)`, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is NULL, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

For automatic fixed-bandwidth CVLS with a beta response kernel, repeated complete observations at a declared response endpoint require an explicitly positive resolved `scale.factor.search.lower`. The default `0.1` already satisfies this requirement. Endpoint repetitions can cause the criterion to prefer vanishing bandwidths; this is a conservative search-admissibility policy, not a claim that every such sample has a divergent criterion. Manual bandwidth evaluation and nearest-neighbor searches remain available. No positive floor is silently substituted for an explicitly requested zero.

A selected scale factor at this floor is a boundary solution of the constrained search, not evidence of an interior optimum. In particular, endpoint observations can make beta CVLS sensitive to the chosen floor. For a returned fixed-bandwidth object `bw`, inspect `bw$sfactor$x[bw$ixcon]` and `bw$sfactor$y[bw$iycon]` alongside `bw$scale.factor.search.lower`; physical bandwidths are on a different scale and should not be compared directly with this dimensionless bound. Near-bound solutions merit the same sensitivity consideration, but proximity alone does not prove that the unconstrained criterion diverges. Changing the floor changes the admissible search problem.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

For `bwmethod="cv.ml"`, the ordinary contribution is based on $\log(\hat{f}_i)$. Compact-support kernels can yield zero leave-one-out conditional-density estimates, and local-polynomial or higher-order-kernel fits can yield negative estimates. In those cases the package optimizes a guarded CVML criterion rather than the literal log likelihood. Every represented strictly positive fitted value, including a positive subnormal value retained in log form, uses the ordinary branch $-\log(\hat{f}_i)$. Let ϵ denote the positive machine floor used only to encode an invalid nonpositive contribution. Exact zero and negative values with magnitude no greater than ϵ use $-\log(\epsilon)$; negative values of larger magnitude use the smooth penalty $\log(-\hat{f}_i) - 2\log(\epsilon)$. The `guarded` count reported in bandwidth summaries records objective evaluations containing an invalid nonpositive contribution; a positive

value is not guarded merely because it is small. The bwmethod="cv.ls" criterion remains well defined for negative kernel density estimates and is not modified by this CVML guard.

Documentation guide: see np.kernels for kernels, np.options for global options, and plot, plot.np for plotting options.

The bandwidth-selection argument surface is easiest to read by decision group. Start with the data and bandwidth inputs (xdat, ydat, bws, and bandwidth.compute), then choose the bandwidth criterion and representation (bwmethod, bwscaling, and bwtype). Next choose continuous kernel and support controls (cxker* and cyker*), categorical kernel controls (uxkertype, uykertype, oxkertype, and oykertype), and numerical search controls including nmulti, tolerances, penalties, and the scale.factor.* search-start and admissibility controls. Bounded continuous-response cv.ls fits may also use the cvls.quadrature.* controls. Local-polynomial and NOMAD controls (regtype, basis, degree*, search.engine, nomad, nomad.nmulti, and bernstein.basis) are relevant when using the explicit local-polynomial route.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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. For fixed local-constant/local-linear fits, and for local-polynomial fits with degree.select="manual", bandwidth search uses multidimensional Powell direction-set optimization.

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.

Computational cost of generalized nearest-neighbor CVLS: For unbounded Gaussian, Epanechnikov, and uniform kernels, bwmethod="cv.ls" with bwtype="generalized_nn" integrates the squared leave-one-out conditional density over the full response support, allowing response bandwidths to vary with the response evaluation point. This requires additional integration work compared with the analytical convolutions available for fixed and adaptive nearest-neighbor bandwidths.

This whole-response criterion also applies to beta predictor kernels and bounded Gaussian predictor kernels when the response domain is unbounded. Predictor boundary correction does not make the response bandwidth constant.

Ambiguous local QR rank in a conditional local-polynomial design uses the existing original-coordinate rank/ridge admission. If a numerical evaluation remains unusable, including a failed coefficient accuracy check or exhausted integration limits, it receives the existing invalid-evaluation penalty and search continues. A penalty is not a verified cross-validation score; the final selected bandwidth must pass the existing raw-objective validation.

Computation time depends on the kernels, nearest-neighbor counts, predictor and response dimensions, and local-polynomial specification. Some configurations can be substantially more expensive. Second-order kernels are the usual practical choice; higher-order kernels remain supported. The cvls.quadrature.* controls described above apply to bounded response integration, not this unbounded generalized-neighbor calculation.

Exact ties can make the literal kth continuous nearest-neighbor radius zero. Such a bandwidth is undefined and is rejected; see np.kernels for the precise contract and remedies.

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.

Response expressions such as log(y) or I(y^2) are not supported on the left-hand side and are rejected. Instead create, for example, data$log_y <- log(data$y) and use log_y ~ x, or supply the transformed response through ydat. Right-hand-side offset() special terms are also unsupported and are rejected rather than omitted. These checks do not evaluate response expressions, alter stored/dynamically constructed formulas, or change supported explanatory transformations.

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.

When regtype="lp" and degree.select != "manual", npcdensbw can jointly determine the xdat-side local polynomial degree vector and the fixed bandwidth coordinates entering the conditional density criterion. With search.engine="cell", the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search. With search.engine="nomad" or "nomad+powell", the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API; "nomad+powell" then performs one Powell hot start from the NOMAD solution and keeps the better of the direct NOMAD and polished answers. This polynomial-adaptive joint-search route is motivated by Hall and Racine (2015) together with Li, Li, and Racine (under revision). When bernstein.basis is not explicitly supplied, the automatic search route defaults to bernstein.basis=TRUE for numerical stability.

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For conditional density bandwidth selection it expands any missing values to the equivalent long-form call

```
npcdensbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
```

```
                bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value `nomad="auto"` applies the same LP shortcut but leaves `search.engine` and `degree.select` eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit `search.engine` or `degree.select` choices are honored.

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.

Prediction follows a two-stage contract. Calling `predict(bw)` on the returned bandwidth object is a convenience forwarder for `npcdens(bws = bw)` and returns a fitted `condensity` object; it does not return a numeric vector of conditional-density estimates. Call `fitted(fit)` or `predict(fit, newdata = ...)` on that fitted object to obtain numeric results. The canonical workflow is

```
bw <- npcdensbw(y ~ x, data = mydata)
fit <- predict(bw)             # equivalent to npcdens(bws = bw)
fitted(fit)
predict(fit, newdata = newdata)
```

The functions `summary` and `plot` also support `conbandwidth` objects.

Book And Method Pointers

Local-polynomial search uses the numerical-ridging policies described in npreg; see Seifert and Gasser (2000) there.

npcdensbw selects response and conditioning bandwidths, and when requested local-polynomial degree/search metadata, for the conditional density target $f(y \mid x)$ estimated by npcdens.

For book-length derivations, see Li and Racine (2007), Chapter 5 *Conditional Density Estimation*, especially Sections 5.1-5.4, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

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(2, 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.

Hall, P. and J.S. Racine (2015), "Infinite Order Cross-Validated Local Polynomial Regression," Journal of Econometrics, 185, 510-525.

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

Li, A. and Q. Li and J.S. Racine (under revision), "Boundary Adjusted, Polynomial Adaptive, Nonparametric Kernel Conditional Density Estimation," Econometric Reviews.

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

np.kernels, np.options, plot, plot.np 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(Italy, {

bw <- npcdensbw(formula=gdp~ordered(year), nmulti=1)

# 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.

})

# 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(Italy, {

bw <- npcdensbw(xdat=ordered(year), ydat=gdp, nmulti=1)

# 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.

})

## End(Not run)
```

---

npcdenshat *Conditional Density Hat Operator*

Description

Constructs the conditional density hat operator associated with npcdens bandwidth objects. The returned operator maps a right-hand side y to Hy; with y = 1 this reproduces the fitted conditional density.

Usage

```
npcdenshat(bws,
           txdat = stop("training data 'txdat' missing"),
           tydat = stop("training data 'tydat' missing"),
           exdat,
           eydat,
           y = NULL,
           output = c("matrix", "apply"),
           deriv = NULL,
           s = NULL)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted conditional density bandwidth object of class "conbandwidth".
exdat	Optional evaluation conditioning data. If omitted, the operator is built on the training conditioning data.
eydat	Optional evaluation response data. If omitted, the operator is built on the training response data.
txdat	Training conditioning data used to construct the operator.
tydat	Training response data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

deriv	Convenience alias for s.
output	Either "matrix" for the hat matrix or "apply" for direct application to y.
s	Optional first-derivative multi-index over continuous conditioning variables. By default the conditional-density operator itself is returned. When s requests one first derivative, the returned operator maps y to the corresponding npcdens(..., gradients = TRUE) conditioning-variable gradient component.
y	Optional right-hand side vector or matrix with one row per training observation.

Details

For the adaptive-nearest-neighbour local-constant (including degree-zero local-polynomial) ratio, every finite nonzero conditioning-weight sum is used without an epsilon floor. If all computed conditioning weights at an external evaluation row are zero, that row is returned as NA with a warning. This describes computed weights, not necessarily absence of geometric support. Zero response weights still give a valid zero numerator when the conditioning denominator is defined. Required internal and bootstrap ratios remain strict.

For output = "matrix", the return value is a matrix with class c("npcdenshat", "matrix") and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For output = "apply", the function returns H y directly. Matrix right-hand sides are applied column-wise.

The optional s argument follows the derivative multi-index convention used by npreghat for continuous conditioning variables. The default s = NULL returns the ordinary conditional-density hat operator. Supplying one first derivative, for example s = 1L in a univariate continuous-x problem or s = c(x2 = 1L) in a multivariate problem, returns the hat operator for that component of the conditional-density gradient. This derivative route is intended to match npcdens(..., gradients = TRUE)$congrad.

For local-constant beta X kernels, first derivatives use the analytic beta PDF derivative. Exact support-endpoint rows whose raw weights jump are returned as NA, because a response-free hat matrix cannot perform the response-specific numerator/denominator cancellation available to the fitted conditional-density estimator.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class "npcdenshat" or the applied result H y, depending on output.

Examples

```
## Not run:
data(cps71)
tx <- data.frame(age = cps71$age)
ty <- data.frame(logwage = cps71$logwage)

bw <- npcdensbw(xdat = tx, ydat = ty, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = c(1.0, 1.0))
```

```
H <- npcdenshat(bws = bw, txdat = tx, tydat = ty)
dens.hat <- npcdenshat(bws = bw, txdat = tx, tydat = ty,
                       y = rep(1, nrow(tx)),
                       output = "apply")
dens.core <- fitted(npcdens(bws = bw, txdat = tx, tydat = ty))

head(cbind(dens.core, dens.hat), n = 2L)

grad.hat <- npcdenshat(bws = bw, txdat = tx, tydat = ty,
                       y = rep(1, nrow(tx)),
                       output = "apply", s = 1L)
grad.core <- gradients(npcdens(bws = bw, txdat = tx, tydat = ty,
                               gradients = TRUE))[, 1L]
head(cbind(grad.core, grad.hat), n = 2L)

## 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 'condbandwidth'
npcdist(bws,
        txdat = stop("invoked without training data 'txdat'"),
        tydat = stop("invoked without training data 'tydat'"),
        exdat,
        eydat,
        gradients = FALSE,
        gradient.order = 1L,
        proper = FALSE,
        proper.method = c("isotonic"),
        proper.control = list(),
        ...,
```

```
        se = FALSE)

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

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

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.
data	This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces. 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.
txdat	a p-variate data frame of sample realizations of explanatory data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.
tydat	a q-variate data frame of sample realizations of dependent data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside npcdist.

nomad	logical shortcut passed through to npcdistbw when bandwidths are computed inside npcdist. When TRUE, the bandwidth route fills any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npcdistbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the fitted conditional distribution is evaluated and which estimates are returned.

`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`.

`se` logical scalar, default `FALSE`. If `TRUE`, compute and retain available asymptotic standard errors of fitted values and, when `gradients = TRUE`, of the requested gradients. If `FALSE`, skip uncertainty-only computation; point estimates and gradients are unchanged. Unavailable derivative errors remain `NA` when requested. To obtain errors without repeating bandwidth selection, refit using `npcdist(bws = fit$bws, se = TRUE)`; also supply `gradients = TRUE` for gradient errors. Explicit training data may be supplied if they are no longer recoverable from the bandwidth object. The argument must be named.

`gradients` a logical value specifying whether to return estimates of the gradients at the evaluation points. Defaults to `FALSE`.

`gradient.order` derivative order for continuous explanatory-variable gradients when `gradients = TRUE` and `regtype = "lp"`. A scalar is recycled across continuous explanatory variables, or one value may be supplied per continuous explanatory variable. The default `1L` returns first derivatives. Higher orders are available for continuous explanatory variables only and must not exceed the corresponding local-polynomial degree, apart from the established all-degree-zero first-derivative compatibility route. An unavailable continuous component is returned as `NA`; plotting retains it as an empty panel when another continuous derivative or categorical effect is available. A request for which no component is available is rejected. Unordered and ordered predictors retain their usual first-order discrete effects.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Fit Properization Controls: These arguments control optional post-estimation properization of the fitted conditional distribution.

`proper` a logical value specifying whether to apply post-estimation properization to the conditional distribution estimate. Defaults to `FALSE`. Local-constant or degree-zero fits bypass correction only when their kernels and bandwidth geometry guarantee a convex mixture of proper response CDFs. Signed kernels and generalized-nearest-neighbour response bandwidths do not have this guarantee. Requested correction then uses the existing supported grid or slice procedure, with its additional evaluation cost and support requirements.

`proper.control` a list of controls for properization. Supported entries are `tol`, `grid.check`, `store.raw`, and `fail.on.unsupported`. Resolved controls are retained in fitted objects and inherited by `predict()` and `plot()` unless explicitly replaced. An explicit control list replaces the retained list; unspecified entries then use their

defaults. Prediction still errors on unsupported correction unless `fail.on.unsupported=FALSE` is explicitly supplied.

`proper.method` a character string specifying the properization method. Currently `"isotonic"` is supported.

Additional Arguments: Further arguments are passed to `npcdistbw` when bandwidths are computed internally, or used to interpret a numeric `bws` vector.

`...` additional arguments supplied to `npcdistbw` when `npcdist` computes bandwidths internally, or arguments needed to interpret a numeric `bws` vector. This is where bandwidth selection controls such as `bwmethod`, `bwtype`, kernel/support controls such as `cxkertype`, `cykertype`, `cxkerbound`, and `cykerbound`, search controls such as `nmulti`, `scale.factor.search.lower`, and `nomad.nmulti`, and local-polynomial controls such as `regtype`, `degree`, `basis`, and `bernstein.basis` are supplied. See `npcdistbw` for the complete bandwidth-selection argument surface.

Details

Conditional fitting and replay use the literal categorical kernels for both explanatory and response variables. Near-upper categorical bandwidths are not replaced by approximate constant factors. The same target is used for levels, gradients, standard errors, conditional hats, and reconstructed plot uncertainty. This conditional-family policy does not change direct regression or unconditional kernel-sum evaluation and does not modify the caller's `np.largelambda` option.

With generalized nearest-neighbor bandwidths, query identity matters when selecting a radius: a training observation excludes its own occurrence, whereas explicit evaluation data remain external queries even if their values equal training observations. Consequently, `predict(fit, newdata = training.data)` can differ from `fitted(fit)`. This does not omit the training contribution from the fitted estimator. See `np.kernels` for the radius convention.

With a formula-derived bandwidth object, supplying only `tydat` replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective `na.action` is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use `data` for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named `newdata` columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over `newdata`.

For a native bandwidth object, `newdata` is split into `exdat` and `eydat` using the retained predictor and response column names. Names must identify both roles unambiguously; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

For explicit external evaluation rows, a completed explanatory-kernel row whose weights are all zero has no conditional estimate. Such rows return `NA` for the fitted value and requested uncertainty

or derivatives, with one notice. A missing categorical counterfactual affects only its contrast; it does not invalidate an otherwise supported base estimate. A zero response-kernel contribution is not a missing explanatory row. Supported rows retain their usual computation. Required internal evaluations remain strict. With `proper = TRUE`, complete unsupported fixed-X slices retain `NA`; projection still applies to supported slices.

Documentation guide: see `npcdistbw` for bandwidth selection and search controls, `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

When `bws` is omitted, the formula and default methods call `npcdistbw` first and pass bandwidth-selection arguments from `...` to that call. When `bws` is already a `condbandwidth` object, `npcdist` estimates with the stored bandwidth metadata in that object.

Beta associated kernels may be selected independently for explanatory X coordinates (`cxkertype="beta"`) and response Y coordinates (`cykertype="beta"`). Orders 2, 4, 6, and 8, fixed/generalized-NN/adaptive-NN bandwidths, explicit finite bounds, sample-range bounds, and automatic `cv.ls` bandwidth selection are supported. Beta kernels act on continuous coordinates alongside the kernels for categorical coordinates. Local-constant, local-linear and local-polynomial fits are supported. For local-constant fits, analytic first derivatives with respect to continuous X coordinates are available with `gradients=TRUE`; the Y-side CDF factor itself is not differentiated. Fixed/adaptive order-2 Y CDF factors are proper on their declared support; generalized-NN order-2 curves may be locally nonmonotone, and signed orders 4, 6, and 8 may also leave $[0, 1]$. No clipping or monotone repair is applied. At exact beta-X endpoints, numerator and denominator jumps are cancelled before classifying the conditional-distribution gradient. A constant Y CDF factor (in particular at its exact support endpoints) gives a structural zero gradient. Genuine ratio jumps are reported as signed infinities and indeterminate numerical cancellations as `NA`, with warnings. Returned infinities retain that extended-real meaning. Plot methods convert them to missing geometry only in a temporary drawing copy, leaving fitted and plot-data objects unchanged. For ordinary conditional-distribution level plots, bootstrap intervals support different continuous X- and Y-side kernel families or orders, including beta on either side. Beta-containing bootstrap levels use the same signed log-domain conditional ratio as the fitted estimator. With non-fixed bandwidths, `np_boot_control(nonfixed="exact")` reconstructs each duplicate-row resample and recomputes its nearest-neighbour bandwidths, while `nonfixed="frozen"` keeps the original fitted bandwidths. In both modes, resolved beta `"range"` support bounds are replayed from the bandwidth object rather than recomputed from bootstrap counts. This mixed-family level-bootstrap support does not alter the separate restrictions on gradient, quantile, or bias-corrected centers.

With finite continuous response bounds, `npcdist` uses the observation-centered truncated-kernel CDF documented in `np.kernels`. Each response contribution is exactly zero and one at the declared lower and upper endpoints and has the linear uniform-CDF limit as the response bandwidth diverges.

Argument groups for bandwidth selection are documented on `npcdistbw`. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls, numerical search controls, and finally local-polynomial/NOMAD controls for polynomial-adaptive fits.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

`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.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, `npcdist(..., nomad=TRUE)` and `npcdistbw(..., nomad=TRUE)` expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

With a raw polynomial basis, high powers of poorly scaled continuous predictors (such as uncentred calendar years) can produce severely ill-conditioned systems. Conditional local-polynomial fits do not apply the design-conditioning rejection used by `npreg`; they may return inaccurate finite values without a warning, with or without a ridge adjustment. Centre or scale such predictors before fitting and apply the same transformation to all evaluation or prediction data. The explicit `bernstein.basis = TRUE` representation is another option for improving conditioning. Check numerical stability for the chosen data, degree and bandwidths. `proper = TRUE` enforces probability-shape constraints; it does not repair an ill-conditioned polynomial solve.

With `regtype = "lp"`, `gradients = TRUE` and `gradient.order` greater than one expose higher-order derivative estimates with respect to continuous explanatory variables. These derivatives use the same local-polynomial basis, degree, Bernstein option, bandwidths, and kernels as the fitted conditional distribution. See `npreg` for the complete heterogeneous GLP multi-index and Bernstein-representation definitions shared by these routes. With `se = TRUE`, available higher-order derivative columns for fixed and adaptive nearest-neighbour bandwidths also have standard errors. These combine the existing local pseudo-response variance with the squared norm of the actual derivative weighting row, including its accepted ridge adjustment. They condition on the selected bandwidths, neighbour radii and local-polynomial map; they do not account for their selection uncertainty. Generalized nearest-neighbour higher-order point derivatives remain subject to the existing hat-operator restriction. Bootstrap inference remains recommended.

With positive-degree local-polynomial fits (including local-linear fits), numerical ridging follows the policies described in `npreg` (Seifert and Gasser, 2000). The conditional pseudo-response uncertainty formulas here are distinct from ordinary regression's residual normalization.

With positive-degree local-polynomial fits (including local-linear fits), categorical first-difference standard errors use paired endpoint influence calculations. The covariance between the two category endpoints is retained, not replaced by the sum of their marginal variances. These calculations condition on the selected basis, smoothing parameters and realized neighbour radii for all three bandwidth types. They run only when both `gradients = TRUE` and `se = TRUE` request them. A non-smooth regularization boundary or nonfinite influence gives `NA` uncertainty with a notice, without changing the point contrast.

For ordinary scalar fits, leading interior standard errors use the squared X-kernel moment; conditional density errors additionally use the independent Y-kernel moment. Analytic first-derivative errors for fixed and generalized nearest-neighbour bandwidths with Gaussian or Epanechnikov kernels use the squared derivative-kernel moment and the X bandwidth. These formulas do not establish uniform-kernel derivative inference or full adaptive-NN inference with estimated random radii. Positive-degree local-polynomial fits use their separate covariance calculation.

For fixed and generalized nearest-neighbour scalar fits with categorical variables, these leading moments retain squared categorical kernel factors and the joint training-category frequencies. Conditional distributions, and conditional densities with no continuous response, retain the numerator–denominator covariance of the ratio estimator. In particular, a constant response-kernel contribution has zero conditional variance. Purely categorical fits use this ratio covariance without a continuous bandwidth factor, including nearest-neighbour-labelled bandwidth objects. Beta-kernel and positive-degree local-polynomial covariance calculations retain their separate targets.

For scalar adaptive nearest-neighbour fits with any continuous variable, requested level and available continuous first-derivative standard errors use the complete empirical influence of the conditional ratio. They retain the actual normalized X weights and continuous/categorical Y contributions, including numerator and denominator derivatives. This working covariance holds the selected model, observed X geometry and realized response nearest-neighbour radii fixed; it does not account for their sampling or selection uncertainty. Local-constant-equivalent degree-zero fits share this calculation, including continuous responses with categorical-only X. Positive-degree local-polynomial, beta-kernel and paired categorical- contrast errors retain their separate covariance calculations. With `se = FALSE`, the additional uncertainty pass is not performed.

Value

The `se` component records whether uncertainty was requested. `conderr` and `congerr` are `NULL` when `se = FALSE`. `se()` and `gradients(..., se = TRUE)` only extract stored results and give a no-search refit message when the errors were omitted. `predict(..., se.fit = TRUE)` explicitly requests the needed errors.

`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. An evaluation row whose required explanatory normalization is exactly zero or non-finite is returned as `NA`; other rows remain available.
`conderr`	when `se = TRUE`, standard errors of the cumulative conditional distribution estimates. Undefined normalization rows are returned as `NA`.
`congrad`	if invoked with `gradients = TRUE`, estimates of the gradients at the evaluation points. For local-polynomial fits, continuous-coordinate derivative orders are

controlled by `gradient.order`. Undefined normalization rows are returned as `NA`.

`congerr` if invoked with `gradients = TRUE` and `se = TRUE`, standard errors of the gradients at the evaluation points, including available higher-order continuous derivatives under the working-covariance assumptions described above. Undefined normalization or uncertainty rows are returned as `NA`.

`log_likelihood` log likelihood of the cumulative conditional distribution estimate

Book And Method Pointers

The conditional distribution target is $F(y \mid x) = \Pr(Y \leq y \mid X = x)$. The estimator uses the selected mixed-data conditional distribution bandwidths and kernels for the response and conditioning coordinates; local-polynomial routes use the selected continuous-coordinate polynomial degree. These fitted conditional CDFs are also the object inverted by `npqreg` when computing conditional quantiles.

Scalar conditional fits involving beta kernels use the empirical observation-influence covariance. For normalized explanatory weights a_i, response-kernel contributions z_i, and fitted value m, the level variance is $\{n/(n-1)\} \sum_i [a_i(z_i - m)]^2$, using the full training count n. Supported continuous-derivative errors use the corresponding differentiated influence. Categorical contrasts use the difference of the two endpoints' observation influences inside the same squared norm, thereby retaining their covariance; self-contrasts have zero variance. These calculations hold the selected bandwidth state fixed; they do not add bandwidth-selection or random-NN-radius uncertainty, and do not replace positive-degree LP covariance.

For book-length derivations, see Li and Racine (2007), Chapter 6 *Conditional CDF and Quantile Estimation*, especially Sections 6.1, 6.2, and 6.5, together with Chapter 5 *Conditional Density Estimation*. The later workflow treatment is Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

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.

Chen, S.X. (1999), "Beta kernel estimators for density functions," Computational Statistics & Data Analysis, 31(2), 131–145. doi:10.1016/S0167-9473(99)00010-9.

Igarashi, G. and Kakizawa, Y. (2020), "Higher-order bias corrections for asymmetric kernel density estimators," Journal of Nonparametric Statistics, 32(3), 716–752. doi:10.1080/10485252.2020.1770754.

Mansouri, B., Rastin, A., and Allah Mombeni, H. (2024), "Beta Kernel Estimator for a Cumulative Distribution Function with Bounded Support," Journal of Sciences, Islamic Republic of Iran, 34(4), 349–361. doi:10.22059/jsciences.2024.367933.1007836.

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

np.kernels, np.options, plot, plot.np 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(Italy, {

# First, compute the bandwidths.

bw <- npcdistbw(formula=gdp~ordered(year), nmulti=1)
```

```
# 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).

if (interactive()) plot(bw)

})

# 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(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, nmulti=1)

# 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).

if (interactive()) plot(bw)
```

```
})

# 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")
with(faithful, {

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

bw <- npcdistbw(formula=eruptions~waiting, nmulti=1)

summary(bw)

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

if (interactive()) plot(bw)

})

# 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")
with(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, nmulti=1)

summary(bw)

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

if (interactive()) plot(bw)
```

```
})

# EXAMPLE 3: Variations on local polynomial conditional distribution
# estimation with proper = TRUE.

data("Italy")

# Work in years relative to 1975 to condition the raw polynomial basis.
# Calendar year = model year + year.offset, also for prediction data.
year.offset <- 1975
Italy2 <- within(Italy, {
  year <- as.numeric(as.character(year)) - year.offset
})

# Plot only: make the plotted surface proper on the plot evaluation grid.

Fhat <- npcdist(gdp ~ year, data = Italy2,
                regtype = "lp", degree = 3, nmulti = 1)

plot(Fhat, proper = TRUE)

# Fit an object whose fitted values are themselves proper.

ctrl_fit <- list(
  mode = "slice",
  apply = "fitted",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

Fhat_fit <- npcdist(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_fit
)

fit_proper <- fitted(Fhat_fit)
fit_raw <- Fhat_fit$condist.raw

# Predict on a common explicit y-grid for several years, and render
# those predictions proper.

g.grid <- seq(min(Italy2$gdp), max(Italy2$gdp), length.out = 200)

nd_grid <- expand.grid(
  gdp = g.grid,
  year = c(1955, 1975, 1995) - year.offset
)
```

```
pred_grid <- predict(Fhat, newdata = nd_grid, proper = TRUE)

# Predict on paired rows with different gdp grids by year, and still
# make the predictions proper via slice mode.

g1 <- seq(quantile(Italy2$gdp, 0.10),
          quantile(Italy2$gdp, 0.60), length.out = 60)
g2 <- seq(quantile(Italy2$gdp, 0.30),
          quantile(Italy2$gdp, 0.90), length.out = 35)

nd_slice <- rbind(
  data.frame(gdp = g1, year = rep(1960 - year.offset, length(g1))),
  data.frame(gdp = g2, year = rep(1985 - year.offset, length(g2)))
)

pred_slice <- predict(
  Fhat,
  newdata = nd_slice,
  proper = TRUE,
  proper.control = list(mode = "slice")
)

# One object that carries properization for fitted values and for later
# predict() calls.

ctrl_both <- list(
  mode = "slice",
  apply = "both",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

Fhat_both <- npcdist(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_both
)

fit_both <- fitted(Fhat_both)
pred_both <- predict(
  Fhat_both,
  newdata = nd_slice,
  proper.control = ctrl_both
)


## 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 'condbandwidth'
npcdistbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          gydat = NULL,
          bws,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          do.full.integral = FALSE,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
          scale.factor.init.lower = 0.1,
          lbd.dir = 0.1,
```

```
            lbd.init = 0.1,
            memfac = 500.0,
            ngrid = 100,
            nmulti,
            penalty.multiplier = 10,
            powell.remin = TRUE,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample = FALSE,
            scale.factor.search.lower = NULL,
            small = 1.490116e-05,
            tol = 1.490116e-04,
            transform.bounds = FALSE,
            ...,
            nomad.opts = list())

## Default S3 method:
npcdistbw(xdat = stop("data 'xdat' missing"),
            ydat = stop("data 'ydat' missing"),
            gydat,
            bws,
            bandwidth.compute = TRUE,
            bwmethod,
            bwscaling,
            bwtype,
            cfac.dir,
            scale.factor.init,
            cxkerbound,
            cxkerlb,
            cxkerorder,
            cxkertype,
            cxkerub,
            cykerbound,
            cykerlb,
            cykerorder,
            cykertype,
            cykerub,
            dfac.dir,
            dfac.init,
            dfc.dir,
            do.full.integral,
            ftol,
            scale.factor.init.upper,
            hbd.dir,
            hbd.init,
            initc.dir,
            initd.dir,
            invalid.penalty,
            itmax,
```

```
            lbc.dir,
            scale.factor.init.lower,
            lbd.dir,
            lbd.init,
            memfac,
            ngrid,
            nmulti,
            oxkertype,
            oykertype,
            penalty.multiplier,
            nomad.remin = FALSE,
            powell.remin,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample,
            scale.factor.search.lower = NULL,
            small,
            tol,
            transform.bounds,
            uxkertype,
            regtype = c("lc", "ll", "lp"),
            basis = c("glp", "additive", "tensor"),
            degree = NULL,
            degree.select = c("manual", "coordinate", "exhaustive"),
            search.engine = c("nomad+powell", "cell", "nomad"),
            nomad = FALSE,
            nomad.nmulti = 0L,
            degree.min = NULL,
            degree.max = NULL,
            degree.start = NULL,
            degree.restarts = 0L,
            degree.max.cycles = 20L,
            degree.verify = FALSE,
            bernstein.basis = FALSE,
            ...,
            nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, optional distribution grid, and whether bandwidths are supplied or computed.

`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`.

`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.

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.

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

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

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).

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).

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.

subset — an optional vector specifying a subset of observations to be used in the fitting process.

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.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search when regtype="lp".

degree.max — optional scalar or integer vector giving upper bounds for automatic degree search over continuous xdat predictors when degree.select != "manual".

degree.max.cycles — positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for "manual" and "exhaustive" degree selection.

degree.min — optional scalar or integer vector giving lower bounds for automatic degree search over continuous xdat predictors when degree.select != "manual".

degree.restarts
: non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for "manual" and "exhaustive" degree selection.

degree.select
: character string controlling local-polynomial degree handling when regtype="lp". "manual" (default) treats degree as fixed. "coordinate" performs cached coordinate-wise search over admissible degree vectors for the continuous xdat predictors. "exhaustive" evaluates the full admissible degree grid when search.engine="cell". For NOMAD-based search engines, any non-"manual" value requests direct joint search over degree and bandwidth coordinates.

degree.start
: optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous xdat predictors.

degree.verify
: logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for search.engine="cell".

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

bwmethod
: which method to use to select bandwidths. cv.ls specifies least-squares cross-validation (Li, Lin and Racine (2013)). normal-reference computes an ad hoc Silverman-style rule-of-thumb intended for fast exploratory data analysis, not production bandwidth selection. For raw continuous bandwidths it uses $1.06\sigma_j n^{-1/(2P+l)}$ for conditioning variables and $1.587\sigma_j n^{-1/(2P+l)}$ for response variables, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, normal-consistent median absolute deviation, interquartile range divided by $2\Phi^{-1}(0.75)$), 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, normal-consistent median absolute deviation, interquartile range divided by $2\Phi^{-1}(0.75)$), 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

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

dfac.dir stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

dfac.init non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

hbd.dir upper bound for direction set search for Powell's algorithm for categorical variables. See Details

hbd.init upper bound for scale factors for categorical variables for Powell's algorithm. See Details

initd.dir initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbd.dir lower bound for direction set search for Powell's algorithm for categorical variables. See Details

lbd.init lower bound 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

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

cfac.dir stretch factor for direction set search for Powell's algorithm for numeric variables. See Details

dfc.dir chi-square degrees of freedom for direction set search for Powell's algorithm for numeric variables. See Details

initc.dir initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbc.dir lower bound for direction set search for Powell's algorithm for numeric variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

cxkerbound character string controlling continuous-kernel support handling for xdat. Can be set as none (default kernel on full support), range (use sample min/max, except that beta kernels use the outer half-spacing rule in np.kernels), or fixed (use cxkerlb/cxkerub). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see np.kernels.

cxkerlb numeric scalar/vector of lower bounds for continuous xdat variables used when cxkerbound="fixed". Must satisfy lower-bound validity for each variable (e.g., <= min(variable)). Use -Inf for unbounded below. See np.kernels for bounded-kernel normalization details.

cxkerub numeric scalar/vector of upper bounds for continuous xdat variables used when cxkerbound="fixed". Must satisfy upper-bound validity for each variable (e.g., >= max(variable)). Use Inf for unbounded above. See np.kernels for bounded-kernel normalization details.

cykerbound character string controlling continuous-kernel support handling for ydat. Can be set as none (default kernel on full support), range (use sample min/max, except that beta kernels use the outer half-spacing rule in np.kernels), or fixed (use cykerlb/cykerub). For the response-side distribution operator, the selected continuous kernel is integrated after observation-centered truncation and normalization on the chosen support; see np.kernels.

cykerlb numeric scalar/vector of lower bounds for continuous ydat variables used when cykerbound="fixed". Must satisfy lower-bound validity for each variable (e.g., <= min(variable)). Use -Inf for unbounded below. See np.kernels for bounded-kernel normalization details.

cykerub numeric scalar/vector of upper bounds for continuous ydat variables used when cykerbound="fixed". Must satisfy upper-bound validity for each variable (e.g., >= max(variable)). Use Inf for unbounded above. See np.kernels for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

scale.factor.init
: deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to 0.5. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is max(scale.factor.init, scale.factor.search.lower). See Details.

scale.factor.init.lower
: lower endpoint for random continuous scale-factor starts. Defaults to 0.1. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). See Details.

scale.factor.init.upper
: upper endpoint for random continuous scale-factor starts. Defaults to 2.0. It must be greater than or equal to the effective lower endpoint, max(scale.factor.init.lower, scale.factor.search.lower); otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Distribution Integral And Grid Controls: These controls tune the conditional distribution-function integral and grid calculations.

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

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.

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

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels for xdat and ydat.

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.

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

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

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

oykertype character string used to specify the ordered categorical kernel type for ydat. Can be set as wangvanryzin, liracine, or racineliyan. Defaults to liracine.

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

Local-Polynomial Model Specification: These arguments control the local-polynomial estimator, basis, and fixed degree specification.

basis character string specifying the polynomial basis used when regtype="lp". Options are "glp", "additive", and "tensor".

bernstein.basis logical value controlling the stabilized representation for regtype="lp". For basis="glp", TRUE selects the complete degree-graded, orthonormal shifted-Legendre definition given in npreg; additive and tensor bases retain their fixed-coordinate-degree representation. When automatic degree search is requested and bernstein.basis is not explicitly supplied, the search route defaults to TRUE for numerical stability. Explicit bernstein.basis=FALSE is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

degree integer scalar or integer vector of polynomial degrees for continuous xdat variables when regtype="lp". If scalar, the value is recycled to all continuous xdat variables. With no continuous xdat predictors, regtype="lp" is only admissible when degree=0, the local-constant equivalent.

regtype character string specifying the conditional local method used for the xdat regression weight operator. Options are "lc", "ll", and "lp". For npc* methods, "ll" is implemented via the canonical local polynomial engine with degree = 1 and basis = "glp". "ll" and positive-degree "lp" require at least one continuous xdat predictor; for categorical-only xdat, use "lc" or "lp" with degree=0. If local-linear cv.ls search fails while using this canonical raw basis, retrying explicitly with regtype="lp", degree=1, and bernstein.basis=TRUE, or centering/scaling the continuous regressors, can improve numerical conditioning without changing package defaults or invoking an automatic fallback.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

nomad logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When TRUE, any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype are filled with regtype="lp", search.engine="nomad+powell", degree.select="coordinate", bernstein.basis=TRUE, degree.min=0L, degree.max=10L, degree.verify=FALSE, and bwtype="fixed". Explicit incompatible settings error immediately; in particular, explicit nomad=TRUE currently requires regtype="lp", bwtype="fixed", automatic degree search, no explicit degree, at least one continuous xdat predictor, and search.engine %in% c("nomad", "nomad+powell"). This shortcut does not change the meaning of nmulti or nomad.nmulti: nmulti remains the outer restart count, while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in bw$nomad.shortcut for a returned object bw; when available, nomad.time and powell.time record the direct-search and Powell-polish timing components.

nomad.nmulti non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when regtype="lp" and automatic degree search uses search.engine="nomad" or "nomad+powell". Defaults to 0L, which preserves the current one-start-per- restart behavior. This does not replace nmulti: nmulti controls outer restarts, while nomad.nmulti controls inner NOMAD multistarts within each outer restart.

nomad.opts optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. NULL is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as NB_THREADS_PARALLEL_EVAL > 1.

nomad.remin logical flag controlling the optional second NOMAD hot start. When TRUE, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to FALSE; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

search.engine character string controlling the automatic local-polynomial search backend when regtype="lp" and degree.select != "manual". "nomad+powell" (default)

performs direct joint search over the xdat-side continuous bandwidth coordinates and degree vector using the native **crs** NOMAD C API, then applies one Powell hot start from the NOMAD solution. "nomad" omits the Powell refinement. "cell" profiles the criterion over the admissible degree grid using repeated fixed-degree bandwidth solves. NOMAD-based search supports bwtype="fixed", bwtype="generalized_nn", and bwtype="adaptive_nn"; it currently requires degree.verify=FALSE and the suggested package **crs** to be installed.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, memory blocking, and bounded search transformations.

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)).

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".

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

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

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

powell.remin
: logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, TRUE restarts the local search from the located minimum. For search.engine="nomad+powell", this controls only the final Powell bandwidth-polish step. The default is TRUE for ordinary Powell routes and FALSE for the Powell polish after NOMAD unless explicitly supplied.

bwsolver
: bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Use search.engine rather than bwsolver for automatic degree search. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

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)).

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)).

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.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

... additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via `newdata` does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state.

Beta associated kernels may be selected independently with `cxkertype="beta"` and `cykertype="beta"`. Orders 2, 4, 6, and 8, fixed/generalized-NN/adaptive-NN bandwidths, `cxkerbound`/`cykerbound` equal to `"fixed"` or `"range"`, explicit bandwidth objects, and automatic `cv.ls` search are supported. Normal-reference selection is not. Beta kernels act on continuous coordinates alongside the kernels for categorical coordinates. Local-constant, local-linear and local-polynomial fits are supported. Local-constant gradients with respect to X are available: a beta X coordinate uses its analytic evaluation-coordinate PDF derivative, while the beta or legacy Y CDF factor is held fixed. Exact matching beta-X endpoints use structural cancellation in the normalized ratio and can yield signed infinity or `NA` when a finite derivative does not exist. Order-2 CDF factors use the observation-centered incomplete-beta construction, while orders 4, 6, and 8 use its signed multiscale extension. See `np.kernels` for the orientation, warnings, and references.

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)`, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

The bandwidth-selection argument surface is easiest to read by decision group. Start with the data and bandwidth inputs (`xdat`, `ydat`, `gydat`, `bws`, and `bandwidth.compute`), then choose the

bandwidth criterion and representation (bwmethod, bwscaling, and bwtype). Next choose continuous kernel and support controls (cxker* and cyker*), categorical kernel controls (uxkertype, oxkertype, and oykertype), and numerical search controls including nmulti, tolerances, penalties, and the scale.factor.* search-start and admissibility controls. Local-polynomial and NOMAD controls (regtype, basis, degree*, search.engine, nomad, nomad.nmulti, and bernstein.basis) are relevant when using the explicit local-polynomial route.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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. For fixed local-constant/local-linear fits, and for local-polynomial fits with degree.select="manual", bandwidth search uses multidimensional Powell direction-set optimization.

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.

Exact ties can make the literal kth continuous nearest-neighbor radius zero. Such a bandwidth is undefined and is rejected; see np.kernels for the precise contract and remedies.

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.

Response expressions such as log(y) or I(y^2) are not supported on the left-hand side and are rejected. Instead create, for example, data$log_y <- log(data$y) and use log_y ~ x, or supply the transformed response through ydat. Right-hand-side offset() special terms are also unsupported and are rejected rather than omitted. These checks do not evaluate response expressions, alter stored/dynamically constructed formulas, or change supported explanatory transformations.

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.

When regtype="lp" and degree.select != "manual", npcdistbw can jointly determine the xdat-side local polynomial degree vector and the fixed bandwidth coordinates entering the conditional distribution criterion. With search.engine="cell", the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search. With search.engine="nomad" or "nomad+powell", the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API; "nomad+powell" then performs one Powell hot start from the NOMAD solution and keeps the better of the direct NOMAD and polished answers. This polynomial-adaptive joint-search route is motivated by Hall and Racine (2015) together with Li, Li, and Racine (under revision). When bernstein.basis is not explicitly supplied, the automatic search route defaults to bernstein.basis=TRUE for numerical stability.

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For conditional distribution bandwidth selection it expands any missing values to the equivalent long-form call

```
npcdistbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value nomad="auto" applies the same LP shortcut but leaves search.engine and degree.select eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit search.engine or degree.select choices are honored.

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.

Prediction follows a two-stage contract. Calling predict(bw) on the returned bandwidth object is a convenience forwarder for npcdist(bws = bw) and returns a fitted condistribution object; it does not return a numeric vector of conditional-distribution estimates. Call fitted(fit) or predict(fit, newdata = ...) on that fitted object to obtain numeric results. The canonical workflow is

```
bw <- npcdistbw(y ~ x, data = mydata)
fit <- predict(bw)             # equivalent to npcdist(bws = bw)
fitted(fit)
predict(fit, newdata = newdata)
```

The functions summary and plot also support condbandwidth objects.

Book And Method Pointers

Local-polynomial search uses the numerical-ridging policies described in npreg; see Seifert and Gasser (2000) there.

npcdistbw selects response and conditioning bandwidths, and when requested local-polynomial degree/search metadata, for the conditional distribution target $F(y \mid x) = \Pr(Y \leq y \mid X = x)$ estimated by npcdist and inverted by npqreg.

For book-length derivations, see Li and Racine (2007), Chapter 6 *Conditional CDF and Quantile Estimation*, especially Sections 6.1, 6.2, 6.3, and 6.5, and Chapter 5 *Conditional Density Estimation*. The later workflow treatment is Racine (2019), Chapter 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: 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(2, 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.

Hall, P. and J.S. Racine (2015), "Infinite Order Cross-Validated Local Polynomial Regression," Journal of Econometrics, 185, 510-525.

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.

Li, A. and Q. Li and J.S. Racine (under revision), "Boundary Adjusted, Polynomial Adaptive, Nonparametric Kernel Conditional Density Estimation," Econometric Reviews.

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

np.kernels, np.options, plot, plot.np 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(Italy, {

bw <- npcdistbw(formula=gdp~ordered(year), nmulti=1)

# 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.

})

# 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")
Italy <- Italy[seq_len(min(300, nrow(Italy))), ]
with(Italy, {

bw <- npcdistbw(xdat=ordered(year), ydat=gdp, nmulti=1)

# 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.

})

## End(Not run)
```

`npcdisthat` *Conditional Distribution Hat Operator*

Description

Constructs the conditional distribution hat operator associated with `npcdist` bandwidth objects. The returned operator maps a right-hand side `y` to Hy; with `y = 1` this reproduces the fitted conditional distribution function.

Usage

```
npcdisthat(bws,
           txdat = stop("training data 'txdat' missing"),
           tydat = stop("training data 'tydat' missing"),
           exdat,
           eydat,
           y = NULL,
           output = c("matrix", "apply"),
           deriv = NULL,
           s = NULL)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted conditional distribution bandwidth object of class "condbandwidth".
exdat	Optional evaluation conditioning data. If omitted, the operator is built on the training conditioning data.
eydat	Optional evaluation response data. If omitted, the operator is built on the training response data.
txdat	Training conditioning data used to construct the operator.
tydat	Training response data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

output	Either "matrix" for the hat matrix or "apply" for direct application to y.
y	Optional right-hand side vector or matrix with one row per training observation.

Derivative Controls: These arguments request continuous-conditioning derivative hat operators.

deriv	Optional derivative selector passed through to the continuous-conditioning derivative resolver.
s	Optional derivative multi-index for continuous conditioning variables. The default s = NULL returns the ordinary conditional-distribution hat operator.

Details

For the adaptive-nearest-neighbour local-constant (including degree-zero local-polynomial) ratio, every finite nonzero conditioning-weight sum is used without an epsilon floor. If all computed conditioning weights at an external evaluation row are zero, that row is returned as NA with a warning. This describes computed weights, not necessarily absence of geometric support. Zero response weights still give a valid zero numerator when the conditioning denominator is defined. Required internal and bootstrap ratios remain strict.

For output = "matrix", the return value is a matrix with class c("npcdisthat", "matrix") and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For output = "apply", the function returns H y directly. Matrix right-hand sides are applied column-wise.

The optional s argument follows the derivative multi-index convention used by npreghat for continuous conditioning variables. The default s = NULL returns the ordinary conditional-distribution hat operator. Supplying one first derivative, for example s = 1L in a univariate continuous-x problem or s = c(x2 = 1L) in a multivariate problem, returns the hat operator for that component of the conditional-distribution gradient. This derivative route is intended to match npcdist(..., gradients = TRUE)$congrad.

For local-constant beta X kernels, first derivatives use the analytic beta PDF derivative; the beta Y CDF factor is not differentiated. Exact support-endpoint rows whose raw X weights jump are returned as NA, because a response-free hat matrix cannot perform the Y-factor-specific ratio cancellation available to the fitted conditional-distribution route.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class "npcdisthat" or the applied result H y, depending on output.

Examples

```
## Not run:
data(cps71)
tx <- data.frame(age = cps71$age)
ty <- data.frame(logwage = cps71$logwage)

bw <- npcdistbw(xdat = tx, ydat = ty, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = c(1.0, 1.0))

H <- npcdisthat(bws = bw, txdat = tx, tydat = ty)
dist.hat <- npcdisthat(bws = bw, txdat = tx, tydat = ty,
                       y = rep(1, nrow(tx)),
                       output = "apply")
dist.core <- fitted(npcdist(bws = bw, txdat = tx, tydat = ty))

head(cbind(dist.core, dist.hat), n = 2L)

## 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"),
          B = 399,
          pivot = TRUE,
          density.weighted = TRUE,
          random.seed = 42,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the model formula/data interface and explicit data inputs.

`data` This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

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.

`formula` a symbolic description of the model to be tested. If `xdat` and `ydat` are omitted, the data are extracted from this formula and `data`.

`model` a model object obtained from a call to `lm` (or `glm`). Important: the model must retain its design and response components, normally requested with `x=TRUE` and `y=TRUE`. The values, not the spelling of the original call, determine availability. Also, the test is based on residual bootstrapping hence the outcome must be continuous (which rules out Logit, Probit, and Count models).

`subset` an optional vector specifying a subset of observations to be used.

`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`.

Bootstrap And Test Controls: These arguments control the test statistic, bootstrap procedure, and reproducibility settings.

`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`.

B — an integer value specifying the number of bootstrap replications to use. Defaults to 399.

density.weighted — a logical value specifying whether the statistic should be weighted by the density of xdat. Defaults to TRUE.

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.

pivot — a logical value specifying whether the statistic should be normalised such that it approaches $N(0,1)$ in distribution. 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: Further arguments are passed to the bandwidth-selection routines used by the test.

... — 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.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

For distribution = "bootstrap", each bootstrap model is still refitted separately using the selected bootstrap law. The resulting residual vectors are collected in bounded groups, and the fixed-design kernel contractions for I_n and, when requested, $\hat{\Omega}$ are evaluated as multi-response kernel sums. This reuses kernel geometry without forming a dense observation-by-observation weight matrix or changing residual order, pivoting, density weighting, or random draws. The selected bandwidth's effective kernel metadata are shared by the observed and bootstrap statistics. Manual bandwidth inputs and selection controls are not forwarded a second time as kernel-sum arguments.

Bootstrap refits preserve the model's observation weights and offsets (GLM prior weights, not iterative working weights).

Model residuals, fitted values, response and design use the model's compact complete-observation sample, including when it was fitted with na.exclude. The supplied test data must represent those same complete observations in the same order. Row counts and response values are checked; when those agree, the caller remains responsible for supplying the matching covariate rows. The test does not infer a new pairing or reorder observations. The original model and its missing-value restoration policy are not modified.

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.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

npcmstest tests a parametric conditional-mean specification against a nonparametric alternative, using residual-based bootstrap or asymptotic calibration as selected by distribution. The statistic is designed for continuous outcomes and fitted lm or glm objects whose model frame, model matrix, and response are available.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests*, especially Section 12.1 and Section 12.1.2. For the conditional-mean estimation context used by the test, see Racine (2019), Chapter 6 *Conditional Mean Function Estimation*.

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.

See Also

np.kernels, np.options, plot, npregbw.

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")
with(cps71, {

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

if (interactive()) 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, B=9, nmulti = 1)

# 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)

if (interactive()) 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, B=9, nmulti = 1)

# 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.

})

# 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")
with(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, B=9, nmulti = 1)

})

## 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,
          ...,
          se = FALSE)

## Default S3 method:
npconmode(bws,
          txdat,
          tydat,
          nomad = FALSE,
          proper = NULL,
          proper.control = list(),
          probabilities = FALSE,
          gradients = FALSE,
          level = NULL,
          ...,
          se = FALSE)

## S3 method for class 'conbandwidth'
npconmode(bws,
          txdat = stop("invoked without training data 'txdat'"),
          tydat = stop("invoked without training data 'tydat'"),
          exdat,
          eydat,
          proper = NULL,
          proper.control = list(),
          probabilities = FALSE,
          gradients = FALSE,
          level = NULL,
          ...,
          se = FALSE)

## S3 method for class 'conmode'
predict(object,
          newdata = NULL,
          type = c("class", "prob"),
          se.fit = FALSE,
          ...)

## S3 method for class 'conmode'
plot(x, ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

bws	a bandwidth specification. This can be set as a conbandwidth object returned from an invocation of npcdensbw
data	This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces. 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.
txdat	a p-variate data frame of explanatory data (conditioning data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.
tydat	a one (1) dimensional vector of unordered or ordered factors, containing the dependent data. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.
object	an object of class "conmode" returned by npconmode.
x	an object of class "conmode" returned by npconmode.

Evaluation Data: These arguments control where the conditional mode is evaluated.

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	Optional observed categorical outcomes at exdat, supplied as a factor or a one-column data frame. Used only for evaluation diagnostics. When exdat is supplied without outcomes, classification diagnostics are NA; training outcomes are not substituted.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
type	prediction type for predict.conmode. Use "class" to return modal class predictions and "prob" to return the full class-probability matrix. Probability prediction requires either a stored probability matrix or an evaluation path with probabilities=TRUE.
se.fit	a logical value for predict.conmode. If TRUE and type="prob", return a list with components fit and se.fit, containing the class-probability matrix and matching asymptotic standard-error matrix. Standard errors are for the smooth class-probability target, not for the discrete modal class label.
se	a scalar logical controlling raw class-probability asymptotic standard errors. Defaults to FALSE; class selection, fitted probabilities and requested probability gradients are unchanged. With TRUE, selected-probability errors are stored,

together with the full error matrix when probabilities are retained. This does not add standard errors for the modal class label or probability gradients, or change the existing unavailable-error treatment after projection. To add uncertainty without repeating bandwidth search, use `npconmode(bws=fit$bws, probabilities=TRUE, se=TRUE)`, supplying the original training/evaluation data if needed. Prediction with new evaluation rows and `se.fit=TRUE` requests this work explicitly; prediction without evaluation rows only extracts stored results.

Probability And Gradient Controls: These arguments control stored class probabilities, proper probability normalization, and level-specific class-probability effects.

`probabilities` a logical value. If `TRUE`, store the full matrix of fitted probabilities over the discrete response support in the returned object. This support is retained from the fitted training data. Response values supplied for evaluation diagnostics, including unused or unseen factor levels, do not enlarge or reorder the predicted class support. The default `FALSE` keeps returned objects compact. If `gradients=TRUE`, probabilities are stored automatically because they are the object-fed target for `plot.conmode`.

`gradients` a logical value. If `TRUE`, compute and store class-probability gradients/effects for one response level with respect to the conditioning variables. These are derivatives for continuous conditioning variables and the corresponding smooth finite-change effects for discrete conditioning variables as produced by the underlying `npcdens` gradient route. The target is the smooth class probability $\Pr(Y = \ell \mid X = x)$, not the derivative of the discrete modal class label. With a heterogeneous local-polynomial conditional-density degree vector, a degree-zero continuous coordinate has no available first-derivative estimate and is stored as `NA`; the established all-degree-zero compatibility route is unchanged. Its plot panel remains present but empty when another derivative or categorical first-difference effect is available; a request with no available component is rejected.

`level` response level for class-probability gradients/effects and `plot.conmode` class-probability displays. If omitted, the default is the base/reference response level, `levels(y)[1]`, for both factor and ordered responses. Users should supply `level` explicitly when a particular binary or multinomial outcome probability is of substantive interest.

`proper` a logical value or `NULL`. If `TRUE`, fitted probabilities over the discrete response support are projected onto the probability simplex before modal selection. If `FALSE`, legacy raw per-level values are used. If `NULL`, the default is estimator-aware: `FALSE` for `regtype="lc"` and `TRUE` otherwise.

`proper.control` a list of controls for discrete probability properization. Currently `tol` may be supplied to set the numerical tolerance for non-negativity and unit-mass checks.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when conditional-density bandwidths are computed inside `npconmode`.

`nomad` logical shortcut passed through to `npcdensbw` when bandwidths are computed inside `npconmode`. When `TRUE`, the conditional-density bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`,

degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npcdensbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Automatic NOMAD degree search requires at least one continuous explanatory variable. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart, prediction/evaluation route, or plot route as appropriate.

... additional arguments supplied to npcdensbw when bandwidths are computed internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth-selection controls such as bwmethod, bwtype, and bwscaling, kernel/support controls such as cxkertype, cykertype, cxkerorder, cykerorder, cxkerbound, and cykerbound, categorical kernel controls such as uxkertype, uykertype, oxkertype, and oykertype, search controls such as nmulti and scale.factor.search.lower, and local-polynomial/NOMAD controls such as regtype, degree, bernstein.basis, degree.select, and nomad.nmulti are supplied. In predict.conmode, additional arguments are passed to npconmode for evaluation with the stored bandwidth object; native exdat and eydat take precedence over newdata. In plot.conmode, additional arguments control the object-fed probability/effect display; common controls include gradients, level, output, data.rug, layout, legend, view, neval, grid.control, perspective, plot.vars, renderer, and graphics arguments. The default view="sample" draws the stored fitted probabilities/effects at the fitted evaluation points. The view="fixed" route constructs one-dimensional object-fed slices over each conditioning variable, holding the remaining variables at their uocquantile median/mode values; np_grid_control(xtrim=..., xq=...) may be used to adjust the slice range and holdout quantiles. With perspective=TRUE, the same fixed-grid route draws a base-graphics probability surface for one selected response level over two continuous conditioning variables; use plot.vars to name the displayed pair when more than two continuous conditioning variables are present; renderer="rgl" requests the corresponding interactive surface when **rgl** is available. Both renderers use the package-wide viridis surface palette and neutral-gray three-face reference grid. The public output="both" spelling is accepted as an alias for the historical output="plot-data" behavior. For class-probability plots, errors="asymptotic" adds probability-level asymptotic standard errors and intervals for the selected response level. These intervals target the smooth probability $\Pr(Y = \ell \mid X = x)$, not the discrete modal class label. Rows whose probabilities were materially repaired by simplex projection report NA standard errors rather than treating raw conditional-density standard errors as projected-probability standard errors. Bootstrap intervals are available for one-dimensional view="sample" and view="fixed" probability plots and use the stored bandwidth object without recomputing bandwidths. Class-probability bootstrap plots default to B=399; an explicit B overrides this count. Asymptotic joint bands are available for regular local-constant and local-polynomial probability and first-gradient/effect routes. A joint Gaus-

sian maximum is calibrated for one named response level on the declared grid within each panel. The complete response support enters the proper-probability Jacobian, including smooth positive normalization, clipping with a stable sign pattern, and projection onto a stable simplex face. Branch seams or unidentified zero-variance rows remain unavailable. A zero Jacobian caused by clipping or projection onto an inactive class or a singleton face does not certify a constant sampling target. Such a panel warns and returns unavailable simultaneous limits, retaining the plotted probabilities and any available pointwise and Bonferroni intervals. Proper probability bands use standard errors from that transformed influence; existing pointwise and Bonferroni probability bands retain their own convention in `band="all"`.

Continuous gradient plots retain the raw underlying conditional-density derivative documented above, including when `proper=TRUE`; the derivative is not silently changed to a normalized-probability derivative. Categorical effects retain the existing view/engine policy: fixed-view and positive-degree sample effects difference endpoint probabilities, while local-constant and degree-zero sample effects retain their native raw contrast. New asymptotic gradient intervals use the covariance and marginal scale of that particular target. Bootstrap gradient intervals remain unsupported.

These are selected-smoothing variability approximations for the named level, not confidence bands for the winning class, all classes jointly, smoothing bias or values between grid points. Independent observations, regular local neighborhoods and valid covariance/marginal approximations are required. Local-polynomial and fixed-view response factors use local response moments as a working covariance approximation conditional on the accepted design and response map. At fixed nonshrinking bandwidths they need not equal a full empirical-caseweight derivative, which also perturbs the design map. Their leading interpretation requires smooth shrinking neighborhoods and nondegenerate local response variation. A growing or data-dependent sample grid additionally needs uniform linearization, non-dominating influences and a Gaussian-maximum approximation; arbitrary sample grids do not carry a uniform-coverage guarantee. `gaussian.draws` (default 9999) and `random.seed` control reproducible joint calibration; pointwise and Bonferroni-only gradient requests need no Gaussian simulation. Surface asymptotic intervals are available as `output="data"` payload columns and are drawn with graphical surface output. The two conditioning variables selected by `plot.vars` must be distinct. Surface bootstrap intervals remain unsupported. See `npcdensbw` and `plot.np` for the complete bandwidth-selection and plotting argument surfaces.

Details

Conditional fitting and replay use the literal categorical kernels for both explanatory and response variables. Near-upper categorical bandwidths are not replaced by approximate constant factors. The same target is used for levels, gradients, standard errors, conditional hats, and reconstructed plot uncertainty. This conditional-family policy does not change direct regression or unconditional kernel-sum evaluation and does not modify the caller's `np.largelambda` option.

New fitted objects record `conditional.kernel.policy`. Stored values in older objects remain available without refitting. A plot that combines such historical stored class probabilities or effects

with newly reconstructed bootstrap or joint asymptotic uncertainty requires an explicit refit under the current policy. Stored pointwise probability standard errors remain readable. Explicit predictions at new evaluation data use the current conditional target and retain the object's probability-normalization controls.

With a formula-derived bandwidth object, supplying only `tydat` replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective `na.action` is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use `data` for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named `newdata` columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over `newdata`.

For both formula and native bandwidth objects, `newdata` supplies evaluation predictors and, when present, named evaluation outcomes. Explicit native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

With evaluation predictors and outcomes, the confusion matrix and accuracy describe the evaluation sample. With evaluation predictors only, predictions are returned normally and these diagnostics are `NA`. Without evaluation inputs, diagnostics describe training data. Missing evaluation outcomes do not remove predictor-complete prediction rows: scores use observed outcome/prediction pairs, and are `NA` when none are available. Supplying outcomes never changes fitted class support, probabilities or gradients.

Formula variables of class `ts`, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Local-polynomial calculations inherit numerical ridging from `npcdens`. See that estimator's help for the solve policy and its distinct conditional uncertainty formula.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

If `bws` is not an explicit `conbandwidth` object, `npconmode` computes conditional density bandwidths by forwarding the call to `npcdensbw`. The resulting `conbandwidth` object is stored unchanged in the returned object's `bws` component. Regression type, local-polynomial degree, and NOMAD/search metadata are also mirrored on the returned `conmode` object for convenient inspection and summary reporting; the bandwidth object remains the canonical source.

For non-local-constant conditional-mode fits, `npconmode` constructs the full set of fitted probabilities over the discrete response support and projects them onto the probability simplex before selecting and reporting the modal outcome. Thus the probabilities used for modal selection are non-negative and sum to one over the discrete support. For binary outcomes this is the complement contract $\Pr(Y = \ell_2 \mid X = x) = 1 - \Pr(Y = \ell_1 \mid X = x)$. Local-constant fits are already proper by construction, so omitted `proper` resolves to `FALSE` for `regtype="lc"` and to `TRUE` otherwise.

The `predict` method follows the usual S3 `newdata` convention. With no evaluation arguments, `predict(fit)` extracts the stored modal class and `predict(fit, type="prob")` extracts stored

class probabilities when the object was fitted with probabilities=TRUE. With newdata, predict evaluates the conditional mode at the supplied rows using the stored bandwidth object. Use type="prob" to return the full matrix of fitted class probabilities at the evaluation rows. Use se.fit=TRUE with type="prob" to return a list containing the class-probability matrix and the matching asymptotic standard-error matrix. Native evaluator arguments exdat and eydat remain available for advanced workflows and take precedence if supplied.

Value

npconmode returns a conmode object with the following components:

omit, rows.omit, nobs.omit
: omitted-row information for the active result: training rows for training evaluation, otherwise evaluation rows. Existing NA padding of row outputs is unchanged.

train.rows.omit, train.nobs.omit
: training omissions, retained separately even when the result evaluates external data.

eval.omit, eval.rows.omit, eval.nobs.omit
: omission information for external evaluation data, when supplied.

bws
: the conbandwidth object used to compute the conditional mode

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

conderr
: a vector of numeric type containing asymptotic standard errors for the modal density estimates at each evaluation point. This component is NULL when se=FALSE. If a row is materially properized by probability projection, this value is set to NA for that row rather than reporting a standard error for the unrepaired raw probability.

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

diagnostics.sample
: "training", "evaluation", or "unavailable", identifying the sample used for classification diagnostics.

diagnostics.nobs
: Number of observed outcome/prediction pairs scored.

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

`probabilities` if requested, a matrix of fitted probabilities over the discrete response support

`probability.levels`
: if `probabilities` are stored, the response support corresponding to the probability matrix columns

`probability.errors`
: if `probabilities` are stored, a matrix of asymptotic standard errors for the raw selected-level conditional probability estimates, with rows set to `NA` where simplex projection materially repaired the fitted probability vector; omitted when `se=FALSE`

`se` whether probability uncertainty was requested

`probability.repaired.rows`
: if `probabilities` are stored, a logical vector identifying rows whose fitted probability vector was materially repaired by proper probability projection

`probability.gradients`
: if `gradients=TRUE`, a matrix whose entries are class-probability gradients/effects for one selected response level, indexed by evaluation row and conditioning variable

`probability.gradient.level`, `probability.gradient.names`, `probability.gradient.info`
: metadata describing the response support, conditioning variables, and interpretation of `probability.gradients`

`proper.requested`, `proper.applied`, `proper.info`
: metadata describing whether discrete probability properization was requested, whether any row was materially projected, and diagnostics for non-negativity and unit-mass checks

`regtype`, `degree`, `nomad`, `search.engine`
: metadata mirrored from `bws` describing the conditional density regression type, local-polynomial degree, whether NOMAD automatic degree search was used, and the search engine when available

`degree.search`, `nomad.shortcut`, `nomad.time`, `powell.time`
: detailed search metadata mirrored from `bws` when automatic degree search or NOMAD/Powell refinement was used

The function `predict` may be used to extract conditional mode class predictions, while `fitted` extracts the conditional density estimates at the conditional mode from the resulting object. The function `gradients` extracts class-probability gradients/effects when `gradients=TRUE`. Also, `summary` and `plot` support `conmode` objects. For `plot.conmode`, first fit with `probabilities=TRUE` so the plot can remain object-fed. The default plot displays the fitted probability for the base/reference response level `levels(y)[1]`; use `plot(fit, level=...)` to select another outcome and `plot(fit, gradients=TRUE)` to display stored class-probability effects. The default `view="sample"` draws stored object-fed probability/effect payloads at the fitted evaluation points. Use `view="fixed"` and `neval` to draw object-fed one-dimensional slices over each conditioning variable, with other variables held at their median/mode values. Use `perspective=TRUE` to draw a base-graphics probability surface for one selected response level over two continuous conditioning variables, and `renderer="rgl"` for the corresponding interactive surface. Use `errors="asymptotic"` for probability-level standard errors and intervals in one-dimensional plots and fixed surfaces. Surface `band="all"` draws the pointwise, simultaneous and Bonferroni endpoints already present in

output="data"; coverage remains for the selected response level on the declared evaluation grid, not jointly across response levels. Automatic surface limits include finite interval endpoints without truncating them to $[0, 1]$. Explicit zlim controls drawing only and does not alter returned endpoints. Unavailable interval cells remain missing. Surface bootstrap intervals and gradient surfaces are not implemented.

Book And Method Pointers

The conditional-mode target is $\arg\max_y \Pr(Y = y \mid X = x)$ for a discrete response. In practice npconmode estimates the conditional probability of each response support point using npcdensbw / npcdens, optionally projects non-local-constant fitted probabilities onto the probability simplex, and then reports the modal support point.

Setting gradients=TRUE stores class-probability effects for one response level. If level is omitted, the base/reference response level levels(y)[1] is used. These effects are useful for asking how the fitted probability of a selected class changes with each covariate; they are not gradients of the arg max classification rule itself.

For book-length background, see Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*, especially the binary and multinomial choice material, and Li and Racine (2007), Chapter 5 *Conditional Density Estimation* together with Chapter 4 *Kernel Estimation with Mixed Data*.

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.

For formula fits, character-valued missing-data policies are resolved through the stats model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit na.action, including NULL, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as stats::model.frame.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after saveRDS() and readRDS(). Custom formula transformations used on new evaluation data must still be available.

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.

See Also

np.kernels, np.options, plot, npcdensbw.

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")

birthwt$low <- factor(birthwt$low)
birthwt$smoke <- factor(birthwt$smoke)
birthwt$race <- factor(birthwt$race)
birthwt$ht <- factor(birthwt$ht)
birthwt$ui <- factor(birthwt$ui)
birthwt$ftv <- ordered(birthwt$ftv)

with(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~smoke+
                   race+
                   ht+
                   ui+
                   ftv+
                   age+
                   lwt,
                   family=binomial(link=logit))

# Generate the confusion matrix and correct classification ratio

cm <- table(factor(low, levels = 0:1),
            factor(ifelse(fitted(model.logit)>0.5, 1, 0), levels = 0:1))
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=low~smoke+
                race+
                ht+
                ui+
                ftv+
                age+
                lwt,
                data=birthwt)

model.np <- npconmode(bws=bw)

# Compare confusion matrices from the logit and nonparametric model

# Logit

cm
ccr

# Nonparametric
summary(model.np)

# Predict modal classes and fitted class probabilities at selected rows
new.birthwt <- birthwt[1:5, c("smoke", "race", "ht", "ui", "ftv", "age", "lwt")]
predict(model.np, newdata=new.birthwt)
predict(npconmode(bws=bw, probabilities=TRUE),
        newdata=new.birthwt, type="prob")

})

# 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")
with(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(factor(low, levels = 0:1),
            factor(ifelse(fitted(model.logit)>0.5, 1, 0), levels = 0:1))
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)

})

# EXAMPLE 3 (CLASS PROBABILITY EFFECTS): compute and plot
# class-probability gradients/effects for one selected response level.
# This example uses a small artificial sample so that it runs quickly.

set.seed(42)
n <- 100
x <- seq(-1, 1, length.out=n)
y <- factor(rbinom(n, 1, plogis(1.5*x)), levels=0:1)

model.effects <- npconmode(y~x,
                           regtype="ll",
                           bwmethod="cv.ls",
                           nmulti=1,
                           probabilities=TRUE,
                           gradients=TRUE)

gradients(model.effects)
plot(model.effects)
plot(model.effects, view="fixed", neval=25)
plot(model.effects, gradients=TRUE)
plot(model.effects, gradients=TRUE, view="fixed", neval=25)

## A two-continuous-predictor fit can be displayed as a probability surface.
## The default level is the base/reference response level.

z <- runif(n, -1, 1)
y2 <- factor(rbinom(n, 1, plogis(1.5*x - z)), levels=0:1)
model.surface <- npconmode(y2~x+z,
                           regtype="ll",
                           bwmethod="cv.ls",
                           nmulti=1,
                           probabilities=TRUE)
plot(model.surface, perspective=TRUE, neval=15)

## End(Not run)
```

---

npcopula *Kernel Copula Estimation with Mixed Data Types*

Description

npcopula estimates a mixed-data kernel copula distribution or copula density. It can be called with an existing unconditional distribution/density bandwidth object, or with a one-sided formula in which case the appropriate bandwidth object is selected internally. Continuous marginal kernel bounds retain their original variable identity when continuous and categorical columns are mixed or reordered.

Usage

```
npcopula(bws, ...)

## S3 method for class 'formula'
npcopula(bws,
         data = NULL,
         u = NULL,
         target = c("distribution", "density"),
         evaluation = c("grid", "sample"),
         neval = 30,
         n.quasi.inv = 1000,
         er.quasi.inv = 1,
         ..., se = FALSE)

## Default S3 method:
npcopula(bws,
         data,
         u = NULL,
         target = NULL,
         evaluation = c("sample", "grid"),
         neval = 30,
         n.quasi.inv = 1000,
         er.quasi.inv = 1,
         ..., se = FALSE)

## S3 method for class 'npcopula'
predict(object,
        newdata = NULL,
        u = NULL,
        se.fit = FALSE,
        output = c("vector", "object", "data"),
        ...)

## S3 method for class 'npcopula'
plot(x,
     perspective = TRUE,
     view = c("rotate", "fixed", "surface", "contour", "image",
              "empirical", "all"),
     renderer = c("base", "rgl"),
     errors = c("none", "bootstrap", "asymptotic"),
```

```
band = c("pointwise", "pmzsd", "bonferroni",
         "simultaneous", "all"),
alpha = 0.05,
bootstrap = c("inid", "fixed", "geom"),
B = 9999,
center = c("estimate", "bias-corrected"),
boot.control = np_boot_control(),
output = c("plot", "data", "plot-data", "both"),
legend = TRUE,
theta = 0.0,
phi = 20.0,
xlab = "u1",
ylab = "u2",
zlab = NULL,
main = NULL,
col = NULL,
border = .np_plot_color("surface_border"),
zlim = NULL,
gaussian.draws = 9999L,
random.seed = 42L,
...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, source data, and formula route.

bws	bandwidth specification or one-sided formula. A bandwidth specification is either an unconditional distribution bandwidth object returned by npudistbw, for a copula distribution, or an unconditional density bandwidth object returned by npudensbw, for a copula density. If bws is a formula such as ~ x + y, npcopula first calls npudistbw when target="distribution" and npudensbw when target="density"; additional bandwidth-selection arguments in ... are forwarded to that bandwidth selector.
data	data frame containing the variables used to construct bws; when bws is a formula, data is passed to the bandwidth selector. Copulas are defined here for numeric and ordered variables; unordered factors are rejected.

Copula Target And Evaluation Grid: These arguments control whether a copula distribution or density is estimated and where it is evaluated.

er.quasi.inv	fraction passed to extendrange when constructing the marginal quasi-inverse grid used for supplied probability values u. See Details.
evaluation	evaluation route used when bws is a formula. The default "grid" constructs a plot-ready two-dimensional probability grid when u is omitted. Use "sample" to evaluate at the sample realizations.
n.quasi.inv	number of grid points used to compute each marginal quasi-inverse when u is supplied or automatically generated.

`newdata` optional prediction data for `predict.npcopula`. This is a compatibility alias for probability coordinates in columns named `u1`, `u2`, ... It is not interpreted as raw data on the original variable scale. If both `u` and `newdata` are supplied, `u` takes precedence.

`neval` number of probability values per margin in the automatically generated two-dimensional grid used when `bws` is a formula, `evaluation="grid"`, and `u` is omitted. This is a constructor argument; `plot` uses the object's stored probability coordinates and rejects plot-level `neval`.

`target` target used when `bws` is a formula. The default `"distribution"` estimates a copula distribution using `npudistbw`; `"density"` estimates a copula density using `npudensbw`. When `bws` is already a bandwidth object, the target is inferred from that object and a conflicting explicit `target` is rejected.

`u` optional matrix or data frame of marginal probability values in $[0, 1]$. Each column corresponds to one variable in the copula. If supplied, `npcopula` evaluates on the Cartesian product `expand.grid(u)`. For two-dimensional displays, the clearest spelling is often `data.frame(u1 = u1.seq, u2 = u2.seq)`; names matching the original variables are also accepted here because `u` explicitly denotes probability coordinates. Repeated axis values retain all original evaluation rows. Asymptotic simultaneous bands evaluate and calibrate each identical target once, then repeat its limits at the original rows; pointwise and Bonferroni intervals are unchanged. A surface draws one representative of each repeated coordinate only when the corresponding points and drawn interval endpoints agree exactly, including missingness; conflicting duplicates are rejected rather than averaged. Surface drawing requires at least two distinct values on each axis, whereas `output="data"` also retains singleton-axis grids. If omitted with a formula route and `evaluation="grid"`, a two-dimensional grid is generated automatically.

`object` an object of class `"npcopula"` returned by `npcopula`.

`se` logical value; default `FALSE`. Compute and retain asymptotic standard errors only when `TRUE`. This is a named fitting control, not a bandwidth-selection option.

`se.fit` logical value. With vector output, `TRUE` returns a list of fitted values and standard errors. New evaluation requests compute the required errors; prediction without new evaluation arguments reads the stored results and requires a fit made with `se=TRUE`. Object and data output retain their existing return shapes.

Plot Display Controls: These arguments control how a two-dimensional grid copula object is displayed.

`x` an object of class `"npcopula"` returned by `npcopula`.

`border` border color for surface facets and interval wireframes when `view="surface"`.

`col` optional surface or image colors. If omitted, `plot.npcopula` uses the same viridis `hcl.colors` palette used by the other modern surface plot methods.

`legend` logical or legend-control value used for `band="all"` interval overlays. Use `FALSE`, `NULL`, or `NA` to suppress the interval legend.

`main`, `xlab`, `ylab`, `zlab`
plot titles and axis labels.

output — return mode. For plot.npcopula, "plot" draws the plot, "data" returns the plotted data with interval columns when requested, and "plot-data" draws and returns the plotted data. "both" is accepted as an alias for "plot-data". For predict.npcopula, "vector" returns fitted copula values, "object" returns the evaluated "npcopula" object, and "data" returns as.data.frame() on that object.

perspective — logical value. If TRUE, draw a surface display; otherwise use view to choose "contour" or "image". The "empirical" and "all" views ignore this argument.

phi, theta — viewing angles passed to persp or to the shared **rgl** surface renderer. The defaults match the package-wide surface-plot defaults used by the other perspective plot methods. As with those methods, the exact default pair theta = 0 and phi = 20 is remapped internally for renderer="rgl" to account for the different viewing-angle convention used by **rgl**; explicitly supplied non-default angles are passed through.

renderer — plotting renderer for surface displays. "base" uses persp. "rgl" uses the shared interactive **rgl** surface renderer when the suggested package **rgl** is installed. Both renderers use the package-wide viridis surface palette and neutral-gray three-face reference grid.

view — display type for grid output. The default "rotate" draws a rotating base persp surface using the same frame step and delay as the other package perspective plots. Use "fixed" for a single fixed surface, "surface" as a backward-compatible fixed-surface alias, or "contour" and "image" with perspective=FALSE. Use "empirical" to plot the empirical copula coordinates. Use "all" for a base-graphics four-panel display containing the copula contour, copula surface, empirical copula coordinates, and a copula-density surface. The "all" view is not currently supported with renderer="rgl".

zlim — optional z-axis limits for surface displays.

Plot Interval Controls: These arguments add asymptotic or bootstrap intervals to two-dimensional surface plots.

alpha — nominal size used for asymptotic or bootstrap intervals when errors!="none".

B — number of bootstrap replications when errors="bootstrap".

band — interval type for plotted surfaces. Supported values are "pointwise", "pmzsd", "bonferroni", "simultaneous", and "all". The "all" option overlays pointwise, simultaneous, and Bonferroni wireframes where available.

bootstrap — bootstrap resampling method used when errors="bootstrap"; supported values are "inid", "fixed", and "geom". Wild bootstrap is not defined for copula surfaces.

boot.control — optional np_boot_control object. For copula surfaces the block length is used by the "fixed" and "geom" block bootstrap routes.

center — centering convention for bootstrap intervals. Copula plots support "estimate" and "bias-corrected". Copula plots do not currently have a validated oversmoothed pilot correction; unsupported bootstrap bias-correction combinations fail clearly.

gaussian.draws, random.seed
: Gaussian maximum simulation count and reproducible seed for asymptotic band="simultaneous" or band="all". The caller's RNG state is restored. These controls do not change bootstrap replication counts.

errors
: interval route for plot.npcopula: "none", "asymptotic", or "bootstrap". Intervals are available for two-dimensional grid evaluation output and are drawn as transparent wireframes over the copula surface.

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart, prediction/evaluation route, or graphics renderer as appropriate.

...
: additional arguments supplied to npudistbw or npudensbw when npcopula computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth-selection controls such as bwmethod, bwtype, and bwscaling, kernel/support controls such as ckertype, ckerorder, and ckerbound, categorical kernel controls such as ukertype and okertype, and search controls such as nmulti and scale.factor.search.lower are supplied. In predict.npcopula, additional arguments are passed to npcopula for evaluation with the stored bandwidth object and training data. In plot.npcopula, additional arguments are passed to the selected graphics routine, such as persp, contour, image, or the shared **rgl** renderer.

Details

Formula variables of class ts, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

npcopula computes the nonparametric copula distribution or copula density using marginal quasi-inversion. For the distribution target, Sklar's theorem gives

$$C(u_1, \ldots, u_d) = H(F_1^{-1}(u_1), \ldots, F_d^{-1}(u_d)),$$

where H is the joint distribution and F_j^{-1} is the quasi-inverse of marginal distribution F_j. For the density target, the estimated copula density is

$$c(u) = \frac{f(x_u)}{\prod_{j=1}^{d} f_j(x_{u,j})}, \quad x_{u,j} = F_j^{-1}(u_j),$$

with numerator and marginal denominators estimated using the selected mixed-data kernel bandwidths.

If u is provided, expand.grid is called on u. As the dimension increases this can become unwieldy because a grid with m points in each of d margins has m^d rows. Therefore the formula route automatically generates a probability grid only for two-dimensional copulas. For higher-dimensional copulas, supply u explicitly or use evaluation="sample". The predict.npcopula method follows these same probability-coordinate semantics: use u for probability values named however convenient, or use newdata only with u1, u2, ... columns.

The 'quasi-inverse' is computed via Definition 2.3.6 from Nelsen (2006). An equi-quantile grid on the data range of length `n.quasi.inv/2` is combined with an equi-spaced grid on the data range extended by `er.quasi.inv` and intersected with the fitted marginal kernel's bounds; the sorted union forms the grid used for marginal inversion. If requested probability values lie outside the attainable estimated marginal distribution range, they are reset to the nearest attainable endpoint. Inspect the returned `u` columns when endpoint behavior matters. Higher-order continuous CDF kernels can yield nonmonotone estimated marginal CDFs on the finite inversion grid; the quasi-inverse follows the first grid point whose estimated CDF reaches the requested probability.

The `plot.npcopula` method supports base `persp`, `contour`, and `image` displays for two-dimensional grid output. Central and uncertainty surfaces share one coordinate ordering: probability axes are sorted for display, while fitted values and returned plot data keep their supplied-grid order. Duplicate display coordinates are collapsed only when their fitted values and interval endpoints agree; conflicting duplicates are rejected. Asymptotic simultaneous calibration uses the unique targets after exact agreement of their fitted values, physical coordinates and marginal standard errors. Repeating a target does not change the maximum of its Gaussian process. Calibration metadata records the unique grid size and, when duplicates occur, the original row count and target mapping. All supplied rows remain in returned plot data; existing pointwise and Bonferroni intervals retain their original calibration. Surface plots use the package-wide viridis default palette, detailed perspective ticks, and the same default viewing angles and base-graphics rotation cadence as the other surface plot methods. `renderer="rgl"` requests the shared interactive **rgl** surface renderer, using the same default-angle remapping used by the other package surface plots. For mixed ordered margins and endpoint-clamped continuous margins, grid displays are drawn against the requested probability grid, while the returned `u` columns retain the attainable marginal probability values produced by quasi-inversion. The `"empirical"` view plots empirical copula coordinates, and `"all"` gives a base-graphics four-panel diagnostic display.

Standard-error extraction with `se(object)` only reads stored results; it does not silently refit the model. To add errors without repeating bandwidth selection, use `npcopula(bws=object$bws, data=object$data, u=object$u.input, se=TRUE, n.quasi.inv=object$n.quasi.inv, er.quasi.inv=object$er.q`
The default skips uncertainty-only work. Explicit asymptotic plot requests obtain their needed errors even if the original fit omitted them.

For grid surfaces, `plot.npcopula` can add asymptotic or bootstrap intervals. Distribution-copula asymptotic intervals use the joint distribution standard error evaluated at the marginal quasi-inverse grid. For fixed bandwidths, density-copula standard errors use the centered empirical influence of $c(u) = f(x_u)/\prod_j f_j(x_{u,j})$, including the same-sample covariance of the joint density and all marginal densities. Query coordinates and selected bandwidths are held fixed. A zero marginal density or nonfinite influence produces an unavailable row standard error (`NA`) and a notice. Nearest-neighbor density-copula errors retain the existing joint-density error divided by the marginal-density product; their complete ratio and random-radius covariance is not covered by the fixed-bandwidth correction. Bootstrap intervals resample rows and recompute the plotted copula surface on the same probability grid. These interval conventions do not fully propagate uncertainty from the marginal quasi-inverse transform. `band="simultaneous"` calibrates a joint Gaussian maximum over the declared grid. Distribution copulas use the selected joint-CDF influence. Fixed-bandwidth density copulas use the complete same-sample ratio influence. Nearest-neighbor density simultaneous bands also combine the joint-density and all estimated marginal-density influences on the same observations, including their covariance in the density ratio. Generalized-neighbor factors retain the selected query radii as fixed; adaptive-neighbor factors include the existing random-radius contributions in both joint and marginal densities. The fitted inverse coordinates remain fixed for these working targets. A one-dimensional density ratio has exactly zero simultaneous variability when

its positive numerator and sole marginal are the same canonical primitive. These new simultaneous scales do not replace the existing nearest-neighbor pointwise or Bonferroni scales.

These are finite-grid selected-smoothing variability bands, not bias-corrected population or continuum confidence bands. Unavailable incumbent uncertainty, nonregular support/radius geometry and memory-limited grids retain NA simultaneous limits with a specific notice. Adaptive-neighbor density and distribution bands include regular finite-bound and beta-kernel routes through a separate simultaneous influence scale. The existing `se()`, pointwise and Bonferroni values are unchanged, including their `NA` values where marginal errors are not implemented. Thus `band="all"` may contain finite simultaneous limits together with unavailable marginal limits. Ordinary fractional neighbor counts use the point estimator's canonical rounding. Extended maximum-radius counts require a separate endpoint or tail law and remain unavailable for these bands. Numerical precision checks can also make a requested band unavailable; they never replace the fitted point estimates. Beta checks compare the same shape with an independent PDF implementation and propagate the discrepancy; this is a numerical agreement check, not a rigorous special-function bound. `band="all"` retains pointwise and Bonferroni intervals and adds the qualified simultaneous wireframe.

Value

`npcopula` returns an object of class `"npcopula"`. The main components are:

`copula`	estimated copula distribution value or copula density value.
`u1`, `u2`, ...	marginal probability coordinates associated with the sample realizations or evaluation grid.
`x`, `y`, ...	marginal quasi-inverse coordinates corresponding to the requested probability grid when grid evaluation is used.
`bws`	selected unconditional distribution or density bandwidth object.
`eval`	data frame containing the copula values, probability coordinates, and quasi-inverse coordinates. `as.data.frame(object)` returns this component for data-frame workflows.
`copulaerr`	stored asymptotic standard errors when `se=TRUE`; absent when they were not requested. Requested unavailable values retain their existing `NA` representation. The separate logical `se` component records the fitting request.

The source data, target, evaluation route, grid dimensions, and timing metadata are retained as list components. The functions `fitted`, `predict`, `se`, `summary`, `as.data.frame`, and `plot` support `"npcopula"` objects.

Book And Method Pointers

The copula distribution target is $C(u) = H(F_1^{-1}(u_1), \ldots, F_d^{-1}(u_d))$; the copula density target is $c(u) = f(x_u)/\prod_j f_j(x_{u,j})$. The mixed-data kernel implementation follows Racine (2015), with quasi-inversion in the sense of Nelsen (2006). For the underlying mixed-data density and distribution estimators, see Li and Racine (2007), Chapter 1 *Density Estimation*, Chapter 3 *Kernel Estimation with Mixed Data*, and Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions*.

Usage Issues

Use a `data.frame` rather than `cbind` for mixed data so that ordered variables remain ordered. Unordered factors are not valid for copula estimation in this implementation.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

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

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

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

Racine, J.S. (2019), *An Introduction to the Advanced Theory and Practice of Nonparametric Econometrics: A Replicable Approach Using R*.

See Also

`npudistbw`, `npudist`, `npudensbw`, `npudens`, `np.kernels`, `np.options`, `plot`

Examples

```
## Not run:
library("MASS")

## Example 1: bivariate mixed data, continuous x and ordered y.

set.seed(42)
n <- 1000
n.eval <- 30
rho <- 0.99
mu <- c(0, 0)
Sigma <- matrix(c(1, rho, rho, 1), 2, 2)
xy <- mvrnorm(n = n, mu = mu, Sigma = Sigma)
mydat <- data.frame(
  x = xy[, 1],
  y = ordered(as.integer(cut(xy[, 2],
    quantile(xy[, 2], seq(0, 1, by = .1)),
    include.lowest = TRUE)) - 1)
)

grid.seq <- seq(0, 1, length.out = n.eval)
grid.dat <- data.frame(u1 = grid.seq, u2 = grid.seq)

## Estimate the copula distribution from an npudistbw() object.

bw.cdf <- npudistbw(~ x + y, data = mydat, nmulti = 1)
copula <- npcopula(bws = bw.cdf, data = mydat, u = grid.dat)
summary(copula)
```

```
## Native plotting replaces the older manual contour(), persp(), and
## empirical scatterplot calls.

plot(copula, perspective = FALSE, view = "contour")
plot(copula, perspective = FALSE, view = "image")
plot(copula, view = "fixed", zlim = c(0, 1))
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula, view = "fixed", renderer = "rgl", zlim = c(0, 1))
plot(copula)

## Plot empirical copula coordinates from the retained sample data.

plot(copula, view = "empirical")

## Or request the four-panel base-graphics diagnostic display.

plot(copula, view = "all")

## Estimate and plot the copula density from an npudensbw() object.

bw.pdf <- npudensbw(~ x + y, data = mydat, nmulti = 1)
copula.dens <- npcopula(bws = bw.pdf, data = mydat, u = grid.dat)
summary(copula.dens)
plot(copula.dens, view = "fixed")
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula.dens, view = "fixed", renderer = "rgl")
plot(copula.dens)

## Intervals are available for two-dimensional grid surfaces.

plot(copula, errors = "asymptotic", band = "pointwise")
plot(copula, errors = "bootstrap", bootstrap = "inid", B = 399,
     band = "pointwise")

## Prediction evaluates the retained bandwidth object on a supplied
## probability grid.

predict(copula, u = data.frame(x = c(0.25, 0.75),
                               y = c(0.25, 0.75)))
predict(copula, newdata = data.frame(u1 = c(0.25, 0.75),
                                     u2 = c(0.25, 0.75)))

## Example 2: bivariate continuous data.

set.seed(42)
n <- 1000
n.eval <- 30
rho <- 0.99
mu <- c(0, 0)
Sigma <- matrix(c(1, rho, rho, 1), 2, 2)
xy <- mvrnorm(n = n, mu = mu, Sigma = Sigma)
mydat <- data.frame(x = xy[, 1], y = xy[, 2])
```

```
grid.seq <- seq(0, 1, length.out = n.eval)
grid.dat <- data.frame(u1 = grid.seq, u2 = grid.seq)

bw.cdf <- npudistbw(~ x + y, data = mydat, nmulti = 1)
copula <- npcopula(bws = bw.cdf, data = mydat, u = grid.dat)
summary(copula)

plot(copula, perspective = FALSE, view = "contour")
plot(copula, perspective = FALSE, view = "image")
plot(copula, view = "fixed", zlim = c(0, 1))
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula, view = "fixed", renderer = "rgl", zlim = c(0, 1))
plot(copula)
plot(copula, view = "empirical")
plot(copula, view = "all")

bw.pdf <- npudensbw(~ x + y, data = mydat, nmulti = 1)
copula.dens <- npcopula(bws = bw.pdf, data = mydat, u = grid.dat)
summary(copula.dens)
plot(copula.dens, view = "fixed", zlim = c(0, 40))
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula.dens, view = "fixed", renderer = "rgl",
       zlim = c(0, 40))
plot(copula.dens, zlim = c(0, 40))

## The formula interface is a shorter route when bandwidths do not need
## to be reused explicitly.

copula.short <- npcopula(~ x + y, data = mydat, neval = n.eval,
                         nmulti = 1)
plot(copula.short, view = "all")

## 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,
            B = 399,
```

```
random.seed = 42,
...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the two samples and any supplied bandwidths.

`bw.x`, `bw.y` optional common bandwidth specification. If neither is supplied, one bandwidth is selected on the pooled sample by least-squares cross-validation (`bwmethod = "cv.ls"`, overridable through `...`), then rescaled to the harmonic-mean sample size as described below. If only one is supplied it is reused for both samples without a search. If both are supplied their effective bandwidths, kernels and support specifications must agree; incompatible specifications are rejected. This test requires fixed bandwidths and symmetric kernels without boundary normalization. Nearest-neighbour bandwidths, beta kernels and finite continuous normalization bounds are rejected: the test's directed cross sum and variance formula assume symmetry. Unnormalized Gaussian, Epanechnikov and uniform kernels remain supported, including mixed continuous/categorical data. An active ordered Racine–Li–Yan kernel is also rejected because it is asymmetric and has a nonconstant self-diagonal. Other estimators' kernel support is unchanged.

`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).

Bootstrap Controls: These arguments control bootstrap replication and reproducibility settings.

`B` 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: Further arguments are passed to the bandwidth-selection routines used by the test.

`...` additional arguments supplied to specify the bandwidth type, kernel types, and so on. These selection arguments are used only when both `bw.x` and `bw.y` are absent; they do not modify a supplied bandwidth specification. To do this, you may specify any of `bwscaling`, `bwtype`, `ckertype`, `ckerorder`, `ukertype`, `okertype`.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

`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.

The same bandwidth is held fixed in both samples and every bootstrap replication. Automatic selection uses one pooled-sample search, followed by a *harmonic-mean reference-size adjustment*. Write $N = n_1+n_2$ and $n_H = 2/(1/n_1+1/n_2)$, where the sample sizes count complete observations. Retaining the pooled spread estimates, continuous bandwidths are multiplied by $(N/n_H)^{1/(2P+q)}$ and categorical bandwidths by $(N/n_H)^{2/(2P+q)}$, following the scale-factor convention of npudensbw. Here P is the continuous kernel order (two by default), and q is the number of continuous variables. For the default second-order kernel the continuous exponent is $1/(4+q)$. Categorical parameters cannot exceed their kernel's permitted maximum: an adjusted value above that maximum is set to the maximum and a warning names the affected variables.

Under a common density, the leading variance coefficient of the equally weighted average of the two density-estimation risks is $(1/n_1 + 1/n_2)/2 = 1/n_H$. This motivates the common reference size; it is not a claim of optimal test power or exact finite-sample calibration. The mixed-data and higher-order adjustments use the existing bandwidth scaling convention. This rule differs from the archived first-sample simulation implementation and from unadjusted pooled selection. To use an unadjusted pooled bandwidth, construct it with `npudensbw(dat = rbind(x, y), bwmethod = "cv.ls")` and supply it as `bw.x`. Explicit `bw.x`/`bw.y` are never rescaled. Likewise, `bws = ...`, `bandwidth.compute = FALSE` in `...` remains a manual specification, with scale factors interpreted using `x`'s complete observations rather than the pool.

Categorical coordinates use the union of the declared levels in both samples and supplied bandwidth objects, including unused levels. The same domain and effective kernel metadata are retained in every contraction and bootstrap draw. Ordered domains must have an unambiguous common order; otherwise declare the same ordered levels in both samples. Automatic selection uses both samples with this common metadata. A raw numeric bandwidth retains the default raw kernel-sum conventions; a density bandwidth object retains its selected density-kernel conventions, including normalized ordered Li–Racine masses. Selection arguments in `...` do not override an explicitly supplied bandwidth specification.

With fixed, unbounded bandwidths, pooled bootstrap samples are represented by multiplicities on their common support and processed in bounded groups. Multi-response kernel sums then reuse the fixed geometry for the first- and second-power contractions without constructing a dense kernel matrix. Boundary-normalized and nearest-neighbour specifications are not supported by this density-equality statistic.

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`.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

`npdeneqtest` compares two distributions by estimating the integrated squared difference between their density or probability functions. The two samples must have matching variables and data types because the statistic compares like with like.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests*, especially Section 12.2, and Chapter 13 *Nonsmoothing Tests*, especially Section 13.2. For density and mixed-data density context, see Racine (2019), Chapters 2 and 3.

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.

Incomplete rows are omitted independently from each sample before bandwidth selection, bootstrap resampling and computation of the statistic. Factor levels are retained. Each sample must contain at least two complete observations. Supplying one bandwidth reuses it for both samples; it is not reselected when the other bandwidth argument is missing.

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

`np.kernels`, `np.options`, `plot` `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,B=99)

if (interactive()) 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,B=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,B=99)

if (interactive()) 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,B=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,
          B = 399,
          random.seed = 42)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the paired samples being tested.

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 Controls: These arguments control bootstrap execution and reproducibility settings.

B	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
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.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

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.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

npdeptest compares the joint density or probability law of two variables with the product of their marginals. Departures from the product form indicate dependence, lack of fit, or predictability depending on how the two inputs are constructed.

For book-length background, see Li and Racine (2007), Chapter 13 *Nonsmoothing Tests*, especially Sections 13.4 and 13.5, and Chapter 12 *Model Specification Tests*, especially Sections 12.4.1 and 12.4.3. For entropy and density context, see Racine (2019), Chapter 2.

Usage Issues

The integration version uses deterministic tensor-product, corrected composite trapezoidal quadrature. Along each axis the domain extends eight relevant bandwidths beyond the observed support and uses two evaluation points per smallest relevant bandwidth. A registered native callback evaluates the Gaussian marginal and joint densities in bounded tiles, so package-owned working memory does not grow as a dense sample-by-quadrature array. The integration version is generally more computationally demanding than summation, but avoids the density division that can make the summation statistic unstable.

The summation version fuses both marginal density sums and the joint density sum in one registered native pass with constant auxiliary storage. During bootstrapping, bounded groups of sampled data.x indices share the invariant data.y marginal work while retaining every draw's positionwise pairing, reduction order, and output order. Workspace grows only with the sample size times the bounded group size; no dense sample-by-sample kernel matrix is retained. Counts alone are not used because they would discard the required pairing with data.y.

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

np.kernels, np.options, plot 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,B=99,method="summation")

if (interactive()) 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,B=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,B=99)

if (interactive()) 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,B=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, ..., B = 399)

## S3 method for class 'formula'
npindex(bws,
        data = NULL,
        newdata = NULL,
        y.eval = NULL,
        se = TRUE,
        ..., se.type = c("asymptotic", "bootstrap"))


## Default S3 method:
npindex(bws,
        txdat,
        tydat,
        nomad = FALSE,
        se = TRUE,
        ..., se.type = c("asymptotic", "bootstrap"))

## S3 method for class 'sibandwidth'
npindex(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        exdat,
        eydat,
        B = 399,
```

```
se = TRUE,
gradients = FALSE,
residuals = FALSE,
..., se.type = c("asymptotic", "bootstrap"),
y.eval = NULL)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

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).

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

txdat — a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

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 corresponding training data recoverable from the bandwidth object in the current session. For method="kleinspady", response values must be numeric 0 or 1. Factor responses are rejected rather than converted to internal category codes; explanatory variables may still be factors. The same response restriction applies to supplied evaluation outcomes.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside npindex.

nomad — logical shortcut passed through to npindexbw when bandwidths are computed inside npindex. When TRUE, the single-index bandwidth route fills any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npindexbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the single-index fit is evaluated and which evaluation quantities are returned.

`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`. If all computed kernel weights are zero at an external evaluation row, the undefined fitted quantities are returned as `NA`, with one notice per call. This policy is independent of the `se` and `gradients` switches; required training computations remain strict. If a bootstrap resample has no support at a requested row, its bootstrap standard error is `NA`; a supported original point estimate is retained. Replications are not dropped or redrawn.

`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.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

`y.eval` Selects the outcomes used for goodness-of-fit diagnostics. The default `NULL` uses evaluation outcomes when supplied in `newdata` or `eydat`. With evaluation predictors only, predictions are returned normally and diagnostics are `NA`. Explicit `TRUE` requires evaluation outcomes when evaluating new predictors. Explicit `FALSE` ignores outcomes in `newdata` and preserves historical training diagnostics, identified as such in the summary. Explicit `eydat` takes precedence. With no evaluation inputs, diagnostics describe training data.

Fitted Quantities And Inference: These arguments control residuals, gradients, and asymptotic or bootstrap inference.

`B` an integer specifying the number of bootstrap replications to use when `se=TRUE`, `se.type="bootstrap"`. Defaults to `399`.

`se` a logical value requesting asymptotic coefficient covariance and standard errors for the conditional mean and, when `gradients=TRUE`, gradients. Defaults to `TRUE` for this estimator. Set `FALSE` to omit all covariance and standard-error calculations, independently of whether gradient estimates are requested.

`se.type` the standard-error method for the conditional mean and gradients: `"asymptotic"` (the default) or `"bootstrap"`. The latter also supplies bootstrap standard errors for average gradients and uses `B` replications. Coefficient covariance is asymptotic in both modes: the fitted-value/gradient bootstrap holds beta and the selected bandwidth fixed while resampling observations. No bootstrap is run when `se=FALSE` or `se.type="asymptotic"`.

`gradients` a logical value indicating that you want gradients computed and returned. Defaults to `FALSE`. Their standard errors additionally require `se=TRUE`. Coefficient covariance does not require this flag.

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

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart when bandwidths are not supplied.

`...` additional arguments supplied to specify the parameters to the `sibandwidth` S3 method, which is called during estimation.

Details

Factor Predictors and Coefficient Inference: New bandwidth objects expand factors into the retained R contrast design described in npindexbw. Training refits, prediction and plots reuse the same columns, levels and contrasts. Numeric scores and older saved bandwidth objects retain their existing interpretation. Supply raw predictor data to the public fitting and prediction interfaces; expansion is automatic.

Coefficients, covariance and gradients use expanded design-column names. Gradients for factor contrast columns are derivatives of the numerical design extension, $g'(X'\beta)\beta_j$; they are not the effect of switching between observed categories. Such effects require comparing predictions at the desired categories. No coefficient is estimated for the intercept, which is absorbed into the unknown link.

Klein–Spady coefficient covariance uses the link derivative multiplied by covariates centered on their conditional means given the index. This corrects a legacy raw-covariate information calculation, including for numeric-only inputs. Ichimura retains its conditional-centered sandwich calculation. Coefficient inference requires additional conditional-moment computation; `se=FALSE` skips this work. This correction does not change the search objective or the conditional fitted-value uncertainty calculation.

Fixed kernel endpoints retained in the bandwidth object are also used by fitting, prediction, inference and bootstrap resamples. They are not recomputed from the evaluation grid or from a resampled training set.

With a formula-derived bandwidth object, supplying only `tydat` replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective `na.action` is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use `data` for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named `newdata` columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over `newdata`.

For a native bandwidth object, `newdata` supplies `exdat` and, in automatic mode, a named response column supplies evaluation outcomes. Explicit native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection or re-estimating index coefficients. For held-out assessment, estimate the coefficients and any data-selected bandwidths using the training sample.

For Ichimura fits trained on an unordered factor response (or an ordered response without numeric levels), evaluation outcomes must also be factors compatible with the training levels: raw numeric labels do not identify the factor-code scale used for fitting. In automatic mode such numeric outcomes leave diagnostics unavailable, without blocking predictions; explicit evaluation-outcome requests report an error. For ordered training responses with numeric levels, numeric evaluation outcomes on that same scale are also accepted.

Evaluation outcomes affect diagnostics only. In automatic mode, outcomes that cannot be evaluated, aligned, or validated are treated as unavailable for diagnostics; predictions remain available. Explicit `y.eval=TRUE` and explicitly supplied `eydat` retain strict validation. Missing outcomes do

not remove predictor-complete prediction rows. Time-series outcomes are matched to the predictor time coordinates; predictions outside the supplied outcome time support remain available but are not scored. Scores use observed outcome/prediction pairs, and are NA when none are available. Formula responses, including transformations, are evaluated only when their variables are present in newdata; an ambient response is never substituted. Predictor omission and row-restoration conventions are unchanged.

Formula variables of class ts, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Documentation guide: see np.kernels for kernels, np.options for global options, and plot, plot.np for plotting options.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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

Interpreting β Coefficients: Single-index coefficients are identified only up to scale, so npindexbw normalizes the first coefficient to one. The remaining coefficients describe relative movement in the index $X'\beta$, not standalone linear-model slopes. Holding other variables fixed, a one-unit increase in X_j changes the index by the same amount as a β_j-unit increase in X_1. Where the link derivative exists and is nonzero, this is also the local marginal-effect ratio relative to X_1. Choose the first continuous regressor deliberately when coefficient interpretation is important.
For $E[Y \mid X] = g(X'\beta)$, the component plotted for predictor X_j is

$$\frac{\partial E[Y \mid X]}{\partial X_j} = \beta_j g'(X'\beta)$$

Accordingly, plot(..., gradients=TRUE) displays every component against the common scalar index $X'\beta$; the y-axis identifies the predictor with respect to which the derivative is taken. These panels are not partial-dependence plots against the individual X_j values.

Optimizer Choice When Bandwidths Are Computed: When npindex computes its bandwidth object internally, optimizer controls in ... are passed to npindexbw. The default optim.method="Nelder-Mead" is a robust derivative-free starting point for low-dimensional searches, such as one bandwidth plus only a few index coefficients. For larger p, optim.method="BFGS" can be useful as a performance and sensitivity comparison, but users should compare objective values, convergence behavior, and fitted objects because the single-index criterion can be nonconvex and route-sensitive.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, npindex(..., nomad=TRUE) and npindexbw(..., nomad=TRUE) expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

Value

npindex returns a singleindex object. The generic functions fitted, residuals, coef, vcov, se, predict, and gradients, extract (or generate) estimated values, residuals, coefficients, variance-covariance matrix, asymptotic or bootstrap 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

`diagnostics.sample`
`"training"`, `"evaluation"`, or `"unavailable"`, identifying the sample used for goodness-of-fit statistics.

`diagnostics.nobs`
Number of observed outcome/prediction pairs scored.

`training.MSE` For Ichimura fits, the training mean squared error when training fitted values were computed, otherwise `NA`. Prediction's `residual.scale` retains this training quantity in its historical MSE units, independently of evaluation outcomes. No additional training fit is performed solely to populate this field.

`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

Book And Method Pointers

The single-index model reduces a multivariate predictor to an index, typically written $E[Y \mid X] = g(X'\beta)$ for continuous outcomes, with a normalization on β for identification. For binary outcomes, the Klein-Spady route estimates the corresponding binary-choice probability through the index. The fitted index regression, gradients, and average derivatives are computed from the selected index direction and kernel regression fit.

For book-length derivations, see Li and Racine (2007), Chapter 8 *Semiparametric Single Index Models*, especially Sections 8.1, 8.2.1, 8.4, 8.5, and 8.10. The later workflow treatment is Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*, especially the single-index material.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. For formula evaluation with `newdata`, an explicit `na.action` also applies to the evaluation predictors: `na.omit` drops incomplete predictor rows, `na.exclude` restores their positions with `NA` in predictions, standard errors and requested gradients, and `na.fail` rejects them. When omitted, evaluation uses the current model-frame missing-data policy. Evaluation outcomes are scored separately and do not remove prediction rows. With both replacement training data and evaluation data, the policy is applied separately to each frame. An evaluation-only request does not refilter retained training data. Explicit native evaluation arguments retain precedence and their existing complete-row handling. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

`vcov` extracts the asymptotic coefficient covariance computed when `se=TRUE`, independently of `gradients` and `se.type`. Its first row and column are zero because the first coefficient is normalized to one. A singular free-coefficient information matrix raises an informative error; the calculation does not substitute a generalized inverse or bootstrap. In particular, the local-constant uniform kernel has zero first derivative away from its support boundaries and cannot supply this covariance for free coefficients. Use `se=FALSE` for point estimates in that case. Selecting `se.type="bootstrap"` does not replace the coefficient covariance calculation, which remains asymptotic. The default inference requires additional training derivatives and moments; it can cost appreciably more than a point-estimate-only fit, especially when bandwidths have already been selected. Use `se=FALSE` to omit that work. Asymptotic fitted-value and gradient standard errors use the same scalar-index regression calculation as the asymptotic plot intervals, treating the fitted index coefficients as fixed. They inherit the residual normalization of Ruppert et al. (1997) described under Standard Errors in `npreg`. This is not the separate coefficient covariance returned by `vcov()`. Missing training residual information can therefore make individual fitted-value or gradient errors unavailable without removing their point estimates. Local-polynomial index fits also inherit the numerical-ridging policy described in `npreg`. Asymptotic standard errors of average gradients are not supplied; `mean.gerr` is available only in bootstrap mode.

Missing quantities can be computed without repeating bandwidth search. After `model <- npindex(...)`, use `model <- npindex(bws=model$bws, gradients=TRUE, se=TRUE)` to add gradients and asymptotic inference, or `se=FALSE` for gradients alone. This reuses both the fitted beta and bandwidth,

just as fitting an object returned by npindexbw() does. The training data must remain recoverable from the retained state/call, or be supplied explicitly. predict(..., se.fit=TRUE) uses the fitted object's stored se.type for new evaluation data unless an explicit se.type is supplied.

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.

See Also

np.kernels, np.options, plot, plot.np, npindexbw

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...

if (interactive()) Sys.sleep(5)

# Or you can visualize the input with plot.

if (interactive()) plot(bw)

if (interactive()) 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...
```

```
if (interactive()) Sys.sleep(5)

# Or you can visualize the input with plot.

if (interactive()) plot(bw)

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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,
          ...)

## Default S3 method:
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          bandwidth.compute = TRUE,
          basis = c("glp", "additive", "tensor"),
          bernstein.basis = FALSE,
          degree = NULL,
          degree.select = c("manual", "coordinate", "exhaustive"),
          search.engine = c("nomad+powell", "cell", "nomad"),
          nomad = FALSE,
          nomad.nmulti = 0L,
          degree.min = NULL,
          degree.max = NULL,
          degree.start = NULL,
          degree.restarts = 0L,
          degree.max.cycles = 20L,
          degree.verify = FALSE,
          nmulti,
```

```
            nomad.remin = FALSE,
            powell.remin = TRUE,
            only.optimize.beta,
            optim.abstol,
            optim.maxattempts,
            optim.maxit,
            optim.method,
            optim.reltol,
            random.seed,
            regtype = c("lc", "ll", "lp"),
            scale.factor.init.lower = 0.1,
            scale.factor.init.upper = 2.0,
            scale.factor.init = 0.5,
            scale.factor.search.lower = NULL,
            ...,
            nomad.opts = list())

## S3 method for class 'sibandwidth'
npindexbw(xdat = stop("training data xdat missing"),
            ydat = stop("training data ydat missing"),
            bws,
            bandwidth.compute = TRUE,
            .certify.selected = TRUE,
            nmulti,
            only.optimize.beta = FALSE,
            optim.abstol = .Machine$double.eps,
            optim.maxattempts = 10,
            optim.maxit = 500,
            optim.method = c("Nelder-Mead", "BFGS", "CG"),
            optim.reltol = sqrt(.Machine$double.eps),
            random.seed = 42,
            scale.factor.init.lower = 0.1,
            scale.factor.init.upper = 2.0,
            scale.factor.init = 0.5,
            scale.factor.search.lower = NULL,
            .restore.first.start = TRUE,
            ...,
            .fixed.h.lower = NULL)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, method label, and whether bandwidths are supplied or computed.

`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`.

`.certify.selected`
: internal logical control requiring one raw-objective replay of the selected search point. This argument is used by package search owners and is not intended for direct user control.

`.restore.first.start`
: internal logical control enabling bounded restoration of eligible automatically generated raw-invalid first optimizer evaluations. It is used by package search owners and is not intended for direct user control.

`.fixed.h.lower`
: internal physical-bandwidth floor passed from a NOMAD search to its fixed-degree refinement. The default retains the direct-call scale-factor domain. This argument is not intended for direct user control.

`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 design column i (see Factors and Index Coordinates) 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.

`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.

`data`
: This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

  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.

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

`method`
: the single index model method, one of either "ichimura" (Ichimura (1993)) or "kleinspady" (Klein and Spady (1993)). Defaults to `ichimura`.

`na.action`
: a function which indicates what should happen when the data contain `NA`s. 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`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

`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`. For `method="kleinspady"`, the response must be numeric 0/1 with both values present after jointly omitting rows with missing predictors or responses. Missing outcomes are permitted;

a complete sample containing only one response class is rejected. Factor responses are rejected rather than converted to internal category codes; explanatory variables may still be factors.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search.

`degree.max` optional scalar or integer vector giving upper bounds for automatic degree search when `degree.select != "manual"`.

`degree.max.cycles` positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.min` optional scalar or integer vector giving lower bounds for automatic degree search when `degree.select != "manual"`.

`degree.restarts` non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.select` character string controlling local-polynomial degree handling when `regtype="lp"`. `"manual"` (default) treats `degree` as fixed. `"coordinate"` performs cached coordinate-wise search over admissible degree values for the index smoother. `"exhaustive"` evaluates the full admissible degree grid when `search.engine="cell"`. For NOMAD-based search engines, any non-`"manual"` value requests direct joint search over the degree and bandwidth coordinates.

`degree.start` optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline for the index smoother.

`degree.verify` logical value indicating whether an automatic degree-search solution should be exhaustively certified over the admissible degree grid after the heuristic phase completes. With `search.engine="cell"`, this verifies a coordinate-search solution using repeated fixed-degree solves. With NOMAD-based search engines, this first runs the joint NOMAD search over bandwidth and degree coordinates, then verifies the result over the requested degree grid using repeated fixed-degree solves. The default is `FALSE` because certification can be substantially slower than the heuristic search. Set `degree.verify=TRUE` when degree-selection certification or sensitivity checking is more important than runtime.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

`scale.factor.init` deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

`scale.factor.init.lower` lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details. Random fixed-bandwidth restarts use the index scale at their own

drawn beta. After conversion to search coordinates, an automatically generated bandwidth outside the existing search bounds is set to the nearest bound without drawing a new random value. The first start and explicitly supplied or held bandwidths are not changed by this step.

`scale.factor.init.upper`
: upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

`scale.factor.search.lower`
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to `NULL`. If `NULL`, an existing bandwidth object's stored value is inherited when available; otherwise the package default `0.1` is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with `bandwidth.compute = FALSE`. Final fixed-bandwidth search candidates must also have a finite valid raw objective value. A supplied positive fixed bandwidth held by a `only.optimize.beta=TRUE` refit is not a bandwidth-search candidate and is not restricted by this control.

Local-Polynomial Model Specification: These arguments control the index smoother, local-polynomial basis, and fixed degree specification.

`basis`
: local polynomial basis selector used when `regtype="lp"`: one of `"glp"`, `"additive"`, or `"tensor"`. Ignored for `"lc"` and `"ll"`.

`bernstein.basis`
: logical flag used when `regtype="lp"`. For `basis="glp"`, `TRUE` selects the complete degree-graded, orthonormal shifted-Legendre representation defined in `npreg`; additive and tensor bases retain their fixed-coordinate-degree Bernstein representation. When automatic degree search is requested and `bernstein.basis` is not explicitly supplied, the search route defaults to `TRUE` for numerical stability. Explicit `bernstein.basis=FALSE` is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

`degree`
: integer degree vector for continuous predictors when `regtype="lp"`. When `degree.select="manual"`, this must be supplied explicitly. For single-index regression this is typically length one.

`regtype`
: a character string specifying local smoothing type for the nonparametric index regression fit used downstream in `npindex`. Supported values are `"lc"`, `"ll"`, and `"lp"`.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

`nomad`
: logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When `TRUE`, any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` are filled with `regtype="lp"`, `search.engine="nomad+powell"`, `degree.select="coordinate"`, `bernstein.basis=TRUE`, `degree.min=0L`, `degree.max=10L`,

degree.verify=FALSE, and bwtype="fixed". Explicit incompatible settings error immediately; in particular, explicit nomad=TRUE currently requires regtype="lp", bwtype="fixed", automatic degree search, no explicit degree, and search.engine %in% c("nomad", "nomad+powell"). This shortcut does not change the meaning of nmulti or nomad.nmulti: nmulti remains the outer restart count, while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in bw$nomad.shortcut for a returned object bw; when available, nomad.time records the direct-search component. The legacy field powell.time records the subsequent family-native R optim refinement selected by optim.method.

nomad.nmulti — non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when regtype="lp" and automatic degree search uses search.engine="nomad" or "nomad+powell". Defaults to 0L, which preserves the current one-start-per- restart behavior. This does not replace nmulti: nmulti controls outer restarts, while nomad.nmulti controls inner NOMAD multistarts within each outer restart.

nomad.opts — optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. NULL is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as NB_THREADS_PARALLEL_EVAL > 1.

nomad.remin — logical flag controlling the optional second NOMAD hot start. When TRUE, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to FALSE; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

search.engine — character string controlling the automatic local-polynomial search backend when regtype="lp" and degree.select != "manual". The compatibility value "nomad+powell" (default) performs direct joint search over the index-smoother bandwidth and degree using the native **crs** NOMAD C API, then applies one family-native R optim hot start from the NOMAD solution using optim.method. The value is retained for API compatibility; "nomad" omits this R-optimizer refinement. "cell" profiles the criterion over the admissible degree grid using repeated fixed-degree solves. Direct exhaustive degree-grid search is available with degree.select="exhaustive" and search.engine="cell". NOMAD-based search is intended to avoid exhaustive degree enumeration; users who want post-search certification over the requested degree grid can set degree.verify=TRUE. NOMAD-based search requires the suggested package **crs** to be installed.

Numerical Search Controls: These controls set search restart behavior.

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(2,ncol(xdat)).

powell.remin — compatibility argument shared with bandwidth families that use the Powell direction-set solver. Single-index fixed-degree and post-NOMAD refinement use R optim with optim.method; consequently, this argument does not alter their refinement behavior.

Optimization Controls: These arguments control outer optimization behavior for the semiparametric search.

`only.optimize.beta`
: minimizes the objective function with respect to beta while retaining the existing polynomial degree and bandwidth (or nearest-neighbor count). Automatic degree search, including NOMAD and cell search, is not run; `nomad`, `degree.select` and other degree-search controls do not replace the retained model specification. A bandwidth object carries its selected degree; with a numeric `bws` vector and `regtype="lp"`, supply `degree` explicitly. The public `scale.factor.search.lower` control does not restrict a supplied positive fixed bandwidth held in this way, whether inherited or explicitly supplied. The control is retained for later bandwidth searches. Positivity, finiteness and raw-objective validity checks still apply. When no positive bandwidth is supplied, the usual initialization rules apply. A supplied normalized beta vector is used as the first starting point, including when all its free coefficients are zero. With `nmulti=1`, this provides a single hot start; the usual convergence checks and recovery rules still apply.

`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 `10`.

`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 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.

  The default `"Nelder-Mead"` method is a reasonable, robust derivative-free default when the search dimension is small, such as one scalar bandwidth plus only a few index coefficients. As the number of predictors grows, the β search becomes higher dimensional and `optim.method="BFGS"` can be a useful performance and sensitivity comparison. Because the single-index objective can be nonconvex and route-sensitive, compare the selected objective value, convergence behavior, and resulting bandwidth object against the default search before relying on the result.

`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`.

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

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` additional arguments supplied to specify the parameters to the `sibandwidth` S3 method, which is called during the numerical search.

Details

Factors and Index Coordinates: For newly constructed bandwidth objects, factors enter the linear index through R model-matrix contrasts. An unordered factor with k observed levels ordinarily contributes $k - 1$ coefficients. Ordered factors use their R contrasts (polynomial contrasts by default), including when their level labels look numeric. To fit a single coefficient on specified scores instead, supply those scores as a numeric predictor.

For supported additive formulas, coefficient names and predictor columns follow `model.matrix()`. The intercept is absorbed into the unknown link and is not an index coefficient. The first retained design coefficient is normalized to one; the package does not reorder predictors to choose a different normalization. Thus a factor listed first can supply the normalized contrast coefficient. These are relative index coefficients, not linear-regression slopes. This representation does not add support for otherwise unsupported formula syntax.

Native data-frame calls use the corresponding additive design. A manually supplied `bws` vector contains one beta for each expanded design column, in model-matrix order, followed by the scalar bandwidth. Automatic start vectors use the same coordinates. Numeric-only inputs keep their existing representation. A new factor design with aliased columns or a constant direction is rejected: for example, `y ~ 0 + x + group` supplies all group indicators and has an unidentified constant direction. Use the ordinary contrast formula `y ~ x + group` or independent numeric columns.

Factor levels and the actual contrast matrices are retained at construction. Prediction uses this design even if global contrast options change or an evaluation sample omits some training levels. New evaluation levels are an error. Levels absent from the complete training sample do not identify coefficients. Custom contrasts must remain identifiable on that sample.

Older saved bandwidth objects without retained design metadata preserve their historical single-score factor representation. Reconstruct a bandwidth object to adopt the contrast design; the package does not silently reinterpret an existing coefficient vector. The `index.design` component records the raw predictor names, retained levels/contrasts, and expanded column mapping; coefficient names and dimensions refer to the expanded design.

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via `newdata` does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state.

The scale.factor.* controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

scale.factor.init controls the deterministic first search start when that control is exposed. scale.factor.init.lower and scale.factor.init.upper define the random multistart interval when exposed. scale.factor.search.lower is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is max(scale.factor.init, scale.factor.search.lower) when both controls are present, and the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). scale.factor.init.upper must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When scale.factor.search.lower is NULL, an existing bandwidth object's stored floor is inherited when available; otherwise the package default 0.1 is used. Explicit bandwidths supplied for storage with bandwidth.compute = FALSE are not rewritten by the search floor.

Categorical search-start controls such as dfac.init, lbd.init, and hbd.init have separate semantics and are not affected by scale.factor.search.lower.

Documentation guide: see np.kernels for kernels, np.options for global options, and plot, plot.np for plotting options.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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).

Invalid outer-search guidance uses the exact declared leave-one-out null: the constant-model least-squares null for Ichimura and the clipped Bernoulli likelihood null for Klein–Spady. Inner regression evaluation remains raw; only terminal invalidity is mapped at the outer boundary. The selected point is replayed once against the raw objective and is rejected if it is not finite and valid.

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.

Interpreting β Coefficients: The scale of $X'\beta$ and the scale of the unknown link function are not separately identified, so npindexbw normalizes the first coefficient to one. The remaining coefficients should therefore be read relative to the first explanatory variable, not as standalone linear-model slopes. Holding other variables fixed, a one-unit increase in X_j changes the index by the same amount as a β_j-unit increase in X_1. Where the link derivative exists and is nonzero, marginal-effect ratios satisfy $\partial g(X'\beta)/\partial X_j \;/\; \partial g(X'\beta)/\partial X_1 = \beta_j$. Signs and magnitudes are therefore relative to the chosen first variable and its units; choose the first continuous regressor deliberately when interpretation is important.

For fixed-bandwidth R optim searches, an automatic raw-invalid starting bandwidth is doubled while keeping its starting coefficients and degree fixed. This applies to all supported regression types, kernel orders and support options, including generated starts in automatic degree-search children. Each multistart uses its own coefficients' index scale. Only the first scalar evaluation already requested by optim is checked. A valid original start continues in that same optimizer invocation

without an additional objective evaluation. A rejected start does not incur a numerical-gradient stencil.

Restoration permits at most eight doublings (nine starting scalar evaluations) for each multistart or existing convergence-retry start and respects the existing search bounds. Each rejected start adds one scalar evaluation, which can be costly. The random-draw control `scale.factor.init.upper` is not a search upper bound, so restoration may exceed it. Explicit initial bandwidths and bandwidths held by `only.optimize.beta=TRUE` are never expanded; a raw-invalid first evaluation fails with the method, start/retry, attempted bandwidth range, count and reason. Budget exhaustion also fails the complete call.

This rule does not change subsequent optimizer trials, finite differences, penalties, convergence-retry controls or final exact-point certification. A valid start does not guarantee convergence or a valid final point. Increasing the bandwidth does not guarantee feasibility for higher-order kernels or local-polynomial fits; every attempted start must pass the same raw-objective check. Nearest-neighbour bandwidths and NOMAD's own direct search retain their existing feasibility mechanisms. An explicit NOMAD refinement start is checked but not expanded. No alternative solver or earlier multistart winner is substituted when restoration fails.

Choosing `optim.method`**:** The default `"Nelder-Mead"` search is often a good starting point for low-dimensional single-index problems because it uses only objective values and is comparatively tolerant of rough objective surfaces. When p is larger, especially when many coefficients enter β, `"BFGS"` may reduce runtime for some problems, but it is more dependent on local smoothness and on the quality of numerical gradient information. For important fits, try both methods with adequate multistarts and compare the final objective values, convergence messages, and fitted objects.

`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(2, ncol(xdat)) times) as is done for a number of examples. See the optimizer guidance above when choosing between the default `"Nelder-Mead"` method and `optim.method="BFGS"` for higher-dimensional index searches.

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.

When `regtype="lp"` and `degree.select != "manual"`, `npindexbw` can jointly determine the local-polynomial degree for the index smoother together with its bandwidth coordinate. With `search.engine="cell"`, the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search. With `search.engine="nomad"` or `"nomad+powell"`, the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API. For this family,

the compatibility value "nomad+powell" then performs one family-native R optim hot start using optim.method and retains the better of the direct NOMAD and refined solutions. For the index-smoother local-polynomial component, this polynomial-adaptive joint-search route follows Hall and Racine (2015).

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For single-index bandwidth selection it expands any missing values to the equivalent long-form call

```
npindexbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value nomad="auto" applies the same LP shortcut but leaves search.engine and degree.select eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the compatibility "nomad+powell" route. Explicit search.engine or degree.select choices are honored.

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.

Book And Method Pointers

Local-polynomial index search inherits the numerical-ridging policies described in npreg; see Seifert and Gasser (2000) there.

npindexbw selects the index direction, bandwidth, and when requested local-polynomial degree/search metadata for the single-index model. The continuous-outcome target is typically written $E[Y \mid X] = g(X'\beta)$, with β normalized for identification; binary outcomes use the Klein-Spady route.

For book-length derivations, see Li and Racine (2007), Chapter 8 *Semiparametric Single Index Models*, especially Sections 8.1, 8.2.1, 8.4, 8.5, and 8.10, and Racine (2019), Chapter 8.

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(2, 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.

Hall, P. and J.S. Racine (2015), "Infinite Order Cross-Validated Local Polynomial Regression," Journal of Econometrics, 185, 510-525.

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...

if (interactive()) 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...

if (interactive()) 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,
       bandwidth.divide = FALSE,
       compute.ocg = FALSE,
       compute.score = FALSE,
       kernel.pow = 1.0,
       leave.one.out = FALSE,
       operator = names(ALL_OPERATORS),
       permutation.operator = names(PERMUTATION_OPERATORS),
       return.kernel.weights = FALSE,
       ...)

## S3 method for class 'numeric'
npksum(bws,
       txdat = stop("training data 'txdat' missing"),
       tydat,
       exdat,
       weights,
       bandwidth.divide,
       compute.ocg,
       compute.score,
       kernel.pow,
       leave.one.out,
       operator,
       permutation.operator,
       return.kernel.weights,
       ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: Core inputs defining the symbolic interface, training data, optional weighted response, and evaluation points for the generalized product kernel sum.

bws	a bandwidth specification. This can be set as any suitable bandwidth object returned from a bandwidth-generating function, or a numeric vector.

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

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.

exdat — a p-variate data frame of sum evaluation points (if omitted, defaults to the training data itself).

formula — a symbolic description of variables on which the sum is to be performed. The details of constructing a formula are described below.

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`.

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.

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.

weights — a n by q matrix of weights which can optionally be applied to `tydat` in the sum. See details. In formula calls, full-length weights follow the same time alignment, `subset` and missing-value selection as the formula variables. Weights already having the selected sample length are also accepted when that length differs from the original sample length.

Kernel-Sum Operators And Output: Controls for bandwidth normalization, kernel powers, leave-one-out evaluation, operator variants, and returned kernel weights.

bandwidth.divide — a single, non-missing logical specifying whether or not to divide continuous kernel weights by their bandwidths. Use this with nearest-neighbor methods. Defaults to FALSE.

compute.ocg — a single, non-missing 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.

compute.score — a single, non-missing 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.

kernel.pow — a single, finite integer specifying the power to which the kernels will be raised in the sum. Defaults to 1.

`leave.one.out` a single, non-missing logical value specifying whether or not to compute the leave one out sums. The matching contribution is omitted; nearest-neighbor radii retain the full-sample geometry. This flag alone is not a literal deleted-sample estimator refit. Will not work if `exdat` is specified. 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.

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

Additional Arguments: Further arguments passed to the default kernel-sum method.

`...` additional arguments supplied to specify the parameters to the `default` S3 method, which is called during estimation.

Details

Formula variables of class `ts`, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

`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.

Exact ties can make the literal kth continuous nearest-neighbor radius zero. Such a bandwidth is undefined and is rejected; see np.kernels for the precise contract and remedies.

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, Epanechnikov, and beta associated kernels, and the uniform kernel. The beta family uses finite-support coordinate-aware routes described below. 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).

For ckertype="beta", npksum uses Chen's second-order associated PDF kernel as the base contribution on each declared interval. If $L = b - a$, its per-observation contribution is

$$\frac{1}{L}\,\mathrm{BetaPDF}\left(\frac{x_i - a}{L}; 1 + \frac{x - a}{L}\left(\frac{L}{h}\right)^2, 1 + \frac{b - x}{L}\left(\frac{L}{h}\right)^2\right).$$

The complete density scaling $1/L$ is part of this contribution. Consequently, a univariate beta density estimate is npksum(...)$ksum/n, not npksum(...)$ksum/(n*h). Local-constant regression uses the ordinary ratio of beta-weighted response and unit sums. Do not set bandwidth.divide=TRUE; that legacy normalization is intentionally rejected for beta kernels.

For order $2m$, the PDF and CDF operators use $\sum_{s=1}^{m}(-1)^{s+1}\binom{m}{s}K_2(h\sqrt{s})$. Thus the coefficient vectors for orders 2, 4, 6, and 8 are (1), $(2, -1)$, $(3, -3, 1)$, and $(4, -6, 4, -1)$. Higher-order PDF and CDF contributions are signed and are returned without clipping.

For operator="integral", the univariate beta contribution is

$$I_{(x-a)/L}\left(1 + \frac{x_i - a}{L}\left(\frac{L}{h}\right)^2, 1 + \frac{b - x_i}{L}\left(\frac{L}{h}\right)^2\right).$$

This observation-centered analytic CDF is not the integral of the evaluation-centered PDF contribution with respect to x. It is exactly zero and one at the declared endpoints, is monotone for order 2, and has the uniform-CDF limit as h diverges. In several dimensions, operator may mix "normal", "integral", and "convolution" beta factors coordinate by coordinate.

For operator="convolution", the training and evaluation values are interpreted as the centers of two associated beta weights in the common observation coordinate. If their shapes are (α_1, β_1) and

(α_2, β_2), the returned factor is

$$\frac{1}{L}\frac{B(\alpha_1+\alpha_2-1, \beta_1+\beta_2-1)}{B(\alpha_1,\beta_1)B(\alpha_2,\beta_2)}.$$

This symmetric analytic L^2 overlap includes its complete original-scale $1/L$ normalization. Its diagonal is the local beta roughness. It is neither a translation convolution nor the target-coordinate integrated-square term used by bounded density `cv.ls`; those objectives retain their existing quadrature of the actual estimator. Beta PDF, CDF, and overlap factors may be mixed across dimensions.

This beta surface requires continuous `txdat`, `ckerorder` in `c(2,4,6,8)`, `ckerbound="fixed"` or `"range"`, finite lower and upper bounds for every dimension, `operator="normal"`, `operator="integral"`, and/or `operator="convolution"`, and `kernel.pow=1`. It supports multiple continuous dimensions, response and case-weight columns, leave-one-out sums, and returned kernel weights. All three operators support `bwtype="fixed"`. The PDF, CDF, and convolution operators additionally support `bwtype="generalized_nn"` and `bwtype="adaptive_nn"`; their evaluation-specific and observation-specific metric bandwidths are the distances returned by the existing observation-support nearest-neighbor implementation, including its tie and extended-NN policy. Generalized overlap evaluates generalized-NN distances at both centers. Adaptive overlap keeps adaptive bandwidths attached to training centers; a genuinely new evaluation center uses the established generalized-NN auxiliary bandwidth. Fixed and adaptive order-2 CDF factors are monotone in their target. Generalized-NN CDF factors remain in $[0,1]$ with exact endpoints, but their target-varying bandwidth can produce local nonmonotonicity. Orders 4, 6, and 8 may additionally leave $[0,1]$ or be locally nonmonotone in every bandwidth mode; no clipping or monotone repair is applied. One-coordinate analytic first derivatives and their continuous-coordinate permutations are available as documented below. Higher and mixed derivatives, categorical-product, score, OCG, and other unsupported beta operations fail with an explicit condition. See `np.kernels` for the bandwidth convention, orientation, and references.

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.

For `ckertype="beta"`, `operator="derivative"` is the analytic first derivative of the associated PDF contribution with respect to its evaluation coordinate. A direct operator vector may contain one derivative coordinate, while `permutation.operator="derivative"` returns one derivative product per continuous coordinate. Set `bandwidth.divide=FALSE`; beta contributions already carry their original-scale units. Generalized- and adaptive-NN bandwidths are held fixed during differentiation. Exact matching support endpoints can produce signed infinite scalar derivatives; see `np.kernels` for the endpoint and ratio-cancellation conventions. Signed infinities are returned as `-Inf/Inf`, not recoded as missing values; `NA_real_` is reserved for unavailable or indeterminate results.

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.

For categorical variables with kernel.pow=1, the score differentiates the requested operator with respect to its categorical bandwidth: ordinary weights for "normal", overlap weights for "convolution", and ordered cumulative weights for "integral". An unordered "integral" retains its ordinary-weight convention. Convolution scores include both factors and, where applicable, both normalizers.

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.

Ordered evaluation values used with compute.ocg=TRUE must belong to the bandwidth object's retained category table, because this operation requires an adjacent reference category. Other ordered kernel evaluations retain their usual numeric extrapolation behavior.

When a permutation, score, or OCG operation selects only a subset of variables, the corresponding blocks in p.ksum and, when requested internally, derivative kernel weights are packed without empty coordinates and follow the original data-column order among the selected variables.

The option return.kernel.weights=TRUE returns a matrix of dimension 'number of training observations' by 'number of evaluation observations' containing generalized product kernel weights, not response- or observation-weighted sums. The bandwidth.divide flag controls normalization of ksum, not this raw kw export. For purely continuous data with fixed physical bandwidths h and unbounded ordinary location kernels, colMeans(foo$kw)/prod(h) gives a density estimate when every operator is "normal". With every operator "integral", the corresponding distribution estimate is colMeans(foo$kw), without division by the bandwidth product. Mixed operators require their coordinate-specific normalizations. Nearest-neighbor and associated beta kernels must use their own bandwidth/normalization conventions; one global bandwidth product is not a general reconstruction rule.

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)
p.ksum	packed permutation, score, or OCG sums, when requested

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.

Chen, S.X. (1999), "Beta kernel estimators for density functions," Computational Statistics & Data Analysis, 31(2), 131-145. doi:10.1016/S0167-9473(99)00010-9.

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

if (interactive()) 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...

if (interactive()) 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")
with(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)

})

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

if (interactive()) 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")
with(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

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

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

if (interactive()) 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

})

# EXAMPLE 4: For this example, we conduct least-squares cross-validation
# for the fixed-bandwidth 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), then restore the user's options on exit.

old <- options(scipen=100, digits=12)
on.exit(options(old), add = TRUE)

# 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, bwtype="fixed",
                    bandwidth.divide=TRUE,
                    bws=h)$ksum/
             npksum(txdat,
                    leave.one.out=TRUE, bwtype="fixed",
                    bandwidth.divide=TRUE,
                    bws=h)$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, bwtype="fixed")

# 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...

if (interactive()) 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)

# EXAMPLE 6: The beta associated kernel for bounded data.  Its complete
# original-scale normalization is already included, so divide the sum by n,
# not by n*h.  The same weights can be used directly in a local-constant
# numerator/denominator ratio.

x <- c(0, 0.01, 0.04, 0.15, 0.4, 0.75, 1)
x.eval <- seq(0, 1, length.out=101)
h <- 0.12
```

```
beta.args <- list(txdat=data.frame(x=x),
                  exdat=data.frame(x=x.eval),
                  bws=h,
                  ckertype="beta",
                  ckerorder=2,
                  ckerbound="fixed",
                  ckerlb=0,
                  ckerub=1)

beta.density <- do.call(npksum, beta.args)$ksum/length(x)

y <- sqrt(x)
beta.mean <- do.call(npksum, c(beta.args, list(tydat=y)))$ksum/
             do.call(npksum, beta.args)$ksum

beta.cdf <- do.call(npksum,
                    c(beta.args,
                      list(operator="integral")))$ksum/length(x)

beta.overlap <- do.call(npksum,
                        c(beta.args,
                          list(operator="convolution")))$ksum

## End(Not run)
```

---

nplsqreg *Location-Scale Kernel Quantile Regression with Mixed Data Types*

Description

nplsqreg computes a location-scale kernel estimate of the conditional quantile function for a one dimensional dependent variable and mixed continuous, unordered factor, and ordered factor explanatory data. Unlike npqreg, which obtains conditional quantiles by inverting an estimated conditional distribution, nplsqreg estimates the requested conditional quantile surface directly using a locally weighted quantile-kernel construction.

Usage

```
nplsqreg(bws, ...)

## S3 method for class 'lsqregression'
predict(object, se.fit = FALSE, ..., tau = NULL)

## S3 method for class 'formula'
nplsqreg(bws, data = NULL, newdata = NULL, tau = 0.5,
       gradients = FALSE, residuals = FALSE, subset, na.action,
       gradient.order = 1L, se = FALSE, ...)

## S3 method for class 'lsqregressionbandwidth'
```

```
nplsqreg(bws, txdat = NULL, tydat = NULL,
       tau = bws$tau, se = FALSE, ...)

## Default S3 method:
nplsqreg(bws,
       txdat = stop("training data 'txdat' missing"),
       tydat = stop("training data 'tydat' missing"),
       tau = 0.5,
       exdat,
       gradients = FALSE,
       residuals = FALSE,
       gradient.order = 1L,
       se = FALSE,
       ...)
```

Arguments

object	a fitted lsqregression object.
se.fit	logical; whether prediction standard errors are returned. Defaults to FALSE.

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

bws	a formula, an lsqregressionbandwidth object returned by nplsqregbw, an rbandwidth object, a numeric bandwidth vector, or omitted for automatic bandwidth selection. Exact nplsqreg reuse is through the fitted object's $bws component; $reg.bws is internal regression state. A formula-based rbandwidth retains its training sample and bandwidth specification; supplying data explicitly replaces the training sample. With bandwidth.compute = FALSE, its bandwidths are held fixed in the quantile fit.
data	This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces. an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(bws).
subset	an optional vector specifying a subset of observations to be used by the formula method. When using a formula-based regression bandwidth object as a starting bandwidth, a row-changing subset requires explicit data; NULL and TRUE reuse all retained rows. A stored LSQ bandwidth object cannot be resubsetted; rebuild it with nplsqregbw() for another training sample.
na.action	a function specifying the action to take when missing values are found by the formula method.
txdat	a p-variate data frame of explanatory data used as training data. Defaults to the training data stored in bws.
tydat	a one dimensional numeric vector of dependent data. Defaults to the training response stored in bws.

Evaluation Data And Returned Quantities: These arguments control where the quantile regression is evaluated and which fitted quantities are returned.

`newdata` an optional data frame in which to look for evaluation covariates for formula fits. If omitted, the training data are used.

`exdat` a p-variate data frame of evaluation points. By default, evaluation takes place on `txdat`. The native `exdat` argument takes precedence over `newdata` when both are supplied.

`gradients` a logical value indicating whether gradients and categorical effects of the conditional quantile with respect to the conditioning variables should be computed and returned. Defaults to `FALSE`.

`se` a logical value indicating whether asymptotic standard errors should be computed and retained. Defaults to `FALSE`. Request `se=TRUE` when `se()` or gradient standard errors will be used.

`gradient.order` for a final `regtype="lp"` fit with `gradients=TRUE`, a positive integer scalar or one positive integer per continuous predictor. Each entry requests that coordinate's derivative order; it does not request mixed partial derivatives. The default is the first derivative. Ordered and unordered predictors retain their existing first-difference effect semantics.

`residuals` a logical value indicating whether residuals should be returned for training-data fits. Defaults to `FALSE`.

Quantile Index And Additional Controls: These arguments control the requested quantile probability and the bandwidth selection, prediction, or plotting route.

`tau` a numeric scalar or vector specifying the quantile probability or probabilities τ. Values must lie strictly in $(0, 1)$. For `predict()`, omit `tau` or use `NULL` to predict all stored quantiles. Otherwise supply distinct, exactly matching stored values in the desired order. Unfitted values are an error; prediction does not select bandwidths or recalibrate a new quantile.

`...` additional arguments supplied to `nplsqregbw` when bandwidths are computed internally, to `npreg` for the final transformed-response fit, or to plotting and prediction methods as appropriate. Common bandwidth-selection controls include `regtype`, `bwtype`, `nmulti`, `degree`, `nomad`, `search.engine`, `tau.search`, `delta`, `scale`, `regtype.pilot`, `nomad.pilot`, and `pilot.args`. The primary `nomad` control accepts `FALSE`, `TRUE`, or `"auto"`; AUTO uses exhaustive/cell degree search with one continuous smoothing dimension and NOMAD/Powell with two or more.

Details

The `tau` prediction argument must be named. Selecting one stored quantile returns the ordinary scalar prediction vector (or the scalar prediction list when `se.fit = TRUE`). Selecting several returns prediction and standard-error matrices with columns in the requested order, preserving the stored quantile labels and aligning `residual.scale`. Omitting `tau`, or supplying `NULL`, preserves the fitted object's original scalar or matrix output shape.

Prediction with `se.fit = TRUE` reports known training mean squared error in `residual.scale`, retaining MSE units, or `NA` when training provenance is unavailable. Evaluation outcomes do not

replace this training score. No additional fit is performed to recover it. This score concerns the regression on the transformed quantile response. For vector tau, residual.scale is a named vector with one training MSE for each requested quantile, matching the prediction columns.

A stored lsqregressionbandwidth object retains its training sample, pilots and transformed response. Replacement data, a row-changing subset, or a new na.action is an error: use nplsqregbw() to construct bandwidth state on the new sample. Use newdata or exdat for prediction instead.

Formula fits retain trained terms and prediction expressions, so data-dependent transformations are reused for newdata rather than recalculated on each prediction grid. The same terms are retained by the scalar children of vector-tau fits. Native exdat already contains model coordinates and is not transformed again. An unnamed evaluation argument in ... follows newdata for formula predictions and exdat for native predictions; name se.fit explicitly when using this positional form. Older saved objects without trained terms retain their formula-based fallback.

Omitted training rows are restored as NA in fitted values and residuals, including fits obtained from a retained bandwidth object and multiple quantile probabilities. Prediction restores omitted evaluation rows in both estimates and standard errors. Explicit training inputs own their row positions; a complete-case replacement is not padded using an older bandwidth object's omissions.

For native (non-formula) objects, named newdata columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over newdata.

Formula variables of class ts, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

The estimator follows the locally weighted quantile-kernel approach of Racine and Li (2017). Given a conditional scale pilot $\hat{\sigma}(X_i)$, define

$$Y_i^{\delta} = Y_i + \hat{\sigma}(X_i)\Phi^{-1}(\delta),$$

where $0 < \delta < 1$ and Φ^{-1} is the standard normal quantile function. For a requested quantile probability τ, nplsqregbw selects the bandwidths and δ by leave-one-out check-loss cross-validation. With the selected bandwidths and δ, nplsqreg then fits a kernel regression of Y_i^{δ} on X_i using the ordinary mixed-data machinery in npreg. The fitted mean of the transformed response is the estimated conditional quantile.

The scale pilot is interpreted as a conditional standard deviation. The default pilot estimates the conditional mean, smooths squared residuals, floors the fitted variance before taking square roots, and then uses the resulting positive scale vector in the quantile-kernel transformation. The local-linear residual-variance pilot follows the idea of Fan and Yao (1998); regtype.pilot can be used to select the pilot regression type independently of the final quantile fit.

If tau has length greater than one, tau.search="full" performs a separate bandwidth/δ search for each quantile while sharing the same pilot scale. The explicit tau.search="refined" route fits the central quantile first and warm-starts the remaining quantiles, recording the search order and warm-start provenance in the returned object. The conservative default is tau.search="full".

Gradients and categorical effects are those returned by the final npreg fit on the transformed response. Thus ordered-factor effects are finite-difference contrasts and unordered-factor effects follow the corresponding mixed-data regression semantics. For a local-polynomial fit, gradient.order is applied coordinate by coordinate to continuous predictors. Availability follows npreg: Apart

from the established all-degree-zero first-derivative compatibility route, components whose order exceeds their fitted polynomial degree are returned as NA with a warning whenever another continuous derivative or categorical first-difference effect is available; plots retain those components as empty panels. A request with no available component is rejected. No mixed partials, Hessians, derivative tensors, or multiple derivative orders are stored. With vector tau, the existing evaluation-by-coordinate-by-tau array layout is retained. The gradients method extracts exactly the stored order and rejects any different requested order; it does not recompute derivatives after fitting.

A scalar quantile can be displayed as a surface by explicitly supplying perspective=TRUE when there are two continuous or numeric-labelled ordered conditioning variables. Omitting perspective, or supplying perspective=FALSE, retains the usual slice layout. Vector quantiles and gradient plots also retain their slice layout. Surface uncertainty is calibrated over the complete displayed grid for that one tau. Both the base renderer and renderer="rgl", view="fixed" support level surfaces and their interval meshes. Returned evaluation rows retain the trained predictor names and ordered-factor support; numeric axis coordinates are used only for drawing. The data overlay can be disabled with data.overlay=FALSE.

Asymptotic plot(..., band="simultaneous") and band="all" use the accepted local regression response maps for each fitted transformed-response curve. Level, continuous-derivative and paired categorical-effect factors retain common observation columns, the selected local-constant or local-polynomial specification, and the actual native moment/ridge convention. Residual normalization uses the training fit map; reconstruction checks reject numerically unidentifiable maps instead of changing the fitted estimator. The fixed-regression fitting and bandwidth selection algorithms are unchanged.

Calibration is over the retained finite grid within one panel and one tau; it is not joint over taus or panels. gaussian.draws controls the Gaussian maximum simulation, and random.seed preserves reproducibility without changing the caller's RNG state. These are variability bands under the frozen transformation convention below; they do not add uncertainty from transformation selection or remove smoothing bias, and do not cover points between the declared grid. Coverage requires a valid joint Gaussian approximation and consistent local marginal scales; arbitrary growing or data-dependent grids do not carry a uniform-coverage guarantee. Unavailable marginal errors, unidentified influences or resource limits produce a specific unavailable joint-band notice.

Standard errors are the asymptotic standard errors from that final transformed-response regression, conditional on the selected bandwidths, pilot scale, and δ. They inherit the residual normalization described under Standard Errors in npreg (Ruppert et al., 1997), applied to the transformed response. This does not add uncertainty from estimating the pilot scale or transformation. Local-polynomial fits inherit the numerical-ridging policy documented there. For vector tau, quantiles are fitted separately, so estimated quantile curves are not monotonized and may cross in finite samples. Separate fits on the same observations do not imply zero cross-quantile sampling covariance.

Value

nplsqreg returns an object of class lsqregression. The generic functions fitted, quantile, se, predict, residuals, and gradients extract estimated conditional quantiles, asymptotic standard errors from the transformed-response regression, predictions, residuals when requested, and gradients or categorical effects. The functions summary, print, and plot support objects of this type. Local-polynomial fit objects retain the compact continuous-coordinate derivative order in $gradient.order.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When converting a formula-based `rbandwidth` without replacement `data`, an explicitly supplied `scale` vector must follow the retained complete training sample in its stored order. With replacement `data`, scale follows the formula's ordinary subset and missing-data handling.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

Author(s)

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

References

Fan, J. and Q. Yao (1998), "Efficient Estimation of Conditional Variance Functions in Stochastic Regression," Biometrika, 85, 645-660. doi:10.1093/biomet/85.3.645

Racine, J.S. and K. Li (2017), "Nonparametric conditional quantile estimation: A locally weighted quantile kernel approach," Journal of Econometrics, 201, 72-94. doi:10.1016/j.jeconom.2017.06.020

Racine, J.S. and I. Van Keilegom (2020), "A smooth nonparametric, multivariate, mixed-data location-scale test," Journal of Business & Economic Statistics, 38, 784-795. doi:10.1080/07350015.2019.1574227

See Also

`nplsqregbw`, `npreg`, `npqreg`, `np.kernels`, `np.options`, `plot.np`

Examples

```
## Not run:
data("Italy")

model.q <- nplsqreg(gdp ~ ordered(year), data = Italy,
                    tau = c(0.25, 0.50, 0.75))
plot(model.q)

model.med <- nplsqreg(gdp ~ ordered(year), data = Italy, tau = 0.50)
model.q2 <- nplsqreg(bws = model.med$bws, tau = 0.50)
plot(model.med, gradient = TRUE)
```

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

---

nplsqregbw *Bandwidth Selection for Location-Scale Kernel Quantile Regression*

Description

nplsqregbw selects bandwidths and the scalar location-scale shift parameter delta for nplsqreg by minimizing leave-one-out check loss.

Usage

```
nplsqregbw(bws, ...)

## S3 method for class 'formula'
nplsqregbw(bws, data = NULL, tau = 0.5,
       subset, na.action, ...)

## S3 method for class 'lsqregressionbandwidth'
nplsqregbw(bws, tau = bws$tau, ...)

## Default S3 method:
nplsqregbw(bws,
       xdat = stop("invoked without data 'xdat'"),
       ydat = stop("invoked without data 'ydat'"),
       tau = 0.5,
       tau.search = c("full", "refined"),
       delta = NULL,
       scale = NULL,
       regtype = c("lc", "ll", "lp"),
       regtype.pilot = c("auto", "ll", "lc", "lp"),
       nomad = FALSE,
       nomad.pilot = FALSE,
       pilot.args = list(),
       bandwidth.compute = TRUE,
       delta.bounds = c(1e-4, 1 - 1e-4),
       optim.control = list(maxit = 50L),
       ...,
       nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

bws — a formula, an lsqregressionbandwidth object, an rbandwidth object, a numeric bandwidth vector, or omitted for automatic selection. Ordinary npregbw rbandwidth objects may be used as starting values. Exact nplsqreg reuse is through an lsqregressionbandwidth object, typically fit$bws; fit$reg.bws is internal regression state.

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

an optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(bws).

subset — an optional vector specifying a subset of observations to be used by the formula method.

na.action — a function specifying the action to take when missing values are found by the formula method.

xdat — a p-variate data frame of explanatory data used as training data.

ydat — a one dimensional numeric vector of dependent data.

Quantile Index And Vector-Tau Search: These arguments identify the quantile probabilities and how vector-tau fits are orchestrated.

tau — a numeric scalar or vector specifying the quantile probability or probabilities τ. Values must lie strictly in $(0, 1)$.

tau.search — either "full" or "refined". The default "full" route performs a full bandwidth/delta search for each tau, sharing only the pilot scale. The explicit "refined" route fits the central tau first and uses the resulting bandwidths, delta, and when applicable selected degree vector as warm starts for the remaining quantiles.

Scale Pilot: These arguments control or supply the conditional standard deviation used in the location-scale transformation.

scale — an optional strictly positive scale vector interpreted as the conditional standard deviation at the training observations. If omitted, a pilot scale is computed automatically. In formula calls, a full-length vector follows the formula's time alignment, subset and missing-value selection. A vector already having the selected sample length remains accepted when that length differs from the original sample length.

regtype.pilot — regression type used for the pilot mean and residual-variance smooths. The default "auto" uses local linear smoothing when continuous predictors are present and local constant smoothing for categorical-only designs.

nomad.pilot — logical value indicating whether the pilot regressions should use the automatic NOMAD local-polynomial route when compatible with regtype.pilot. Defaults to FALSE.

pilot.args — optional named list of additional arguments supplied to the pilot npreg calls.

Bandwidth, Delta And Degree Search: These arguments control the main check-loss bandwidth and `delta` optimization.

`regtype` regression type for the main transformed-response fit and check-loss bandwidth search. This is independent of `regtype.pilot` and follows `npregbw` semantics.

`delta` optional starting value for the shift parameter. If omitted, the first start uses `0.5`; subsequent multistarts draw starting values from $(0, 1)$.

`delta.bounds` two numeric values giving the lower and upper bounds for `delta`. The interval must lie strictly inside $(0, 1)$.

`nomad` logical value or the character string `"auto"`. `TRUE` uses the local-polynomial NOMAD route for joint bandwidth, `delta`, and continuous degree-vector search. `"auto"` uses exhaustive/cell degree search for one continuous smoothing dimension and NOMAD/Powell for two or more. Defaults to `FALSE`.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry.

`bandwidth.compute` logical value indicating whether automatic bandwidth selection should be performed. Defaults to `TRUE`.

`optim.control` a list of controls passed to the Powell optimizer used for fixed degree searches and NOMAD hot-start refinement.

`...` additional bandwidth, kernel, local-polynomial, and search controls forwarded to the package bandwidth machinery. Common examples include `regtype`, `bwtype`, `bwmethod`, `nmulti`, `degree`, `basis`, `bernstein.basis`, `search.engine`, `degree.min`, `degree.max`, `degree.start`, `degree.restarts`, `powell.remin`, `nomad.remin`, and `nomad.nmulti`. The explanatory-variable kernel aliases used by `npqreg` and `npcdistbw`, such as `cxkertype`, `cxkerorder`, `uxkertype`, and `oxkertype`, are accepted and mapped to the corresponding regression-kernel controls. Response-side conditional-distribution kernel controls such as `cykertype` are not meaningful for the location-scale transformed-response estimator and fail clearly rather than being ignored.

Details

Regression searches and pilot fits inherit numerical ridging from `npregbw` and `npreg`.

For a requested quantile probability τ, `npnlsqregbw` chooses bandwidths and `delta` for the transformed response

$$Y_i^{\delta} = Y_i + \hat{\sigma}(X_i)\Phi^{-1}(\delta)$$

by minimizing the leave-one-out check loss. The selected bandwidths are then used by `npnlsqreg` for the final mixed-data `npreg` fit of the transformed response.

When the response fitted inside a leave-one-out fold differs from the response entering the requested check loss, the deleted fitted value is reconstructed with the fitted response while the residual is formed with the requested loss response. Invalid search guidance is the exact constrained constant-quantile check-loss null. A final NOMAD or Powell payload is accepted only when replay of its selected point returns a raw finite objective.

The default pilot scale is a residual scale: first npreg estimates the conditional mean, then npreg smooths squared residuals using the same bandwidth object, and the square root of the floored fitted variance is used as the scale. The local-linear pilot option follows Fan and Yao (1998). A user-supplied scale must be strictly positive and is interpreted as a conditional standard deviation, not a variance.

When nomad=TRUE, the search can include the continuous local-polynomial degree vector in addition to bandwidths and delta. The search.engine="nomad+powell" route uses NOMAD for degree search and Powell for hot-start refinement at the selected degree. When nomad="auto", a one-dimensional smoothing problem instead evaluates the complete requested degree lattice and jointly optimizes bandwidth and delta at every degree; higher-dimensional problems use the NOMAD/Powell policy. nomad.pilot remains an independent logical control and is not enabled by primary AUTO.

Value

nplsqregbw returns an object of class lsqregressionbandwidth. For vector tau, the object stores the per-tau bandwidth objects, selected delta values, objectives, fit order, warm-start provenance, and shared pilot scale.

Author(s)

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

References

Fan, J. and Q. Yao (1998), "Efficient Estimation of Conditional Variance Functions in Stochastic Regression," Biometrika, 85, 645-660. doi:10.1093/biomet/85.3.645

Racine, J.S. and K. Li (2017), "Nonparametric conditional quantile estimation: A locally weighted quantile kernel approach," Journal of Econometrics, 201, 72-94. doi:10.1016/j.jeconom.2017.06.020

Racine, J.S. and I. Van Keilegom (2020), "A smooth nonparametric, multivariate, mixed-data location-scale test," Journal of Business & Economic Statistics, 38, 784-795. doi:10.1080/07350015.2019.1574227

See Also

nplsqreg, npregbw, npreg, np.kernels, np.options

---

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,
       ...,
       se = FALSE)

## Default S3 method:
npplreg(bws,
        txdat,
        tydat,
        tzdat,
        nomad = FALSE,
        ...,
        se = FALSE)

## 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,
        ...,
        se = FALSE)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and partially linear training data.

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.
data	This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including

explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

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 corresponding training data recoverable from the bandwidth object in the current session.

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.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside npplreg.

nomad — logical shortcut passed through to npplregbw when bandwidths are computed inside npplreg. When TRUE, the partially linear bandwidth route fills any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npplregbw. The preset selects local-polynomial degrees and bandwidths separately for the nuisance regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$ before the final partially linear solve. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the partially linear regression is evaluated and which fitted quantities are returned.

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. Supplying only ezdat uses the original training txdat for the linear evaluation role. The two evaluation roles must have the same row count before omitting missing rows.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

se — a scalar logical value indicating whether to compute and retain coefficient standard errors and their asymptotic covariance matrix. Defaults to FALSE; coefficients, fitted values, residuals and bandwidth selection do not depend on this switch. Use TRUE before extracting coef(object, se = TRUE) or vcov(object). These extractors do not refit the model. If uncertainty was omitted, npplreg(bws = object$bws, se = TRUE) reuses the bandwidths without repeating search; supply the original training data if they are no longer recoverable from the bandwidth object.

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

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.

Additional Arguments: Further arguments are passed to npplregbw and its component npregbw searches when bandwidths are computed internally.

... — additional arguments supplied to npplregbw when npplreg computes bandwidths internally, or arguments needed to interpret a numeric or matrix bws specification. This is where bandwidth selection controls such as bwmethod, bwtype, and bwscaling, kernel/support controls such as ckertype, ckerorder, and ckerbound, categorical kernel controls such as ukertype and okertype, search controls such as nmulti and scale.factor.search.lower, and local-polynomial/NOMAD controls such as regtype, degree, bernstein.basis, degree.select, and nomad.nmulti are supplied. See npplregbw and npregbw for the complete bandwidth-selection argument surface, including the child-indexed matrix/list degree form for fixed local-polynomial partially linear fits.

Details

With a formula-derived bandwidth object, supplying only tydat replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective na.action is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use data for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named newdata columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over newdata.

For a native bandwidth object, newdata is split into exdat and ezdat using the retained linear and nonparametric predictor names. Names must identify both roles unambiguously; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

Nuisance regressions inherit numerical ridging from npreg. Their point fits do not apply its residual uncertainty correction to the separate partially linear coefficient covariance described below.

Documentation guide: see npplregbw for partially linear bandwidth selection, npregbw for the component nonparametric regression search controls, np.kernels for kernels, np.options for global options, and plot, plot.np for plotting options.

When bws is omitted, the formula and default methods call npplregbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a plbandwidth object, npplreg estimates with the stored bandwidth metadata in that object.

Argument groups for bandwidth selection are documented on npplregbw and, for the component nonparametric regressions, npregbw. The most common workflow is to choose the linear X variables and nonparametric Z variables first, then bandwidth/search controls for the Z-side nonparametric regressions, and finally local-polynomial/NOMAD controls when using polynomial-adaptive fits.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, npplreg(..., nomad=TRUE) and npplregbw(..., nomad=TRUE) expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

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:

For plregression objects, predict(object, se.fit = TRUE) returns a list with components fit and se.fit. The se.fit component contains asymptotic prediction standard errors computed from

the partially linear fitted-value linear operator and the squared residuals from the training fit. This explicit prediction request works even when the original model used `se = FALSE`; it does not require coefficient covariance or alter the original object.

With `se = TRUE`, coefficient covariance allows conditional heteroskedasticity. Let $\widetilde{X}$ be the residualized linear design, u the residuals from the training fit, and $A = \widetilde{X}'\widetilde{X}$. The returned covariance is $\{n/(n-p-q)\}A^{-1}\widetilde{X}'\mathrm{diag}(u^2)\widetilde{X}A^{-1}$, where n is the number of complete training observations, p is the number of linear regressors and q the number of smoothing variables. The multiplier retains the package's finite-sample convention; it is not an estimate of a smoother's effective degrees of freedom. Coefficient standard errors are square roots of its diagonal. This coefficient covariance and the fitted-value prediction errors above are different targets; neither changes the coefficient or bandwidth estimates.

`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, or `NULL` when `se = FALSE`
`xcoefvcov`	covariance matrix of the coefficients, or `NULL` when `se = FALSE`
`se`	whether coefficient uncertainty was requested
`bws`	the canonical bandwidth object, stored as a `plbandwidth` object
`bw`	backward-compatible alias for `bws`
`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

Book And Method Pointers

The partially linear model is $Y = X\beta + \Theta(Z) + \epsilon$. The Robinson-style construction residualizes Y and each column of X on Z, then solves the final least-squares problem using those residualized variables. Thus the nonparametric part of the problem is a sequence of conditional-mean regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$, followed by the parametric solve for β. In local-polynomial/NOMAD fits, the nuisance regressions may select child-specific continuous polynomial degrees.

For book-length derivations, see Li and Racine (2007), Chapter 7 *Semiparametric Partially Linear Models*, especially Sections 7.1, 7.2, and 7.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*, especially Robinson's partially linear model.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

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

`np.kernels`, `np.options`, `plot`, `plot.np` `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, se=TRUE)

# Print a summary of the model...

summary(pl)

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

if (interactive()) Sys.sleep(5)

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

coef(pl)
coef(pl, se = 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...

if (interactive()) Sys.sleep(5)

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

if (interactive()) 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 (9999) reducing them to 25
# in order to quickly compute standard errors in this instance - don't
# of course do this in general)

plot(bw,
     B=25,
     errors="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, se=TRUE)

# Print a summary of the model...

summary(pl)
```

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

if (interactive()) Sys.sleep(5)

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

coef(pl)
coef(pl, se = 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...

if (interactive()) Sys.sleep(5)

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

if (interactive()) 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 (9999) reducing them to 25
# in order to quickly compute standard errors in this instance - don't
# of course do this in general)

plot(bw,
     B=25,
     errors="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, ...)


## Default S3 method:
npplregbw(xdat = stop("invoked without data `xdat'"),
          ydat = stop("invoked without data `ydat'"),
          zdat = stop("invoked without data `zdat'"),
          bandwidth.compute = TRUE,
          bws,
          degree = NULL,
          degree.select = c("manual", "coordinate", "exhaustive"),
          search.engine = c("nomad+powell", "cell", "nomad"),
          nomad = FALSE,
          nomad.nmulti = 0L,
          degree.min = NULL,
          degree.max = NULL,
          degree.start = NULL,
          degree.restarts = 0L,
          degree.max.cycles = 20L,
          degree.verify = FALSE,
          scale.factor.search.lower = NULL,
          ftol,
          itmax,
          nmulti,
          nomad.remin,
          powell.remin,
          small,
          tol,
          ...)

## 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

Data, Bandwidth Inputs And Formula Interface: These arguments identify the linear, nonparametric, formula, and bandwidth inputs.

`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`.

`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.

`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.

`data`
: This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

  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.

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

`na.action`
: a function which indicates what should happen when the data contain `NA`s. 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`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

`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)

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search.

degree.max — optional scalar or integer vector giving upper bounds for automatic degree search over continuous zdat predictors when degree.select != "manual".

degree.max.cycles — positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for "manual" and "exhaustive" degree selection.

degree.min — optional scalar or integer vector giving lower bounds for automatic degree search over continuous zdat predictors when degree.select != "manual".

degree.restarts — non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for "manual" and "exhaustive" degree selection.

degree.select — character string controlling local-polynomial degree handling for the nuisance regressions of Y on Z and each column of X on Z. "manual" (default) treats degree as fixed. "coordinate" performs cached coordinate-wise search over admissible degree vectors. "exhaustive" evaluates the full admissible degree grid when search.engine="cell". For NOMAD-based search engines, any non-"manual" value requests a separate degree and bandwidth search for each nuisance regression.

degree.start — optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous zdat predictors.

degree.verify — logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for search.engine="cell".

Continuous Scale-Factor Search Controls: These controls define lower admissibility bounds for continuous fixed-bandwidth search.

scale.factor.search.lower — optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Local-Polynomial Model Specification: These arguments control fixed local-polynomial specification for the nonparametric component.

degree — for local-polynomial partially linear fits, polynomial degree specification for each continuous nonparametric regressor in zdat. When supplied with degree.select="manual",

bandwidth optimization treats this input as fixed. A numeric vector is used as a common degree vector for every nuisance regression. To specify child-specific fixed degrees, supply either a matrix with $p + 1$ rows and one column per continuous zdat predictor, or a list with $p + 1$ entries. The child order matches the bandwidth matrix/object order: first the regression of Y on Z (row or entry yzbw), followed by the regressions of the columns of X on Z. Named matrix rows or list entries are accepted and reordered to this canonical order.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

nomad
: logical or character shortcut for the recommended automatic local-polynomial NOMAD route for the partially linear nuisance regressions. When TRUE, any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype are filled with regtype="lp", search.engine="nomad+powell", degree.select="coordinate", bernstein.basis=TRUE, degree.min=0L, degree.max=10L, degree.verify=FALSE, and bwtype="fixed". Explicit incompatible settings error immediately; in particular, explicit nomad=TRUE currently requires regtype="lp", bwtype="fixed", automatic degree search, no explicit degree, and search.engine %in% c("nomad", "nomad+powell"). This shortcut does not change the meaning of nmulti or nomad.nmulti: nmulti remains the outer restart count, while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in bw$nomad.shortcut for a returned object bw; when available, nomad.time and powell.time record the direct-search and Powell-polish timing components. Each nuisance regression selects its own admissible degree and bandwidths.

nomad.nmulti
: non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when regtype="lp" and automatic degree search uses search.engine="nomad" or "nomad+powell". Defaults to 0L, which preserves the current one-start-per- restart behavior. This does not replace nmulti: nmulti controls outer restarts, while nomad.nmulti controls inner NOMAD multistarts within each outer restart. Advanced native NOMAD parameters may be supplied as nomad.opts through ...; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as NB_THREADS_PARALLEL_EVAL > 1.

nomad.remin
: logical flag controlling the optional second NOMAD hot start. When TRUE, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to FALSE; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

search.engine
: character string controlling the automatic local-polynomial search backend for each nuisance regression when degree.select != "manual". "nomad+powell" (default) performs direct search over fixed bandwidths and the degree vector using the native **crs** NOMAD C API, then applies one Powell hot start from the NOMAD solution, separately for each nuisance regression. "nomad" omits the Powell refinement. "cell" profiles the criterion over the admissible degree grid

using repeated fixed-degree bandwidth solves. NOMAD-based search supports fixed, generalized-nearest-neighbor, and adaptive-nearest-neighbor child templates; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances and restart behavior.

`ftol` tolerance on the value of the cross-validation function evaluated at located minima. Defaults to `1.19e-07 (FLT_EPSILON)`

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

`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(2,ncol(zdat))`.

`powell.remin` logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, `TRUE` restarts the local search from the located minimum. For `search.engine="nomad+powell"`, this controls only the final Powell bandwidth-polish step. The default is `TRUE` for ordinary Powell routes and `FALSE` for the Powell polish after NOMAD unless explicitly supplied.

`small` a small number, at about the precision of the data type used. Defaults to `2.22e-16 (DBL_EPSILON)`

`tol` tolerance on the position of located minima of the cross-validation function. Defaults to `1.49e-08 (sqrt(DBL_EPSILON))`

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` 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`, `bernstein.basis`, and `basis`, as described in `npregbw`.

Details

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via `newdata` does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state.

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start when that control is exposed. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval when exposed. `scale.factor.search.lower`

is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)` when both controls are present, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `npregbw` for component nonparametric regression bandwidth controls, `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

The partially linear bandwidth-selection argument surface is easiest to read by decision group: linear `xdat` inputs, nonparametric `zdat` inputs, and existing bandwidth inputs; local-polynomial/NOMAD controls for the nonparametric component; numerical search and feasibility controls; formula-interface controls; and additional bandwidth, kernel, and support controls that are passed to the component `npregbw` searches.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

`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.

When the nonparametric component is estimated with `regtype="lp"` and `degree.select != "manual"`, `npplregbw` determines the `zdat`-side degree vector and associated bandwidth coordinates separately for the nuisance regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$. With `search.engine="cell"`, the criterion is profiled over the degree grid using cached coordinate-wise or exhaustive search together with repeated fixed-degree bandwidth solves. With `search.engine="nomad"` or `"nomad+powell"`, each nuisance criterion is optimized directly over that child's degree/bandwidth space using the native **crs** NOMAD C API; `"nomad+powell"` then performs one Powell hot start for that child and keeps the better of the direct NOMAD and polished solutions. For the nonparametric regression components, this polynomial-adaptive joint-search route follows Hall and Racine (2015).

Setting `nomad=TRUE` is a convenience preset for this automatic LP route, not a generic optimizer alias. For partially linear regression it expands any missing values to the equivalent long-form call

```
npplregbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value `nomad="auto"` applies the same LP shortcut but leaves `search.engine` and `degree.select` eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit `search.engine` or `degree.select` choices are honored.

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.

Book And Method Pointers

The component regression searches inherit numerical ridging from `npregbw`; its motivation is described in `npreg`.

`npplregbw` selects the bandwidths and, for local-polynomial routes, child-specific degree/search metadata used by `npplreg`. The target model is $Y = X\beta + \Theta(Z) + \epsilon$; the bandwidth object controls the nuisance regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$ used in the final partially linear solve.

For book-length derivations, see Li and Racine (2007), Chapter 7 *Semiparametric Partially Linear Models*, especially Sections 7.1, 7.2, and 7.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*.

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(2, 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.

Hall, P. and J.S. Racine (2015), "Infinite Order Cross-Validated Local Polynomial Regression," Journal of Econometrics, 185, 510-525.

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

np.kernels, np.options, plot, plot.np 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...

if (interactive()) 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...

if (interactive()) 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"),
           B = 399,
           pivot = TRUE,
           density.weighted = TRUE,
           random.seed = 42,
           ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the model formula/data interface and explicit data inputs.

data This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

formula — a symbolic description of the quantile regression model to be tested. If xdat and ydat are omitted, the data are extracted from this formula and data.

model — a model object obtained from a call to rq. Important: the model must retain its model, design and response components, normally requested with model=TRUE. Availability is checked from those components, not the spelling of the original call.

subset — an optional vector specifying a subset of observations to be used.

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.

Bootstrap And Test Controls: These arguments control the quantile level, test statistic, bootstrap procedure, and reproducibility settings.

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.

B — an integer value specifying the number of bootstrap replications to use. Defaults to 399.

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".

density.weighted — a logical value specifying whether the statistic should be weighted by the density of xdat. Defaults to TRUE.

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.

pivot — a logical value specifying whether the statistic should be normalised such that it approaches $N(0,1)$ in distribution. Defaults to TRUE.

random.seed — an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

tau — a finite scalar in (0,1) specifying the desired τth quantile. It must equal the single quantile fitted by model; the default remains 0.5, so pass tau explicitly for other fitted quantiles.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

... 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.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

For distribution = "bootstrap", each quantile model is still refitted separately using the selected bootstrap law. The resulting score vectors are collected in bounded groups, and the fixed-design kernel contractions for I_n and, when requested, $\hat{\Omega}$ are evaluated as multi-response kernel sums. This reuses kernel geometry without forming a dense observation-by-observation weight matrix or changing score order, tau, bwydat, pivoting, density weighting, or random draws.

The selected bandwidth's effective kernel metadata are shared by the observed and bootstrap statistics. Manual bandwidth inputs and selection controls are not forwarded a second time as kernel-sum arguments.

Bootstrap refits preserve the model's observation weights. Weighted quantile models supply their raw design and response through the retained model frame; zero observation weights need not be inverted.

Model residuals, fitted values, response and design use the model's compact complete-observation sample, including when it was fitted with na.exclude. The supplied test data must represent those same complete observations in the same order. Row counts and response values are checked; when those agree, the caller remains responsible for supplying the matching covariate rows. The test does not infer a new pairing or reorder observations. The original model and its missing-value restoration policy are not modified.

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.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

`npqcmstest` tests a parametric conditional-quantile specification against a nonparametric conditional-quantile alternative at the supplied τ. The test therefore sits at the intersection of conditional CDF/quantile estimation and model specification testing.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests* and Chapter 6 *Conditional CDF and Quantile Estimation*. For the later workflow treatment of conditional CDFs and quantiles, see Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

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.

See Also

`np.kernels`, `np.options`, `plot`, `npregbw`.

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")
with(cps71, {

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

if (interactive()) 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...

if (interactive()) 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)

if (interactive()) 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)

})

## 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, ...,
       se = FALSE)


## 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,
       tol = 1.490116e-04,
       small = 1.490116e-05,
       itmax = 10000,
       ...,
       se = FALSE)

## Default S3 method:
npqreg(bws, txdat, tydat, nomad = FALSE, ...,
       se = FALSE)

## S3 method for class 'qregression'
predict(object, se.fit = FALSE, ...)

## S3 method for class 'qregression'
plot(x, ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

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.

`data` This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

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.

`txdat` a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

`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 corresponding training data recoverable from the bandwidth object in the current session.

`object` an object of class `"qregression"` returned by `npqreg`.

`x` an object of class `"qregression"` returned by `npqreg`.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when conditional-distribution bandwidths are computed inside `npqreg`.

`nomad` logical shortcut passed through to `npcdistbw` when bandwidths are computed inside `npqreg`. When `TRUE`, the conditional-distribution bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in `npcdistbw`. Additional NOMAD tuning arguments such as `nomad.nmulti` may also be supplied through `...`; `nmulti` remains the outer restart count while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect `fit$bws$nomad.shortcut` on the returned object `fit` to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the quantile regression is evaluated and which fitted quantities are returned.

`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` a logical value indicating that you want gradients of the conditional quantile with respect to the conditioning variables computed and returned in the resulting `qregression` object. Defaults to `FALSE`.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

`se.fit` logical value. If `TRUE`, `predict.qregression` returns a list with components `fit` and `se.fit`; otherwise it returns fitted conditional quantiles.

`se` a scalar logical controlling asymptotic standard-error computation. Defaults to `FALSE`. With `TRUE`, quantile standard errors are computed, and available gradient standard errors are computed when `gradients=TRUE`. Point quantiles and gradients do not depend on this switch. `se(fit)` and `gradients(fit, se=TRUE)` only extract stored results; they never initiate search or refitting. To add omitted uncertainty, use `npqreg(bws=fit$bws, tau=fit$tau, se=TRUE)` (also `gradients=TRUE` for gradient errors), with the original training/evaluation data if needed. The bandwidth object avoids repeating search. `predict(..., se.fit=TRUE)` explicitly requests its required uncertainty during the existing prediction fit.

`tau` a numeric scalar or vector specifying the quantile probability or probabilities τ. Defaults to `0.5`.

Quantile Solver Controls: These arguments control the one-dimensional numerical quantile extraction step.

`itmax` integer maximum number of iterations allowed in the one-dimensional quantile refinement. Defaults to `10000`.

`small` absolute interval-width allowance, in response units, added to the relative stopping tolerance in the one-dimensional quantile refinement. Defaults to `1.490116e-05` (approximately `1000*sqrt(.Machine$double.eps)`).

`tol` relative tolerance on the one-dimensional quantile location refinement. Refinement stops when the bracket width is at most `tol` times the initial observed response range plus `small`, or when no distinct floating-point midpoint remains. The relative scale does not depend on the response origin. When rescaling the response, rescale `small` in the same units to retain the same accuracy contract. Defaults to `1.490116e-04` (approximately `10000*sqrt(.Machine$double.eps)`).

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart, prediction/evaluation route, or plot route as appropriate.

`...` additional arguments supplied to `npcdistbw` when `npqreg` computes bandwidths internally, or arguments needed to interpret a numeric `bws` vector. This is where bandwidth selection controls such as `bwmethod`, `bwtype`, and `bwscaling`, kernel/support controls such as `cxkertype`, `cykertype`, `cxkerorder`, `cykerorder`, `cxkerbound`, and `cykerbound`, categorical kernel controls such as `uxkertype`, `uykertype`, `oxkertype`, and `oykertype`, search controls such as `nmulti` and `scale.factor.search.lower`, and local-polynomial/NOMAD controls such as `regtype`, `degree`, `bernstein.basis`, `degree.select`, and `nomad.nmulti` are supplied. In `predict.qregression`, additional arguments are passed to `npqreg` for evaluation with the stored bandwidth object; common examples are `newdata`, native `exdat`, and `tau`. In `plot.qregression`, additional arguments are passed through the package plot route; common controls include `tau`, `gradients`, `output`, `legend`, and graphics arguments. See `npcdistbw` and `plot.np` for the complete bandwidth-selection and plotting argument surfaces.

Details

Conditional fitting and replay use the literal categorical kernels for both explanatory and response variables. Near-upper categorical bandwidths are not replaced by approximate constant factors. The same target is used for levels, gradients, standard errors, conditional hats, and reconstructed plot uncertainty. This conditional-family policy does not change direct regression or unconditional kernel-sum evaluation and does not modify the caller's `np.largelambda` option.

With a formula-derived bandwidth object, supplying only `tydat` replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective `na.action` is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use `data` for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named `newdata` columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over `newdata`.

For a native bandwidth object, `newdata` supplies `exdat`; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

Formula variables of class `ts`, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Local-polynomial calculations inherit numerical ridging from `npcdist`. See that estimator's help for the solve policy and its distinct conditional uncertainty formula.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

Given a conditional distribution bandwidth object, `npqreg` estimates the conditional distribution function $F(y|x)$ and extracts the requested conditional quantile. For $0 < \tau < 1$, the conditional quantile at probability τ is

$$q_\tau(x) = \inf\{y : F(y|x) \geq \tau\}.$$

Equivalently, $q_\tau(x)$ is a quasi-inverse of the conditional distribution in the sense of Nelsen (2006): an inverse agrees with F on the range of F, while outside that range the generalized inverse is defined by the lower endpoint at which F reaches or exceeds the requested probability. Numerically, `npqreg` inverts the selected conditional distribution estimator represented by `bws`. This includes the selected bandwidth type, kernels, local-polynomial regression type, selected polynomial degree, basis, and Bernstein-basis setting inherited from `npcdistbw`. If the bandwidth object was selected with `nomad=TRUE`, the returned conditional-distribution bandwidth object is an LP object: its `regtype`/`regtype.engine` metadata identify the selected local-polynomial route and its `degree`/`degree.engine` metadata record the selected continuous-coordinate polynomial degree. `npqreg`, `predict`, and `plot` reuse this stored LP metadata; plotting additional `tau` values does not recompute or downgrade the selected degree. The one-dimensional inversion is carried out over the observed support of the dependent variable using the same selected conditional CDF estimator that is later used for quantile standard errors and gradients. The arguments `tol`, `small`, and `itmax` control this one-dimensional refinement. Conditional quantile inversion requires a monotone dependent-variable conditional distribution estimate; therefore `npqreg` supports `cykerorder`

= 2 for this route and rejects higher-order dependent-variable CDF kernels, which can be nonmonotone.

During this inversion, an external predictor row identified by the selected CDF as having no kernel support returns NA for its quantiles and requested standard errors and gradients. Supported rows retain their original order, including repeated evaluation points and vector-tau requests. An unavailable categorical counterfactual affects only the corresponding contrast, not an otherwise supported base quantile. One warning summarizes affected external rows after the public evaluation or plot call completes. Other numerical failures, and required training or bootstrap evaluations, retain their strict error behavior.

The existing point-only shortcut for an exactly constant response is unchanged: it returns that response value without evaluating the conditional CDF. External predictor support is not checked by this shortcut; requests for standard errors or gradients still enter the conditional evaluation machinery.

Let $f(y|x) = \partial F(y|x)/\partial y$ denote the conditional density. The asymptotic standard error of the conditional quantile is computed by the first-order delta method,

$$se\{\hat{q}_\tau(x)\} = \frac{se\{\hat{F}(\hat{q}_\tau(x)|x)\}}{\hat{f}(\hat{q}_\tau(x)|x)},$$

using the selected conditional distribution standard-error machinery and the selected conditional density evaluated at the fitted quantile. This corresponds to the quantile variance expression in Li, Lin and Racine (2013).

Conditional-quantile plots support first derivatives only. The default and explicit gradient.order=1 retain that target; other derivative orders are rejected before plot fitting or drawing. Higher polynomial degrees in the fitted bandwidth object do not add higher-order quantile derivatives. This restriction does not apply to npplsqreg or conditional-distribution gradient plots.

A scalar quantile can be displayed as a surface by explicitly supplying perspective=TRUE when there are two continuous or numeric-labelled ordered conditioning variables. Omitting perspective, or supplying perspective=FALSE, retains the usual slice layout. Vector quantiles and gradient plots also retain their slice layout. Surface uncertainty is calibrated over the complete displayed grid for that one tau. Both the base renderer and renderer="rgl", view="fixed" support level surfaces and their interval meshes. Returned evaluation rows retain the trained predictor names and ordered-factor support; numeric axis coordinates are used only for drawing. The data overlay can be disabled with data.overlay=FALSE.

Local-polynomial conditional-response factors use local response moments as a working covariance approximation conditional on the accepted design and response map. At fixed nonshrinking bandwidths they need not equal the full empirical-caseweight derivative, which also perturbs the design map. Their leading interpretation requires smooth shrinking neighborhoods and nondegenerate local response variation.

Asymptotic plot(..., band="simultaneous") calibrates a joint Gaussian maximum over the retained grid within one panel and one tau; calibration is not joint across quantiles or panels. Levels use inverse-CDF influence rows at their own fitted quantiles. Continuous gradients use the selected CDF-derivative influence and the existing leading plug-in standard error. Positive-degree categorical effects use paired own-quantile endpoint influences with shared observation columns. Local-constant and degree-zero categorical effects retain their different public target, the same-root CDF contrast divided by the selected density: their joint bands include density and root-evaluation terms in a full local delta calculation, with separately computed joint-band standard errors. band="all"

retains existing pointwise and Bonferroni widths even when the simultaneous calculation uses this different scale.

Local-constant and local-polynomial routes retain the fitted kernel, bandwidth type, degree and basis. Quantile inversion still requires a second-order response kernel. A valid joint grid requires finite positive slopes, interior unclamped quantiles, negligible inversion residuals, identifiable local influence and regular kernel/support geometry. The nearest-neighbor same-root calculation holds the realized neighbor counts and geometry fixed in its empirical-mass approximation, while retaining movement of a generalized-neighbor query radius when differentiating in the response coordinate. It does not claim an unconditional law for estimating the counts or radii.

These are selected-smoothing variability approximations, not smoothing-bias corrections or continuum confidence bands. An asymptotic coverage interpretation additionally requires independent observations, consistent local covariance and marginal scales, negligible smoothing/ridge bias and non-dominating influence rows. The continuous-gradient leading approximation requires shrinking explanatory bandwidth h_x, with $h_x^2/h_y \longrightarrow 0$, and sufficient local derivative sample size; fixed nonshrinking smoothing retains a working approximation. A growing or data-dependent grid needs a corresponding uniform linearization and Gaussian-maximum approximation. Unsupported or nonregular grids receive a specific unavailable-band notice. `gaussian.draws` controls simulation precision and `random.seed` restores the caller's RNG state.

If `gradients=TRUE`, `npqreg` also computes gradients of the conditional quantile with respect to the conditioning variables for which gradients are defined. Differentiating $F(q_\tau(x)|x) = \tau$ gives

$$\nabla_x q_\tau(x) = -\frac{F_x(q_\tau(x)|x)}{f(q_\tau(x)|x)},$$

where $F_x(y|x)$ is the derivative of the same selected conditional distribution estimator with respect to x. For `regtype="lc"`, this uses the local-constant conditional-gradient machinery; for `regtype="ll"` it uses the canonical local-polynomial degree-one route; and for `regtype="lp"` it uses the selected or supplied degree vector. The corresponding first-order gradient standard errors are computed componentwise as

$$se\{\nabla_x \hat{q}_\tau(x)\} = \frac{se\{\hat{F}_x(\hat{q}_\tau(x)|x)\}}{\hat{f}(\hat{q}_\tau(x)|x)}.$$

This is a plug-in estimate of the population implicit-CDF identity, using the selected density in the denominator. It is not in general the exact derivative of the fitted finite-smoothing quantile curve. In particular, bounded and beta response kernels use observation-centred CDFs and target-centred densities, so numerical differentiation of the fitted quantiles need not reproduce the reported plug-in gradients. The same selected-density convention is used for the delta-method standard errors.

For categorical conditioning variables in positive-degree local-polynomial fits (including local-linear fits), first-difference standard errors retain the paired covariance between the two category endpoints. Each endpoint uses its own fitted quantile and positive conditional-density slope in the implicit-CDF calculation. The selected basis, smoothing parameters and realized neighbour radii are held fixed. This requested-only calculation supports fixed, generalized nearest-neighbour and adaptive nearest-neighbour bandwidths; a nonpositive density slope or nonsmooth/nonfinite influence produces `NA` uncertainty with a notice.

For continuous conditioning variables, these gradient standard errors are componentwise leading asymptotic approximations. They propagate the conditional CDF-gradient uncertainty but do not

jointly propagate fluctuations of the density denominator and estimated quantile location. Their use requires appropriate smoothness, positive-density and relative bandwidth-rate conditions. They are not a full finite-smoothing covariance or a joint covariance across coordinates or tau values.

In a heterogeneous fitted local-polynomial degree vector, a degree-zero coordinate has no available first-derivative estimate and is returned as NA; the established all-degree-zero compatibility route is unchanged. Its plot panel remains present but empty when another derivative or categorical first-difference effect is available; a request with no available component is rejected.

When npqreg is called without an explicit bws object, it first computes conditional distribution bandwidths using npcdistbw and stores them in the returned object's bws component. If a scalar tau was used initially and additional quantiles are later desired as fitted objects, reuse those selected bandwidths directly, for example npqreg(bws = fit$bws, tau = c(0.25, 0.5, 0.75)). If the goal is only to inspect additional quantiles graphically, use plot(fit, tau = c(0.25, 0.5, 0.75)); this reuses the stored bandwidths and recomputes only the one-dimensional quantile extraction step for the requested tau values. Vector-tau plots are overlaid and include a legend; use legend=FALSE, legend=NULL, or a legend=list(...) control to suppress or customize it.

The predict method follows the usual S3 newdata convention. For formula fits, supply a data frame of evaluation covariates via predict(fit, newdata=...). For non-formula fits, newdata is translated to the native evaluator argument exdat when exdat is not supplied. The native exdat argument remains available for advanced workflows, contains already transformed predictor coordinates for formula fits, and takes precedence if both newdata and exdat are supplied. If tau is omitted in predict, the fitted object's stored tau value is used.

Value

npqreg returns a qregression object. The generic functions fitted (or quantile), se, predict, and gradients extract stored estimated values, asymptotic standard errors, and gradients, or compute predictions from the retained bandwidth object. predict uses the object's stored tau value by default; supply tau= to override it. 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. If tau has length greater than one this is an evaluation-by-tau matrix.
quanterr	asymptotic standard errors of the quantile regression estimates, obtained from the conditional distribution standard error and the estimated conditional density at the fitted quantile. If tau has length greater than one this is an evaluation-by-tau matrix. This component is NULL when se=FALSE.
quantgrad	gradients of the conditional quantile with respect to the conditioning variables at each evaluation point, when gradients=TRUE. If tau has length greater than one this is an evaluation-by-gradient-by-tau array.
quantgerr	asymptotic standard errors for gradients, when gradients=TRUE and se=TRUE. If tau has length greater than one this is an evaluation-by-gradient-by-tau array. This component is NULL when se=FALSE.
se	whether asymptotic uncertainty was requested
tau	the quantile probability or probabilities computed

`fit.controls` resolved `tol`, `small`, and `itmax` controls used for numerical quantile extraction. Prediction and quantile plots (including bootstrap fits and centers) reuse these values unless the corresponding argument is explicitly supplied. Older objects without this component use the historical defaults.

Book And Method Pointers

The conditional quantile target is the generalized inverse $q_\tau(x) = \inf\{y : F(y \mid x) \geq \tau\}$ of the conditional distribution. The standard errors and gradients described above are first-order delta-method quantities evaluated using the same selected conditional CDF, conditional density, bandwidths, kernels, and local-polynomial degree inherited from the supplied `npcdistbw` object.

For book-length derivations, see Li and Racine (2007), Chapter 6 *Conditional CDF and Quantile Estimation*, especially Sections 6.3-6.5, and Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*. The quasi-inverse terminology follows Nelsen (2006).

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

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.

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

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

See Also

np.kernels, np.options, plot, plot.np, **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")
with(Italy, {

# Compute conditional distribution bandwidths and extract three
# conditional quantiles using the same selected bandwidths.

model.q <- npqreg(gdp~ordered(year), tau=c(0.25, 0.50, 0.75))

# Plot the overlaid quantiles.

plot(model.q)

# If a scalar tau was used first, additional quantiles can reuse the
# selected bandwidths without recomputing cross-validation. Use npqreg()
# when the additional fitted values are needed as an object, or plot()
# when graphical inspection is all that is desired.

model.med <- npqreg(gdp~ordered(year), tau=0.50)
model.q <- npqreg(bws=model.med$bws, tau=c(0.25, 0.50, 0.75))
plot(model.med, tau=c(0.25, 0.50, 0.75))

})

# 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")
with(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.q <- npqreg(bws=bw, tau=c(0.25, 0.50, 0.75))

plot(model.q)

})

## 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

Data, Bandwidth Inputs And Formula Interface: These arguments identify the distribution object, data, and bandwidth controls used for quantile extraction.

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 ....
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).

Evaluation And Quantile Controls: These arguments control the target quantile level, evaluation size, and distribution interpolation.

`f`	an optional argument fed to `extendrange`. Defaults to `1`. See `?extendrange` for details.
`num.eval`	an optional integer specifying the length of the grid on which the quasi-inverse is computed. Defaults to `10000`.
`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)`.

Additional Arguments: Further arguments are passed to bandwidth or distribution routines as needed.

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

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

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 bounded kernels such as ckertype="epanechnikov". The default options(np.tree="auto") uses kd-trees for eligible bounded-kernel continuous routes; options(np.tree=TRUE) remains available to force kd-tree use explicitly.

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

np.kernels, np.options, plot

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,
      se = FALSE,
      ...)

## Default S3 method:
npreg(bws,
      txdat,
      tydat,
      nomad = FALSE,
      se = FALSE,
      ...)

## S3 method for class 'rbandwidth'
npreg(bws,
      txdat = stop("training data 'txdat' missing"),
      tydat = stop("training data 'tydat' missing"),
      exdat,
      eydat,
      gradient.order = 1L,
      gradients = FALSE,
      residuals = FALSE,
      se = FALSE,
      ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

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.

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

txdat — a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

tydat — a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of txdat. A factor response is also supported when the bandwidth object retains the fitted factor-response coding. A bandwidth object fitted to a numeric response cannot supply that coding and is rejected when reused with a factor response. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside npreg.

nomad — logical shortcut passed through to npregbw when bandwidths are computed inside npreg. When TRUE, the regression bandwidth route fills any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npregbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the regression is evaluated and which fitted quantities are returned.

se — a logical value indicating whether the residual-normalized sandwich standard errors described below are computed and retained. Defaults to FALSE. With se

= FALSE, fitted values and any requested gradients are still computed, but the returned merr and gerr components are NULL; se() and gradients(..., se = TRUE) then fail with a message directing the user to refit or re-evaluate with se = TRUE. This option controls fit/evaluation output only and is not forwarded to bandwidth selection.

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. When evaluation predictors are supplied without evaluation outcomes, goodness-of-fit scores (including R2 and MSE) are unavailable and reported as NA; no additional training fit is performed for these diagnostics.

gradient.order — for regtype="lp" with gradients=TRUE, a positive integer (or integer vector with one entry per continuous predictor) specifying derivative order(s). Defaults to 1L. Orders exceeding degree for a variable are returned as NA, apart from the established all-degree-zero first-derivative compatibility route. Higher-order derivatives are available for continuous predictors when the requested order does not exceed the corresponding local-polynomial degree.

gradients — a logical value indicating that you want gradients computed and returned in the resulting npregression object. Defaults to FALSE. For regtype="lp", derivative components that are not defined (requested order exceeds the variable-specific polynomial degree) are returned as NA. For ordered and unordered predictors the gradient columns are first-order effects/contrasts where supported; gradient.order controls derivative order for continuous predictors. Plotting retains one panel per requested predictor: an unavailable continuous derivative is shown as an empty, correctly labelled panel when another continuous derivative or categorical effect is available. A request for which no component is available is rejected with the affected orders and degrees reported.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

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

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.

Additional Arguments: Further arguments are passed to npregbw when bandwidths are computed internally, or used to interpret a numeric bws vector.

... — additional arguments supplied to npregbw when npreg computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth selection controls such as bwmethod, bwtype, bwscaling, kernel/support controls such as ckertype, ckerorder, and ckerbound, categorical kernel controls such as ukertype and okertype, search controls such as nmulti and scale.factor.search.lower, and local-polynomial/NOMAD controls such as regtype, degree, basis, bernstein.basis, degree.select, and nomad.nmulti are supplied. See npregbw for the complete bandwidth-selection argument surface.

Details

With generalized nearest-neighbor bandwidths, query identity matters when selecting a radius: a training observation excludes its own occurrence, whereas explicit evaluation data remain external queries even if their values equal training observations. Consequently, `predict(fit, newdata = training.data)` can differ from `fitted(fit)`. This does not omit the training contribution from the fitted estimator. See `np.kernels` for the radius convention.

With a formula-derived bandwidth object, supplying only `tydat` replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective `na.action` is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use `data` for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named `newdata` columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over `newdata`.

For a native bandwidth object, `newdata` supplies `exdat`; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

Documentation guide: see `npregbw` for bandwidth selection and search controls, `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

When `bws` is omitted, the formula and default methods call `npregbw` first and pass bandwidth-selection arguments from `...` to that call. When `bws` is already an `rbandwidth` object, `npreg` estimates with the stored bandwidth metadata in that object.

Argument groups for bandwidth selection are documented on `npregbw`. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls, and finally local-polynomial/NOMAD controls when using polynomial-adaptive fits.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

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. Beta associated kernels of orders 2, 4, 6, and 8 are also available for local-constant and local-polynomial regression. 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.

For `ckertype="beta"`, suppose continuous predictor j has declared finite support $[a_j, b_j]$, width $L_j = b_j - a_j$, public metric bandwidth h_j, evaluation coordinate x_j, and observation X_{ij}. Define

$$t_j = (x_j - a_j)/L_j, \quad z_{ij} = (X_{ij} - a_j)/L_j, \quad \tau_j = (L_j/h_j)^2.$$

The local-constant product weight is

$$K_i(x) = \prod_j \left\{ \frac{1}{L_j} \frac{z_{ij}^{t_j\tau_j}(1 - z_{ij})^{(1-t_j)\tau_j}}{B(1 + t_j\tau_j, 1 + (1 - t_j)\tau_j)} \right\},$$

and the fitted conditional mean is the Nadaraya–Watson ratio

$$\widehat{m}(x) = \frac{\sum_i K_i(x) Y_i}{\sum_i K_i(x)}.$$

This uses Chen's evaluation-centred beta associated-kernel contribution inside the established **np** local-constant ratio. It should not be confused with the Gasser–Muller beta regression smoother studied by Chen (2000). The support scale is already present in each contribution, and the implementation applies neither the legacy $1/h_j$ factor nor the generic bounded-kernel normalizer. Although that normalizer cancels from ordinary local-constant numerator/denominator ratios, the beta kernel is not merely such a correction: its shape parameters change with the evaluation point.

At orders 4, 6, and 8, each displayed order-2 coordinate factor is replaced by the signed Igarashi–Kakizawa multiscale combination documented in np.kernels. The resulting product weights and denominator may be signed; no absolute-value, order-2 substitution, or positivity repair is applied.

Products and ratios are evaluated in the log domain, so representable beta regression results are retained even when every unscaled product would underflow to zero. In the local-constant case, $\ell_i(x) = K_i(x)/\sum_r K_r(x)$. The reported standard error uses the residual-normalized sandwich formula below, also for beta kernels; it is not a local residual-variance estimate multiplied by an effective-sample factor.

The beta regression route requires ckerorder in c(2,4,6,8) and finite fixed or sample-range bounds for each continuous predictor. It supports local-constant, local-linear and local-polynomial fits, including mixed continuous and categorical predictors. Local-constant first derivatives requested with gradients=TRUE are analytic C implementations; their standard errors use the derivative of the normalized weights. At exact support endpoints, normalized numerator/denominator jumps are cancelled first. A genuine estimator jump is returned as a signed infinity with gerr=NA; an indeterminate numerical cancellation is returned as NA with a warning. This extended-real gradient is retained in fitted and plot-data objects. Plot methods omit the non-finite coordinate only from their temporary drawing geometry, so an endpoint singularity breaks the plotted line without changing the returned estimate. Manual and cross-validated bandwidth objects support fixed, generalized-NN, and adaptive-NN modes. An evaluation point with no supported observation weight also produces an explicit error.

Standard Errors: Ordinary regression standard errors use a residual-normalized sandwich for the actual fitted linear map, conditional on the selected bandwidths, polynomial specification and realized predictor geometry. Let S map training responses to their fitted values, let $e = (I - S)Y$, and define

$$q_i = \sum_j (\delta_{ij} - S_{ij})^2.$$

For $q_i > 0$, the normalized training residual is $z_i = e_i/\sqrt{q_i}$. This is the pre-smoothing residual normalization in Ruppert, Wand, Holst, and Hössjer (1997, Section 1.1, equation (3)), not their separate variance-function smoothing procedure. If $\ell_i(x)$ is the influence of response Y_i on the fitted mean at x, the reported standard error is

$$\mathrm{se}\{\widehat{m}(x)\} = \left\{ \sum_i \ell_i(x)^2 z_i^2 \right\}^{1/2}.$$

A derivative uses its corresponding influence row. A categorical finite difference uses the difference between the two endpoint influence rows, retaining their covariance. Its estimate and error use the same paired influence calculation rather than subtracting nearly equal fitted values. Requesting or omitting standard errors does not change the computation of a requested categorical effect.

For generalized nearest-neighbor bandwidths, categorical effects at fitted training rows retain that row's training identity when selecting the radius for both endpoints, for all regression types, polynomial degrees and bases. The identity is excluded only from radius selection, not from the fitted response weights. Explicit evaluation data use external-query radii, even when their coordinates equal training observations. Consequently fitted-row and explicitly evaluated effects can differ under this bandwidth type.

The training smoother and influence rows use the actual fitted basis and accepted regularized solve where needed. The residual-row norm is not generally $1 - S_{ii}$; that identity requires a symmetric,

idempotent projection. Nor is q_i necessarily at most one. No additional degrees-of-freedom multiplier or positive variance floor is applied. The corresponding influence representation is used without retaining a full training-by-training matrix.

Treating the smoother as fixed, under independent homoskedastic errors and a mean reproduced by that smoother, $E(e_i^2 \mid X) = \sigma^2 q_i$. The normalization corrects fitted residual noise contraction in that setting, not smoothing bias. With a general mean, squared residual bias also contributes. Under heteroskedasticity, e_i^2/q_i mixes donor variances and need not be unbiased for the variance of observation i. Finite-sample calibration under arbitrary bias, heteroskedasticity or model selection is not guaranteed.

A zero residual-row norm leaves no residual information for that donor. Requested errors depending on such unidentified information are `NA`, with a warning; finite point estimates remain available. A separately certified zero influence does not require that donor's variance. Small positive residual norms are not replaced by zero or by a numerical variance floor.

This convention applies to fixed, generalized-NN and adaptive-NN bandwidths and beta kernels on supported regression routes. External evaluation errors use the training smoother's normalized residuals, not residuals centred on each external evaluation mean. Bandwidth- and degree-selection uncertainty is not incorporated. `se()`, derivative-error accessors, prediction standard errors and asymptotic plot intervals consume the corresponding retained errors. Pivotal `npsigtest` statistics use these gradient errors, recomputed from each bootstrap response when bootstrapping.

Numerical Ridging: Local-linear and local-polynomial moment solves use numerical ridging when needed, motivated by Seifert and Gasser (2000). The package uses bounded implementation-specific ridge policies, not that paper's complete data-adaptive ridge selector. Ridging stabilizes the fit; it is distinct from the residual normalization above, which uses the actual accepted smoother, including any ridge adjustment. A local system with fewer nonzero-weight donor rows than basis coefficients is structurally rank deficient and uses this same ridge policy for fitting and related hat operators. Bootstrap multiplicities do not create additional independent donor rows. This check does not assert full rank when the donor count reaches the basis dimension.

When bandwidths are obtained with `regtype="lp"`, C-level `npreg` supports heterogeneous continuous polynomial degrees via `degree`. The `basis` selector currently supports `basis="glp"`, `"additive"`, and `"tensor"`. For continuous predictors with degree vector d, additive basis size is $1 + \sum_j d_j$, tensor basis size is $\prod_j (d_j + 1)$, and GLP uses admissible multi-indices α with $\alpha_j \le d_j$ and $0 < \sum_j \alpha_j \le \max_j d_j$ plus an intercept. Thus, for $d = (2, 1)$, GLP contains $1, x_1, x_1^2, x_2, x_1 x_2$; the additive basis omits $x_1 x_2$, while the tensor basis additionally contains $x_1^2 x_2$.

The optional flag `bernstein.basis` controls basis construction: `FALSE` (default) uses raw local-polynomial powers. These can be numerically unstable, particularly above degree one or when predictor offsets are large relative to their spread. The option `bernstein.basis=TRUE` is provided to improve numerical conditioning; it does not remedy genuine collinearity. Centring/scaling continuous predictors is another explicit user choice. Existing conditioning warnings and errors assess the selected basis; numerical rank deficiency need not imply genuinely redundant predictors. No automatic basis switch is made. For `basis="glp"`, the historical selector `TRUE` uses a degree-graded, orthonormal shifted-Legendre representation. On the unit-scaled support $u \in [0, 1]$, its order-k factor is $\sqrt{2k+1} P_k(2u - 1)$, where P_k is a Legendre polynomial. This is an exact degree-k linear combination of Bernstein polynomials and, because the GLP multi-index set is downward closed, spans exactly the same heterogeneous polynomial space as the raw representation. It is evaluated by a deterministic Legendre recurrence, not by sample-dependent Gram–Schmidt or QR orthogonalization. Additive and tensor bases retain their fixed-coordinate-degree Bernstein representation. The

homogeneous degree-0 and degree-1 cases remain equivalent to `lc` and `ll`, respectively. For continuous predictors, `gradients=TRUE` returns the derivative order(s) requested by `gradient.order`; the associated `gerr` entries are standard errors for those same derivative orders. Both raw and shifted-Legendre representations report fitted values and derivatives on the original response/predictor scale. In mixed-data GLP settings, unordered and ordered predictors contribute first-order effects/contrasts rather than higher-order derivatives. When `npregbw(..., regtype="lp")` is used with `degree.select="manual"`, the degree vector remains fixed user input. When `degree.select != "manual"`, `npregbw` can jointly select polynomial degree and bandwidth using either the cached cell-search backend or the direct `search.engine="nomad"/"nomad+powell"` route described in `npregbw`; the latter follows Hall and Racine (2015). For practitioners who want that recommended route without spelling out all LP tuning arguments, `npreg(..., nomad=TRUE)` and `npregbw(..., nomad=TRUE)` expand missing settings to the same documented automatic-LP NOMAD preset. Explicit incompatible settings fail fast rather than being silently rewritten. The direct NOMAD backend is provided by the suggested package `crs`, so install `crs` before using `search.engine="nomad"`, `"nomad+powell"`, or `nomad=TRUE`. For `bernstein.basis=TRUE`, evaluation points for continuous predictors must lie within training support; use `bernstein.basis=FALSE` for extrapolation. For `regtype="ll"` and `regtype="lp"`, the training continuous design is checked for rank deficiency and extreme condition number before estimation proceeds.

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:

With `se.fit=TRUE`, `predict()` returns `fit`, `se.fit`, `df`, and `residual.scale`. The last field uses training MSE in its historical units, not its square root or an evaluation MSE. It reuses a known training fit's MSE without an additional fit. It is `NA` when training MSE is unavailable, including an object created at external evaluation points or scored against explicit `eydat`, and external predictions with explicit training-data or model-frame overrides. A prediction evaluated on replacement training data can use that new training fit's MSE. Evaluation outcomes do not substitute for training residual information.

`eval`	evaluation points
`mean`	estimates of the regression function (conditional mean) at the evaluation points. An evaluation row whose required kernel normalization is exactly zero or non-finite is returned as `NA`; other rows remain available.
`merr`	standard errors of the regression function estimates, or `NULL` when `se = FALSE`. Undefined normalization rows are returned as `NA`. Requested errors depending on unidentified training residual information are also `NA`; finite point estimates are retained. See the Standard Errors subsection.
`grad`	estimates of the gradients at each evaluation point. Undefined normalization rows are returned as `NA`.

gerr	standard errors of the gradient estimates, or NULL when se = FALSE. Undefined normalization rows are returned as NA. Requested errors depending on unidentified training residual information are also NA, separately from certified zero effects. See the Standard Errors subsection.
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

Book And Method Pointers

The regression target is the conditional mean $m(x) = E[Y \mid X = x]$. Local-constant estimation can be written schematically as $\hat{m}(x) = \sum_i W_i(x) Y_i$, where the weights are normalized generalized product-kernel weights over the continuous, unordered, and ordered components of X. Local-polynomial regression replaces this scalar weighted average by a weighted least squares fit in powers $b(X_i)$ of the original continuous coordinates, with kernel weights centered at x. If $\widehat{\beta}_x$ denotes that local weighted fit, the reported level and derivative are respectively $b(x)^T \widehat{\beta}_x$ and $D^\gamma b(x)^T \widehat{\beta}_x$. For the downward-closed GLP space this uncentered representation is numerically equivalent, up to floating-point solution paths, to fitting powers of $X_i - x$ and reading the corresponding intercept and derivative coefficients.

For book-length derivations, see Li and Racine (2007), Chapter 2 *Regression*, especially Sections 2.1, 2.2, 2.4, and 2.5, and Chapter 4 *Kernel Estimation with Mixed Data*, especially Sections 4.2 and 4.4. The later workflow treatment is Racine (2019), Chapter 6 *Conditional Mean Function Estimation*, including the local polynomial and mixed-data marginal-effect discussion.

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.

For formula fits, character-valued missing-data policies are resolved through the stats model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit na.action, including NULL, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as stats::model.frame.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after saveRDS() and readRDS(). Custom formula transformations used on new evaluation data must still be available. With explicit replacement training data, an explicit na.action overrides the retained policy, including NULL; any incomplete rows subsequently

excluded by native fitting are recorded in the effective training sample. User formula and custom-function environments remain available and can retain unrelated caller objects in serialized files. See npregbw for storage and missing-value policy details.

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.

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

Chen, S.X. (2000), "Beta kernel smoothers for regression curves," Statistica Sinica, 10, 73-91.

Igarashi, G. and Kakizawa, Y. (2020), "Higher-order bias corrections for asymmetric kernel density estimators," Journal of Nonparametric Statistics, 32(3), 716–752. doi:10.1080/10485252.2020.1770754.

Cohen, A. and G. Migliorati (2018), "Multivariate approximation in downward closed polynomial spaces," in R. Cools and D. Nuyens (eds.), *Contemporary Computational Mathematics*, Springer, 233-282.

Dung, D. and T. Ullrich (2011), "Whitney type inequalities for local anisotropic polynomial approximation," Journal of Approximation Theory, 163, 1590-1605.

Fan, J. and I. Gijbels (1996), *Local Polynomial Modelling and Its Applications,* Chapman and Hall.

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.

Hall, P. and J.S. Racine (2015), "Infinite Order Cross-Validated Local Polynomial Regression," Journal of Econometrics, 185, 510-525.

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.

Ruppert, D. and M.P. Wand (1994), "Multivariate locally weighted least squares regression," The Annals of Statistics, 22, 1346-1370.

Ruppert, D., M.P. Wand, U. Holst, and O. Hössjer (1997), "Local Polynomial Variance-Function Estimation," Technometrics, 39(3), 262-273. doi:10.1080/00401706.1997.10485117.

Seifert, B. and T. Gasser (2000), "Data Adaptive Ridging in Local Polynomial Regression," Journal of Computational and Graphical Statistics, 9(2), 338-360. doi:10.1080/10618600.2000.10474884.

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

See Also

np.kernels, np.options, plot, plot.np 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")
with(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
# errors="asymptotic" the figure will instantly appear).

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

})

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

if (interactive()) 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")
with(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
# errors="asymptotic" the figure will instantly appear).

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

})

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

if (interactive()) 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", package = "np")

bw <- npregbw(logwage~age, regtype="ll", bwmethod="cv.aic", data = cps71)

# Next, plot the regression function...

if (interactive()) plot(bw, errors="asymptotic")
with(cps71, points(age, logwage, cex=.2, col="red"))

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

if (interactive()) Sys.sleep(5)

# Next, plot the derivative...

if (interactive()) plot(bw, errors="asymptotic", gradient=TRUE)

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

if (interactive()) 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", package = "np")

bw <- npregbw(xdat=cps71$age, ydat=cps71$logwage, regtype="ll", bwmethod="cv.aic")

# Next, plot the regression function...

if (interactive()) plot(bw, errors="asymptotic")
with(cps71, points(age, logwage, cex=.2, col="red"))

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

if (interactive()) Sys.sleep(5)

# Next, plot the derivative...

if (interactive()) plot(bw, errors="asymptotic", gradient=TRUE)

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

if (interactive()) 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")
with(oecdpanel, {

bw <- npregbw(formula=growth~
              factor(oecd)+
              factor(year)+
              initgdp+
              popgro+
              inv+
              humancap)

if (interactive()) plot(bw, errors="asymptotic")

})

# 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")
with(oecdpanel, {

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

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

if (interactive()) plot(bw, errors="asymptotic")

})


# 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")
with(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

if (interactive()) plot(bw, errors="bootstrap", band="simultaneous")

})

# 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")
with(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

if (interactive()) plot(bw, errors="bootstrap", band="simultaneous")

})

# 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,
        ...)
```

```
## Default S3 method:
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        bandwidth.compute = TRUE,
        basis,
        bernstein.basis,
        bwmethod,
        bwscaling,
        bwtype,
        cfac.dir,
        scale.factor.init,
        ckerbound,
        ckerlb,
        ckerorder,
        ckertype,
        ckerub,
        degree,
        degree.select = c("manual", "coordinate", "exhaustive"),
        search.engine = c("nomad+powell", "cell", "nomad"),
        nomad = FALSE,
        nomad.nmulti = 0L,
        degree.min = NULL,
        degree.max = NULL,
        degree.start = NULL,
        degree.restarts = 0L,
        degree.max.cycles = 20L,
        degree.verify = FALSE,
        dfac.dir,
        dfac.init,
        dfc.dir,
        ftol,
        scale.factor.init.upper,
        hbd.dir,
        hbd.init,
        initc.dir,
        initd.dir,
        invalid.penalty = c("baseline","dbmax"),
        itmax,
        lbc.dir,
        scale.factor.init.lower,
        lbd.dir,
        lbd.init,
        nmulti,
        okertype,
        penalty.multiplier = 10,
        regtype,
```

```
          nomad.remin = FALSE,
          powell.remin,
          bwsolver = c("powell", "mads", "mads+powell"),
          scale.init.categorical.sample,
          scale.factor.search.lower = NULL,
          small,
          tol,
          transform.bounds = FALSE,
          ukertype,
          ...,
          nomad.opts = list())

  ## S3 method for class 'rbandwidth'
  npregbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          bws,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
          scale.factor.init.lower = 0.1,
          lbd.dir = 0.1,
          lbd.init = 0.1,
          nmulti,
          penalty.multiplier = 10,
          powell.remin = TRUE,
          bwsolver = c("powell", "mads", "mads+powell"),
          scale.init.categorical.sample = FALSE,
          scale.factor.search.lower = NULL,
          small = 1.490116e-05,
          tol = 1.490116e-04,
          transform.bounds = FALSE,
          ...,
          nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, and whether bandwidths are supplied or computed.

`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`. The beta associated-kernel route supports both automatic cross-validation and explicit bandwidth objects.

`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.

`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.

`data`
: This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

  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.

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

`na.action`
: a function which indicates what should happen when the data contain `NA`s. 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`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

`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`.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search when `regtype="lp"`.

`degree.max`
: optional scalar or integer vector giving upper bounds for automatic degree search when `degree.select != "manual"`. If scalar, the value is recycled over continuous predictors.

degree.max.cycles
: positive integer giving the maximum number of coordinate-search sweeps over the continuous-predictor degree vector. Ignored for degree.select="manual" and "exhaustive".

degree.min
: optional scalar or integer vector giving lower bounds for automatic degree search when degree.select != "manual". If scalar, the value is recycled over continuous predictors.

degree.restarts
: non-negative integer giving the number of additional deterministic restarts used by coordinate search. Ignored for degree.select="manual" and "exhaustive".

degree.select
: character string controlling local-polynomial degree handling when regtype="lp". "manual" (default) treats degree as fixed. "coordinate" performs cached coordinate-wise search over admissible degree vectors. "exhaustive" evaluates the full admissible degree grid when search.engine="cell". For NOMAD-based search engines, any non-"manual" value requests direct joint search over degree and bandwidth coordinates.

degree.start
: optional starting degree vector for automatic degree search when degree.select="coordinate". If omitted, cell-based search starts from the degree-zero local-constant baseline on the continuous predictors, while NOMAD-based search starts from a clipped degree-one vector on the searchable continuous predictors. For NOMAD multi-starts, later restart starts are generated reproducibly from a conservative proposal box and screened using dim_basis() so that the initial basis dimension remains well below the training-sample limit. This avoids wasting starts on flat penalty or heavily ridged designs while leaving the full user requested degree search region unchanged.

degree.verify
: logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for search.engine="cell".

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

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

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

dfac.dir stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

dfac.init non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

hbd.dir upper bound for direction set search for Powell's algorithm for categorical variables. See Details

hbd.init upper bound for scale factors for categorical variables for Powell's algorithm. See Details

initd.dir initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbd.dir lower bound for direction set search for Powell's algorithm for categorical variables. See Details

lbd.init lower bound 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

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

cfac.dir stretch factor for direction set search for Powell's algorithm for numeric variables. See Details

dfc.dir chi-square degrees of freedom for direction set search for Powell's algorithm for numeric variables. See Details

initc.dir initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbc.dir lower bound for direction set search for Powell's algorithm for numeric variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

ckerbound character string controlling continuous-kernel support handling. Can be set as none (default kernel on full support), range (use sample min/max, except that beta kernels use the outer half-spacing rule in np.kernels), or fixed (use ckerlb/ckerub). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see np.kernels.

ckerlb numeric scalar/vector of lower bounds for continuous variables used when ckerbound="fixed". Must satisfy lower-bound validity for each continuous variable (e.g., <= min(variable)). Use -Inf for unbounded below. See np.kernels for bounded-kernel normalization details.

ckerub numeric scalar/vector of upper bounds for continuous variables used when ckerbound="fixed". Must satisfy upper-bound validity for each continuous variable (e.g., >= max(variable)). Use Inf for unbounded above. See np.kernels for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

scale.factor.init
deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to 0.5. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is max(scale.factor.init, scale.factor.search.lower). See Details.

scale.factor.init.lower
lower endpoint for random continuous scale-factor starts. Defaults to 0.1. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). See Details.

scale.factor.init.upper
upper endpoint for random continuous scale-factor starts. Defaults to 2.0. It must be greater than or equal to the effective lower endpoint, max(scale.factor.init.lower, scale.factor.search.lower); otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels.

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.

ckertype character string used to specify the continuous kernel type. Can be set as gaussian, epanechnikov, or uniform, or beta; see Details for the finite-support beta contract. Defaults to gaussian.

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

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

Local-Polynomial Model Specification: These arguments control the local-polynomial estimator, basis, and fixed degree specification.

`basis` basis selector relevant only when `regtype="lp"`. Supported values are `"glp"`, `"additive"`, and `"tensor"`. Let d_j denote the degree for continuous predictor j and q the number of continuous predictors. With one segment per predictor (no internal knots), basis dimensions are: $1 + \sum_{j=1}^{q} d_j$ for additive, $\prod_{j=1}^{q}(d_j + 1)$ for tensor, and $1 + |\{\alpha : \alpha_j \leq d_j,\ 0 < \sum_j \alpha_j \leq \max_j d_j\}|$ for generalized local-polynomial (GLP) basis construction.

`bernstein.basis` logical flag relevant only when `regtype="lp"`. If `FALSE` (default), the GLP basis uses raw local-polynomial powers (allowing extrapolation). Raw powers can be numerically unstable, particularly above degree one or when predictor offsets are large relative to their spread. The option `bernstein.basis=TRUE` is provided to improve numerical conditioning; it does not remedy genuine collinearity. Centring/scaling continuous predictors is another explicit user choice. The conditioning diagnostic assesses the selected basis, so numerical rank deficiency need not imply genuinely redundant predictors. The package does not silently switch the selected basis. For `basis="glp"`, the historical selector `TRUE` uses the degree-graded, orthonormal shifted-Legendre construction described in `npreg`; it has exactly the same complete, coordinate-capped total-degree polynomial span as the raw GLP representation and is an exact deterministic linear transformation of Bernstein polynomial spaces. Additive and tensor bases retain their fixed-coordinate-degree Bernstein representation. For `bernstein.basis=TRUE`, prediction/evaluation points must lie within the training support of each continuous predictor. For automatic degree search, if `bernstein.basis` is not explicitly supplied, the search route defaults to `TRUE` for numerical stability. Explicit `bernstein.basis=FALSE` is honored, but raw-polynomial search can be poorly conditioned at higher degrees. For `regtype="ll"` and `regtype="lp"`, a pre-optimization design-conditioning check is performed on the training continuous design: rank deficiency triggers an error, and large condition number ($\kappa(B)$) triggers warning/error thresholds to avoid unstable optimization dominated by ridging.

`degree` a user-supplied vector of fixed polynomial degrees for the continuous predictors (exactly one degree per continuous predictor), relevant only when `regtype="lp"`. When `degree.select="manual"`, this must be supplied explicitly. Entries must be non-negative integers in `[0,100]`; values above 25 trigger a numerical-stability warning. With no continuous predictors, `regtype="lp"` is only admissible when `degree=0`, the local-constant equivalent. Bandwidth optimization treats this vector as fixed input and optimizes only bandwidths.

`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. `lp` specifies a local polynomial estimator with polynomial degree(s) given by `degree` for continuous predictors, or selected automatically when `degree.select != "manual"`. `ll` and positive-degree `lp` require at least one continuous predictor; for categorical-only predictors use `lc` or `lp` with `degree=0`. The beta associated-kernel route supports `lc`, `ll`, and `lp`. Defaults to `lc`.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

`nomad` logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When `TRUE`, any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` are filled with `regtype="lp"`, `search.engine="nomad+powell"`, `degree.select="coordinate"`, `bernstein.basis=TRUE`, `degree.min=0L`, `degree.max=10L`, `degree.verify=FALSE`, and `bwtype="fixed"`. Explicit incompatible settings error immediately; in particular, explicit `nomad=TRUE` currently requires `regtype="lp"`, `bwtype="fixed"`, automatic degree search, no explicit `degree`, at least one continuous predictor, and `search.engine %in% c("nomad", "nomad+powell")`. This shortcut does not change the meaning of `nmulti` or `nomad.nmulti`: `nmulti` remains the outer restart count, while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in `bw$nomad.shortcut` for a returned object `bw`; when available, `nomad.time` and `powell.time` record the direct-search and Powell-polish timing components.

`nomad.nmulti` non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when `regtype="lp"` and automatic degree search uses `search.engine="nomad"` or `"nomad+powell"`. Defaults to `0L`, which preserves the current one-start-per- restart behavior. This does not replace `nmulti`: `nmulti` controls outer restarts, while `nomad.nmulti` controls inner NOMAD multistarts within each outer restart.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for MADS/NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and unsupported cache-off callback settings are rejected before solver entry.

`nomad.remin` logical flag controlling the optional second NOMAD hot start. When `TRUE`, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to `FALSE`; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

`search.engine` character string controlling the automatic local-polynomial search backend when `regtype="lp"` and `degree.select != "manual"`. `"nomad+powell"` (default) performs direct joint mixed discrete/continuous search over fixed bandwidths and the degree vector using the native **crs** NOMAD C API, followed by one Powell hot start from the NOMAD solution. `"nomad"` performs the direct joint NOMAD search without the Powell refinement. `"cell"` uses the legacy profiled degree-grid search built from repeated fixed-degree bandwidth solves. NOMAD-based search supports `bwtype="fixed"`, `bwtype="generalized_nn"`, and `bwtype="adaptive_nn"`; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, and bounded search transformations.

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)).

invalid.penalty
a character string specifying the penalty used when the optimizer encounters invalid bandwidths. "baseline" returns finite guidance derived from the exact criterion-specific constant null; "dbmax" returns DBL_MAX. Terminal invalidity is preserved until finalization, and an automatically selected point is checked once against the raw criterion before it is accepted. Defaults to "baseline".

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

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(2,ncol(xdat)).

penalty.multiplier
a numeric multiplier applied to the baseline penalty when invalid.penalty="baseline". It must be finite and strictly greater than one. Defaults to 10.

powell.remin logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, TRUE restarts the local search from the located minimum. For search.engine="nomad+powell", this controls only the final Powell bandwidth-polish step. The default is TRUE for ordinary Powell routes and FALSE for the Powell polish after NOMAD unless explicitly supplied.

bwsolver bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Use search.engine rather than bwsolver for automatic degree search.

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)).

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)).

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.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

... additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even

if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via `newdata` does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state. The resolved NA policy is retained rather than re-read from later option settings. With explicit replacement training data, an explicitly supplied `na.action` overrides that policy, including `NULL`. Any remaining incomplete native training rows are omitted and recorded; this does not make incomplete rows valid estimator inputs. Character NA policies use the lookup owner of `stats::model.frame`; pass a custom function object explicitly.

Retaining samples increases object size, and derived models can contain separate retained samples in their child bandwidth objects. User formula and custom-function environments are not generally pruned. They can retain unrelated caller objects, so serialized file size may be much larger than `object.size()` suggests. Sample ownership does not guarantee compact serialization of arbitrary user environments.

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)`, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

The bandwidth-selection argument surface is easiest to read by decision group: data and existing bandwidth inputs; local-polynomial/NOMAD controls when polynomial-adaptive regression is requested; bandwidth criterion and representation; continuous kernel and support controls beginning with `cker*`; categorical kernel controls `ukertype` and `okertype`; and numerical search initialization, tolerances, and feasibility controls. Users who call `npreg` without a bandwidth object can pass these same bandwidth-selection controls through that function's `....`

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

`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. For fixed-degree local-constant/local-linear regression, and for local-polynomial regression with `degree.select="manual"`, the bandwidth search uses multidimensional Powell direction-set optimization.

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.

Exact ties can make the literal kth continuous nearest-neighbor radius zero. Such a bandwidth is undefined and is rejected; see `np.kernels` for the precise contract and remedies.

`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. Beta associated kernels of orders 2, 4, 6, and 8 are additionally available for local-constant, local-linear, and local-polynomial bandwidth objects. 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 beta contract requires `ckertype="beta"`, `ckerorder` in `c(2,4,6,8)` and finite `ckerbound="fixed"` bounds or `ckerbound="range"` for each continuous predictor. Local-constant, local-linear and local-polynomial regression support mixed continuous and categorical predictors. Explicit bandwidth objects and automatic `cv.ls`/`cv.aic` search support fixed, generalized-NN, and adaptive-NN modes. Normal-reference selection is unsupported. Gradient and standard-error support is described in `npreg`. See `npreg` and `np.kernels` for the exact beta weight, bandwidth convention, ratio estimator, and standard-error definition.

When `regtype="lp"` and `degree.select != "manual"`, `npregbw` can jointly determine the continuous-predictor degree vector and bandwidth coordinates. With `search.engine="cell"`, the objective is profiled over the degree grid using cached coordinate-wise or exhaustive search together with the existing fixed-degree bandwidth optimizer. With `search.engine="nomad"` or `"nomad+powell"`, the package instead evaluates the cross-validation criterion directly over the joint space of fixed bandwidths and polynomial degrees using the native **crs** NOMAD C API. `"nomad+powell"` then performs one Powell hot start from the NOMAD solution and retains the better of the direct NOMAD and polished solutions. This direct joint-search route follows the polynomial-adaptive cross-

validation rationale of Hall and Racine (2015). When bernstein.basis is not explicitly supplied, the automatic search route defaults to bernstein.basis=TRUE for numerical stability; explicit bernstein.basis=FALSE is honored but can be poorly conditioned at higher degrees. NOMAD multistarts are initialized more conservatively than the full degree search box: start 1 is the user-supplied degree/bandwidth vector when provided and otherwise a clipped degree-one vector, while later starts are reproducible random draws from a reduced degree proposal box whose candidates are screened using dim_basis(). This heuristic is used only to obtain feasible, numerically safer, and quicker initial evaluations; it does not restrict the admissible degree region searched by NOMAD. The direct NOMAD backend is provided by the suggested package crs, so install crs before using search.engine="nomad", "nomad+powell", nomad=TRUE, or nomad="auto".

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For regression it expands any missing values to the equivalent long-form call

```
npregbw(...,
        regtype = "lp",
        search.engine = "nomad+powell",
        degree.select = "coordinate",
        bernstein.basis = TRUE,
        degree.min = 0L,
        degree.max = 10L,
        degree.verify = FALSE,
        bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value nomad="auto" applies the same LP shortcut but leaves search.engine and degree.select eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit search.engine or degree.select choices are honored. When the direct NOMAD route is active, nmulti controls the package-level outer restart count while nomad.nmulti controls the inner native **crs** NOMAD API multistart count used within each outer restart. The default nomad.nmulti=0L preserves the current single-start inner NOMAD behavior. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as NB_THREADS_PARALLEL_EVAL > 1.

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`.

Prediction follows a two-stage contract. Calling `predict(bw)` on the returned bandwidth object is a convenience forwarder for `npreg(bws = bw)` and returns a fitted `npregression` object; it does not return a numeric vector of predictions. Call `fitted(fit)`, `residuals(fit)`, or `predict(fit, newdata = ...)` on that fitted object to obtain numeric results. The canonical workflow is

```
bw <- npregbw(y ~ x, data = mydata)
fit <- predict(bw)             # equivalent to npreg(bws = bw)
fitted(fit)
residuals(fit)
predict(fit, newdata = newdata)
```

The functions `summary` and `plot` also support `rbandwidth` objects.

Book And Method Pointers

Local-polynomial search uses the numerical-ridging policies described in `npreg` (Seifert and Gasser, 2000). This is solve stabilization, not the fitted-residual uncertainty correction.

`npregbw` selects bandwidths and, when requested, local-polynomial degree/search metadata for estimating the conditional mean $m(x) = E[Y \mid X = x]$. For the estimator target and local-polynomial interpretation, see `npreg`.

For book-length derivations, see Li and Racine (2007), Chapter 2 *Regression*, especially Sections 2.2, 2.4, and 2.5, and Chapter 4 *Kernel Estimation with Mixed Data*, especially Sections 4.2 and 4.4. The later workflow treatment is Racine (2019), Chapter 6 *Conditional Mean Function Estimation*.

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(2, 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.

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

Chen, S.X. (2000), "Beta kernel smoothers for regression curves," Statistica Sinica, 10, 73-91.

Fan, J. and I. Gijbels (1996), *Local Polynomial Modelling and Its Applications,* Chapman and Hall.

Hall, P. and J.S. Racine (2015), "Infinite Order Cross-Validated Local Polynomial Regression," Journal of Econometrics, 185, 510-525.

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

np.kernels, np.options, plot, plot.np 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")
with(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...

if (interactive()) 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...

if (interactive()) 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.

})

# 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")
with(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...

if (interactive()) 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...

if (interactive()) 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.

})

## End(Not run)
```

---

npreghat *Nonparametric Regression Hat Operator*

Description

Constructs nonparametric regression hat operators for npreg-compatible bandwidth objects. The returned operator $H^{(s)}$ maps responses to fitted values or derivative estimates via $H^{(s)}y$.

Usage

```
npreghat(bws, ...)

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

## S3 method for class 'rbandwidth'
npreghat(bws,
         txdat = stop("training data 'txdat' missing"),
         exdat, y = NULL,
         output = c("matrix", "apply", "constraint"),
         basis = NULL,
         bernstein.basis = NULL,
         degree = NULL,
         deriv = NULL,
         leave.one.out = FALSE,
         ridge = 0,
         s = NULL,
         ...)

## S3 method for class 'npregression'
npreghat(bws,
         txdat,
         y,
         ...)

## S3 method for class 'npreghat'
predict(object,
        newdata = NULL, y = NULL,
        output = c("matrix", "apply", "constraint"),
        s = attr(object, "s"),
        leave.one.out = attr(object, "leave.one.out"),
        deriv = NULL,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, formula/data interface, training data, and evaluation data.

bws	An object of class rbandwidth or npregression.

`data` This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`. A data frame used with the formula interface.

`exdat` Optional evaluation predictors.

`newdata` Optional evaluation data for formula and predict methods. Without a new evaluation grid, prediction retains the operator's original training-observation or external-query identity, including when changing the derivative order. An explicit grid remains external even when its coordinates equal the training data; this distinction can affect nearest-neighbor radii. Formula newdata is evaluated with the trained predictor transformations. Explicit native `exdat` takes precedence over `newdata` and is already in the transformed predictor coordinates. Retained operators carry their effective polynomial specification, including construction-time degree, basis and Bernstein overrides.

`txdat` Training predictors.

Local-Polynomial Controls: These arguments control local-polynomial basis, degree, derivatives, leave-one-out behavior, and ridge stabilization.

`basis` Local polynomial basis: `"glp"`, `"additive"`, or `"tensor"`.

`bernstein.basis`
Logical; select the stabilized LP representation. For GLP, `TRUE` selects the degree-graded, orthonormal shifted-Legendre representation; additive and tensor bases retain their Bernstein representations. See `npreg` for the definitions.

`degree` Optional local polynomial degree vector override (LP path).

`deriv` Convenience alias for `s`.

`leave.one.out` Logical; if `TRUE`, compute in-sample leave-one-out hat weights. This cannot be combined with explicit `exdat`/`newdata`. Local-constant first-derivative weights with `leave.one.out = TRUE` are not currently supported; local-constant level weights and the supported local-polynomial leave-one-out derivatives remain available.

`ridge` A finite nonnegative scalar. The default `0` retains the existing automatic numerical ridging policy. Computational owners that implement only that policy reject an explicit nonzero value rather than ignoring it; the generic mixed-derivative owner continues to support an explicit base ridge, with absolute increments `1/n.train` on that explicit-ridge route. Default higher and mixed derivatives instead share the canonical LP solve-admission and automatic ridge policy used by fitting. Wider systems restore the response intercept from the signed pristine Gram intercept.

`s` Derivative multi-index over continuous predictors, with finite non-negative integer entries. A named vector selects predictors by their continuous-variable names; omitted coordinates have order zero. An unnamed scalar selects the first continuous predictor. Unknown or duplicated names are errors. Explicit `s` takes precedence over the `deriv` alias; an explicit `deriv` in prediction overrides the retained selector.

Method Objects: This argument identifies a fitted hat-operator object supplied to an S3 method.

`object` An object returned by `npreghat`.

Operator Output: These arguments control whether the operator is returned as a matrix, applied directly, or returned as a quadratic-programming constraint design matrix.

`output` Either `"matrix"` for the hat matrix, `"apply"` for direct application to `y`, or `"constraint"` for the row-weighted transpose `t(H) * y`.

`y` Optional response vector or matrix for apply mode. For `output = "constraint"`, `y` is required and must be a vector or one-column object. When the fitted response is a factor, explicit or borrowed factor responses use its retained numerical level coding, including numeric level labels. Payload levels are aligned to the fitted levels. Explicit numerical vectors or matrices are applied as supplied, without recoding.

Additional Arguments: Further arguments are passed to methods.

`...` Additional arguments passed to methods.

Details

With a formula-derived bandwidth object, supplying only `y` replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective `na.action` is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use `data` for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

Formula variables of class `ts`, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Local-polynomial operators retain the fitted solve's numerical ridge adjustment; see Numerical Ridging in `npreg` and its Seifert–Gasser reference. A hat operator alone does not estimate residual variance or apply the residual normalization used by regression inference. Fewer nonzero-weight donor rows than basis coefficients trigger the same existing ridge policy as fitting, for levels and derivative operators. This is a structural rank bound, not a new numerical conditioning threshold.

For `output = "matrix"`, the return value is a matrix with class `c("npreghat", "matrix")` so it can be used directly in matrix products, e.g. `H %*% y`. Attributes on the matrix store metadata used by `predict.npreghat`.

For `output = "apply"`, the function returns `H^{(s)} y` directly and accepts matrix right-hand sides for one-shot bootstrap-style calculations.

An evaluation row whose required kernel normalization is exactly zero or non-finite is returned as an `NA` hat row. Defined rows remain available, and every finite nonzero normalization is used literally.

For non-beta local-polynomial mean operators with fixed or generalized nearest-neighbor bandwidths, a failed external evaluation row is returned as `NA` only when its pristine moment system is

finite and every computed kernel weight is exactly zero. One warning identifies the affected result rows (after missing evaluation rows are omitted). This reports the computed weights; it does not distinguish absence of geometric support from numerical underflow. Defined rows retain their ordinary values. In matrix output, the corresponding `ridge.used` entries are `NA`; apply output has undefined rows, and constraint output has the corresponding undefined columns. Other solve failures and zero nearest-neighbor radii remain errors. Training, leave-one-out and internal complete-operator consumers retain strict failure behavior.

The same all-zero-weight certificate is used for external non-beta local-constant (including degree-zero local-polynomial) first-derivative operators with fixed, generalized-nearest-neighbor or adaptive-nearest-neighbor bandwidths. Unsupported rows become `NA`, while supported derivative rows keep their usual values. A zero sum formed by cancellation of nonzero weights, non-finite arithmetic, and invalid nearest-neighbor radii remain errors; they are not classified as empty rows.

Default higher and mixed local-polynomial derivative operators also use the finite-pristine-system, all-zero-weight certificate for external rows. Their `ridge.used` entries are then `NA`, with the same warning and output shapes described above. An explicit positive `ridge` retains its requested regularized operator; it is not replaced by automatic ridging or this default unavailable-row policy.

For local-constant beta kernels, one first derivative is evaluated by the same analytic C kernel used by `npreg(..., gradients=TRUE)`. If an exact support endpoint creates a raw-kernel jump, the hat operator cannot know whether a future response will cancel it in the normalized ratio; the entire affected row is therefore returned as `NA` with a warning. The compiled estimator route can still return a finite response-specific gradient.

For `output = "constraint"`, the function returns `t(H^{(s)}) * y`, the row-weighted transpose commonly used as the design matrix in shape-constrained quadratic-programming examples. This is a convenience route that is exactly equivalent to obtaining `H` with `output = "matrix"` and then computing `t(H) * y`; it does not solve a constrained estimation problem.

Value

Either a hat matrix (class `"npreghat"`) or the applied result `H^{(s)} y`, or the constraint design matrix `t(H^{(s)}) * y`, depending on `output`.

Examples

```
## Not run:
data(cps71)
bw <- npregbw(xdat = cps71$age, ydat = cps71$logwage,
              regtype = "ll", bandwidth.compute = FALSE, bws = 1.0)
H <- npreghat(bws = bw, txdat = data.frame(age = cps71$age))
H.fitted <- H
A <- npreghat(bws = bw, txdat = data.frame(age = cps71$age),
              y = cps71$logwage, output = "constraint")
all.equal(A, t(H) * cps71$logwage)
ghat <- npreg(bws = bw)
head(cbind(fitted(ghat), H.fitted), n = 2L)

## 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, ...)

## Default S3 method:
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,
        regtype = NULL,
        degree = NULL,
        nomad = FALSE,
        ...)

## S3 method for class 'formula'
npregiv(y,
        data = NULL,
        subset,
        na.action,
        newdata = NULL,
        zeval = NULL,
        xeval = NULL,
        ...)

## S3 method for class 'npregiv'
plot(x, gradients = FALSE,
        data.overlay = TRUE, data.rug = FALSE, ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the response, endogenous variables, instruments, exogenous covariates, and evaluation data.

`data`	This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.
	an optional data frame used with the formula interface. Formula roles are written explicitly as `y ~ z	w` or `y ~ z	w	x`.
`subset`	an optional expression selecting training rows in the formula interface.
`na.action`	a function specifying missing-value handling for the formula model frame.
`newdata`	an optional data frame supplying fit-time evaluation values for the formula `z` and, when present, `x` partitions. Explicit `zeval` and `xeval` take precedence. This is fit-time evaluation, not a post-fit `predict()` method.
`w`	a q-variate data frame of instruments. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

x — for npregiv.default, an r-variate data frame of exogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof. For plot.npregiv, an npregiv object.

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.

y — a one (1) dimensional numeric or integer vector of dependent data, or a formula using the explicit IV grammar described under data. Each response element i corresponds to 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.

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.

gradients — logical; display the stored first partial derivative instead of the structural function. The current plot method supports one endogenous coordinate. The default is FALSE.

data.overlay — logical; overlay the training response on a structural-function display. The default is TRUE, as for plot.npregression. It is ignored for derivative displays because response values and derivatives do not share a common vertical scale. When limits are not supplied, the structural curve and response are both included in the displayed range.

data.rug — logical; add a rug showing the training z support. The default is FALSE. A rug may accompany either structural-function or derivative displays; categorical endogenous coordinates omit the numerical rug.

Landweber-Fridman Iteration Controls: These arguments control the Landweber-Fridman iteration path.

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 controlling whether alpha is reselected after the first response-adaptive bandwidth/operator refinement. It does not change the order of the fixed order-two criterion used to select alpha.

iterate.Tikhonov.num — a positive integer giving the number of outer response-adaptive bandwidth/operator refinement cycles. It is not the order of iterated Tikhonov regularization.

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.

Optimization Controls: These arguments control numerical optimization for the inverse problem.

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 10.

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). This legacy argument is retained. If it is supplied together with modern smoothing controls, the specifications must be equivalent.

regtype an optional modern fixed-degree spelling: "lc", "ll", or "lp". Omitting both regtype and degree preserves the historical p=1 default exactly.

degree one nonnegative integer supplied with regtype="lp". The scalar controls the existing local-polynomial p engine; vectors are not recycled across IV stages.

nomad logical or "auto". Only FALSE is currently supported by the bespoke npregiv objective. TRUE and "auto" fail explicitly rather than being silently ignored.

Returned Weights And Smooth Residuals: These arguments control returned kernel weights, starting values, residual smoothing, and iteration stopping behavior.

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. For more than one continuous coordinate, the returned component is a named list containing one weight matrix per pure coordinate partial.

`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. For more than one continuous coordinate, the returned component is a named list containing one weight matrix per pure coordinate partial.

`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 $E(E(Y|w)|z)$ (this can be overridden by providing `starting.values` below)

`starting.values`
: an optional finite numeric vector supplying the training-row structural initialization. Its derivative state is estimated from the corresponding local-polynomial interpolant using the selected smoothing contract; it is not set to zero. When weights are requested, this supplied initialization is treated as fixed with respect to y.

`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)

Tikhonov Regularization Controls: These arguments control Tikhonov regularization and its bandwidth.

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

`alpha.iter`
: a positive numeric scalar that, if supplied, is used for the ordinary Tikhonov solve after the outer bandwidth/operator refinement rather than selecting a refined regularization parameter numerically.

`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). A Landweber-Fridman replay returns the recomputed stopping diagnostics and uses `convergence="BANDWIDTH_REPLAY"`, including when `norm.index=1`.

Additional Arguments: Further arguments are passed to lower-level kernel-sum and estimation routines.

`...` Only fixed bandwidths are currently supported. Supplying `bwtype="generalized_nn"` or `bwtype="adaptive_nn"` raises an informative error; use `bwtype="fixed"` (the default). Additional kernel and bandwidth-scaling arguments supplied to the private IV regression and kernel-sum stages. In particular, `bandwidth.divide`, `ukertype`, and `okertype` retain their ordinary lower-level meanings and are owned once per call, avoiding duplicate-argument collisions. Categorical bandwidth searches use the valid upper bound for the selected categorical kernel. For the plot method, graphical arguments control the fitted curve and base-graphics frame as applicable. Explicit `xlim` and `ylim` are honored; automatic limits include every active curve, overlay, and rug layer.

Details

Formula variables of class `ts`, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Local-polynomial nuisance fits may use numerical diagonal ridging. For its Seifert–Gasser motivation, see `npreg`. This is distinct from regularization of the instrumental-variable inverse problem and does not apply the ordinary-regression residual uncertainty correction to IV inference.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

The formula interface uses explicit role order: `y ~ z | w` and `y ~ z | w | x`. Additive transformations and factors are retained through `model.frame`. Dot expansion and formula interaction operators are rejected because they do not identify a unique mixed-data kernel role; create an explicit transformed variable (for example with `I(a * b)`) instead. Native vector/data-frame calls remain supported unchanged.

For Landweber-Fridman regularization, state N means the complete state after exactly N residual-correction updates. Column N of `phi.mat`, `norm.stop[N]`, `norm.index`, the returned structural curve, derivatives, and optional weights therefore all refer to that same state. With `penalize.iteration=TRUE`, the stopping value is N times the normalized squared conditional-residual norm evaluated at state N.

Multivariate derivative components are pure coordinate partials in the order of the continuous columns in the augmented structural block `data.frame(z, x)`. Thus the first-derivative matrix has one named column per continuous endogenous or appended exogenous coordinate; it does not contain a mixed derivative.

The Tikhonov route solves the Darolles et al. operator equation with $r = E[Y|W]$ and right-hand side T^*r. Its default data-driven α selector is the fixed order-two criterion of Feve and Florens (2010), Equation (3.10); the returned curve is the ordinary Tikhonov solution. `iterate.Tikhonov.num` counts outer bandwidth and operator refinements. The current implementation selects `alpha.iter` during the first outer cycle and reuses it in later cycles; it does not implement a per-cycle alpha policy or the alternative Darolles extended- residual selector.

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=min(2,p)` for the first iteration, where `p` is the dimension of the current smoothing problem.

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 `min(2,p)` 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`, `plot`, `fitted`, `residuals`, and `gradients` support objects of this type. `fitted` returns the selected structural function and `gradients` returns its stored first derivatives. Both are training-row accessors even when fit-time evaluation data were supplied; evaluation results remain in the `phi.eval` and `phi.deriv.eval.*` fields. Residuals use the aligned training structural curve. For formula fits using an exclusion-style `na.action`, these accessors restore omitted rows. `summary` returns a structured `summary.npregiv` object with a dedicated print method. The printed smoothing description reports the effective continuous-stage policy using the same `Kernel Regression Estimator:` labels as `npreg`: `Local-Constant`, `Local-Linear`, or, for `regtype="lp"`, the local-polynomial degree and basis details. The retained legacy `p` field is not printed separately.

The plot method displays the structural level by default, uses a separate fit-time evaluation grid when `phi.eval` is available, and overlays the training response by default without replacing that evaluation curve. Set `gradients=TRUE` for the first derivative and `data.rug=TRUE` for the training-support rug.

`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. The compact `bws`, `smoothing.spec`, and `stage.specs` components record stage bandwidth

payloads and requested/effective fixed-degree smoothing state without replacing the legacy bandwidth fields.

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

np.kernels, np.options, plot npregivderiv, npreg

Examples

```
## Not run:
## This illustration was made possible by Samuele Centorrino
## <samuele.centorrino@univ-tlse1.fr>

set.seed(42)
n <- 500
trim <- 0.005

## Endogenous-regressor DGP:
##
## y = phi(z) + u
## E(u | z) != 0
## z = 0.2 w + v
## E(u | w) = 0

v <- rnorm(n, mean = 0, sd = 0.27)
eps <- rnorm(n, mean = 0, sd = 0.05)
u <- -0.5 * v + eps
w <- rnorm(n)

phi <- function(z) z^2

z <- 0.2 * w + v
y <- phi(z) + u

ivdata <- data.frame(y, z, w)
ivdata <- ivdata[order(ivdata$z), ]

model.iv <- npregiv(y ~ z | w, data = ivdata, regtype = "ll")
phi.iv <- fitted(model.iv)

## Compare with the non-IV local-linear estimator of E(y | z).

model.noniv <- npreg(y ~ z, data = ivdata, regtype = "ll")
mean.noniv <- fitted(model.noniv)

## Restrict the display to the central 99 percent of the z observations.

zlim <- quantile(ivdata$z, c(trim, 1 - trim))
keep <- ivdata$z >= zlim[1L] & ivdata$z <= zlim[2L]
ylim <- range(ivdata$y[keep], phi.iv[keep], mean.noniv[keep])

plot(ivdata$z[keep], phi(ivdata$z[keep]),
     ylim = ylim,
     ylab = "Y",
     xlab = "Z",
     main = "Nonparametric Instrumental Kernel Regression",
```

```
     type = "l",
     lwd = 2)

points(ivdata$z[keep], ivdata$y[keep], cex = 0.25, col = "grey")
lines(ivdata$z[keep], phi.iv[keep], col = "blue", lwd = 2, lty = 2)
lines(ivdata$z[keep], mean.noniv[keep], col = "red", lwd = 2, lty = 4)

legend("topright",
       c(expression(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, ...)

## Default S3 method:
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,
              regtype = c("ll","lc","lp"),
              degree = NULL,
              nomad = FALSE,
              ...)

## S3 method for class 'formula'
npregivderiv(y,
              data = NULL,
              subset,
              na.action,
              newdata = NULL,
              zeval = NULL,
              weval = NULL,
              xeval = NULL,
              ...)

## S3 method for class 'npregivderiv'
plot(x, gradients = TRUE,
              data.overlay = TRUE, data.rug = FALSE, ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the response, endogenous variables, instruments, exogenous covariates, and evaluation data.

`data`	This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`. an optional data frame used with the formula interface. This estimator accepts `y ~ z | w`. The shared parser recognizes the optional `| x` partition but this univariate derivative operator rejects it explicitly; see Details.
`subset`	an optional expression selecting training rows in the formula interface.
`na.action`	a function specifying missing-value handling for the formula model frame.
`newdata`	an optional data frame supplying fit-time values of formula variable `z` at which to evaluate the structural derivative. It may contain any positive number of rows. Explicit `zeval` takes precedence.
`w`	a q-variate data frame of instruments. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
`weval`	retained for compatibility. The inverse problem is defined on the training instruments, so a supplied value must equal `w`; use `zeval` to request derivative evaluation.

`x` for `npregivderiv.default`, reserved for interface compatibility. The present derivative estimator supports one univariate endogenous `z` and does not yet support a separate exogenous `x`; a non-`NULL` value fails explicitly. For `plot.npregivderiv`, an `npregivderiv` object.

`xeval` reserved with `x`; it is not available until the derivative operator supports separate exogenous regressors.

`y` a one (1) dimensional numeric or integer vector of dependent data, or a formula using the explicit IV grammar described under `data`. Each response element i corresponds to row i of `z`.

`z` one finite continuous numeric endogenous regressor with at least two distinct support points.

`zeval` finite values of the endogenous regressor at which to evaluate the structural derivative. The training inverse problem, residuals, centering, stopping rule, and selected state remain based on `z` and `w`. By default no separate evaluation surface is stored.

`gradients` logical; display the structural derivative when `TRUE`, the default, or the reconstructed training structural function when `FALSE`. Derivative displays use `zeval` and `phi.prime.eval` when a separate evaluation grid was supplied.

`data.overlay` logical; overlay the training response on a structural-function display. The default is `TRUE`, as for `plot.npregression`. It is ignored when `gradients=TRUE` because response values and derivatives do not share a common vertical scale. When limits are not supplied, the structural curve and response are both included in the displayed range.

`data.rug` logical; add a rug showing the training `z` support. The default is `FALSE`, and it applies to both derivative and structural-function displays.

Iteration Controls: These arguments control derivative iteration and reproducibility settings.

`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 complete positive Landweber-Fridman states to evaluate. The initialization state is numbered zero and does not consume this budget.

`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.

Residual Smoothing And Starting Values: These arguments control residual smoothing and the initial derivative path.

`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 $E(E(Y|w)|z)$ (this can be overridden by providing `starting.values` below)

`starting.values`
an optional vector supplying the derivative immediately before the first reported Landweber-Fridman update. When omitted, the initialization derivative is obtained from `start.from` (see details below). A separate `zeval` grid is not accepted with a supplied initialization because no interpolation rule for the externally supplied derivative has been declared. Repeated training values of `z` must have identical supplied derivative values; conflicting values do not define a single-valued function and are rejected before bandwidth preparation.

`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).

Regression Smoothing Controls: These arguments control all internal `npreg` stages consistently.

`regtype` a fixed regression type: `"ll"` (the default), `"lc"`, or `"lp"`. The default applies local-linear, degree-one smoothing to every internal stage containing continuous predictors, matching the default polynomial order of `npregiv`. A categorical-only internal stage uses local constant because no continuous polynomial term exists. Supply `regtype="lc"` to recover the former omitted-default computation.

`degree` one nonnegative integer required with `regtype="lp"`. It is expanded only across the continuous coordinates of each internal regression stage; vectors are not recycled across heterogeneous stages, and a categorical-only stage uses local constant.

`nomad` logical or `"auto"`. Only `FALSE` is currently supported for the multi-stage derivative estimator. Other values fail explicitly until the stage-wise search/state contract is validated.

Additional Arguments: Further arguments are passed to `npreg` and `npksum`.

`...` Only fixed bandwidths are currently supported. Supplying `bwtype="generalized_nn"` or `bwtype="adaptive_nn"` raises an informative error; use `bwtype="fixed"` (the default). Additional arguments supplied to `npreg` and `npksum`. The private integral kernel sum in the empirical adjoint always uses `bandwidth.divide = TRUE`, as required for the ordinary CDF normalization; a `bandwidth.divide` value in `...` continues to be supplied to `npreg` but does not override this internal normalization. Likewise, `ukertype` and `okertype` in `...` control applicable internal regressions, while the private adjoint retains its required `ukertype = "liracine"` and `okertype = "liracine"`. For the plot method, graphical arguments control the fitted curve and base-graphics frame as applicable. Explicit `xlim` and `ylim` are honored; automatic limits include every active curve, overlay, and rug layer.

Details

Formula variables of class `ts`, including lagged variables, are aligned on their common time support before fitting. Other variables must already have the length of that intersection. Supply explicitly aligned rows when using native data arguments instead of a formula.

Local-polynomial nuisance fits may use numerical diagonal ridging. For its Seifert–Gasser motivation, see `npreg`. This is distinct from regularization of the instrumental-variable inverse problem and does not apply the ordinary-regression residual uncertainty correction to IV inference.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

The formula interface uses explicit role order: `y ~ z | w` and `y ~ z | w | x`. Additive transformations and factors are retained through `model.frame`. Dot expansion and formula interaction operators are rejected because they do not identify a unique mixed-data kernel role; create an explicit transformed variable (for example with `I(a * b)`) instead. Native vector/data-frame calls remain supported unchanged.

The current derivative operator is univariate in `z`. Consequently, although the shared parser recognizes the package-wide optional `| x` partition, `npregivderiv` rejects it explicitly rather than silently changing the derivative target. Post-fit `predict()` is not available; use fit-time `newdata` or the native evaluation arguments.

Fit-time evaluation is output-only. Conditional regressions, the empirical adjoint, fitted-residual centering, structural integration, stopping values, and state selection are always computed on aligned training rows. A separate `zeval` accumulates the derivative corrections from those same training states into `phi.prime.eval` and `phi.prime.eval.mat`; it cannot change `phi`, `phi.prime`, their training paths, or `norm.stop`. Structural levels at arbitrary new points are not returned because this derivative estimator's integration and centering normalization is defined on the training support.

Let φ_0' denote the initialization derivative, obtained from `start.from` or supplied directly through `starting.values`. Its integrated and centered curve is φ_0. This state is retained in `starting.values.phi.prime` and `starting.values.phi`; it is not a column of the positive-iteration matrices. For $N = 1, 2, \ldots$, iteration N means exactly N completed residual-correction updates after this initialization. Column N of `phi.prime.mat` and `phi.mat` therefore contains the coherent pair (φ_N', φ_N), and `norm.stop[N]` is evaluated at that same state using the multiplier N. Check the stopping vector for anomalies such as an immediate increase.

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=min(2,p)` for the first iteration, where `p` is the dimension of the current smoothing problem.

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 `min(2,p)` 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.

With iterate.break=TRUE, iteration can stop after the stopping rule turns upward once the initial guard interval has passed; otherwise all states through iterate.max are evaluated. The selected state is then obtained from the computed stopping profile.

Value

npregivderiv returns a npregivderiv object. The generic functions print, summary, plot, fitted, residuals, and gradients support objects of this type. fitted returns the reconstructed training structural function while gradients returns the training derivative estimate. Residuals use the aligned training structural curve even when a separate evaluation grid was supplied. For formula fits using an exclusion-style na.action, these accessors restore omitted rows. summary returns a structured summary.npregivderiv object with a dedicated print method. The printed smoothing description reports the effective continuous-stage policy using the same Kernel Regression Estimator: labels as npreg: Local-Constant, Local-Linear, or, for regtype="lp", the local-polynomial degree and basis details.

The plot method displays the derivative by default, preserving the estimator family's primary target. Set gradients=FALSE to display the reconstructed training structural function. The training response is overlaid only on that level display; data.rug=TRUE adds the training support without replacing an available derivative evaluation curve.

npregivderiv returns a list with components phi.prime, phi, phi.prime.mat, phi.mat, and, when a separate evaluation grid was supplied, phi.prime.eval and phi.prime.eval.mat, num.iterations, norm.stop, convergence, and the initialization components starting.values.phi.prime and starting.values.phi. num.iterations is the selected state returned in phi.prime and phi; the number of states evaluated, including any stopping-rule overshoot, is length(norm.stop). The compact bws, smoothing.spec, and stage.specs components record stage bandwidth payloads and requested/effective fixed-degree smoothing state without replacing the legacy bandwidth fields.

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

np.kernels, np.options, plot npregiv, npreg

Examples

```
## Not run:
## This illustration was made possible by Samuele Centorrino
## <samuele.centorrino@univ-tlse1.fr>

set.seed(42)
n <- 500
trim <- 0.005

## Endogenous-regressor DGP:
##
## y = phi(z) + u
## E(u | z) != 0
## z = 0.2 w + v
## E(u | w) = 0

v <- rnorm(n, mean = 0, sd = 0.27)
eps <- rnorm(n, mean = 0, sd = 0.05)
u <- -0.5 * v + eps
w <- rnorm(n)

phi <- function(z) z^2
phi.prime <- function(z) 2 * z

z <- 0.2 * w + v
y <- phi(z) + u

ivdata <- data.frame(y, z, w)
ivdata <- ivdata[order(ivdata$z), ]

model.ivderiv <- npregivderiv(y ~ z | w, data = ivdata)
deriv.iv <- gradients(model.ivderiv)

## Restrict the display to the central 99 percent of the z observations.
```

```
zlim <- quantile(ivdata$z, c(trim, 1 - trim))
keep <- ivdata$z >= zlim[1L] & ivdata$z <= zlim[2L]

plot(ivdata$z[keep], deriv.iv[keep],
     main = "",
     xlab = "Z",
     ylab = "Derivative",
     type = "l",
     lwd = 2)

curve(phi.prime(x), add = TRUE, col = 2, lty = 2, lwd = 2)
rug(ivdata$z[keep])

## 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,
        se = FALSE,
        ...)

## Default S3 method:
npscoef(bws,
        txdat,
        tydat,
        tzdat,
        nomad = FALSE,
        se = FALSE,
        ...)
```

```
## S3 method for class 'scbandwidth'
npscoef(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        tzdat = NULL,
        exdat,
        eydat,
        ezdat,
        betas = FALSE,
        se = FALSE,
        iterate = TRUE,
        leave.one.out = FALSE,
        maxiter = 100,
        residuals = FALSE,
        tol = .Machine$double.eps,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and smooth-coefficient training data.

`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.

`data` This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

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.

`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.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside npscoef.

nomad — logical shortcut passed through to npscoefbw when bandwidths are computed inside npscoef. When TRUE, the smooth-coefficient bandwidth route fills any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npscoefbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the smooth-coefficient fit is evaluated and which evaluation quantities are returned.

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. With separate training tzdat, this defaults to tzdat; otherwise it follows exdat. Supplying only ezdat evaluates at the original training txdat. The two evaluation roles must have the same row count before omitting missing rows. Native exdat and ezdat columns correspond positionally to their training roles; different column labels do not change this mapping.

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.

Fitted Quantities And Backfitting: These arguments control returned coefficient estimates, errors, residuals, and iterative backfitting.

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.

se — a logical value indicating whether or not asymptotic standard errors should be 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. Fitted objects retain this control, maxiter, tol, and leave.one.out for prediction and lazy residual extraction. Explicit controls supplied to predict override retained values.

`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`.

`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`.

`residuals` a logical value indicating that you want residuals computed and returned in the resulting `smoothcoefficient` object. Defaults to `FALSE`. Residuals refer to the training sample, including when evaluation data are supplied, and do not require `se = TRUE`.

`tol` desired tolerance on the relative convergence of backfit estimates. Defaults to `.Machine$double.eps`.

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart when bandwidths are not supplied.

`...` 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`.

Details

Prediction with `se.fit = TRUE` reports known training mean squared error in `residual.scale`, retaining MSE units, or `NA` when training provenance is unavailable. Evaluation outcomes do not replace this training score. No additional fit is performed to recover it.

In native calls, three unnamed arguments follow the fitter's bandwidth-first convention: `npscoef(bws, txdat, tydat)`; replacing training data does not reuse a separate smoothing sample from the bandwidth object. Supply `tzdat` explicitly to replace that role; otherwise the smoothing data default to the training predictors. Numeric argument types cannot distinguish this from an intended data-first call. To select bandwidths with separate smoothing data, write `npscoef(xdat, ydat, zdat = zdat)` or name all data roles. The selector `npscoefbw(xdat, ydat, zdat)` retains its data-first convention. No argument roles are inferred from vector lengths or values.

With a formula-derived bandwidth object, supplying only `tydat` replaces the evaluated response in retained-training-sample order: one row per observation retained after the original subset and missing-value selection. Original excluded rows are not restored; row names or equal lengths do not imply original-data order. The effective `na.action` is then applied jointly to the retained predictors and replacement response; any new omission indices refer to this retained sample. The selected bandwidths remain fixed, and the returned object retains the replacement sample for subsequent refits. Use `data` for full formula-data replacement, not together with a partial response replacement. Explicit native training arguments retain their separate positional contract.

For native (non-formula) objects, named `newdata` columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over `newdata`.

For a native bandwidth object, `newdata` is split into `exdat` and `ezdat` using the retained predictor and smoothing variable names. Names must identify both roles unambiguously; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

Value

When an external evaluation row has no usable kernel support or its kernel-weighted system cannot be solved, the corresponding fitted value, coefficients and requested standard errors are NA. One warning identifies affected rows; valid rows are retained in their original order. A larger bandwidth or another kernel may provide usable support. Required training and bootstrap fits instead signal an error on this failure; they do not silently discard rows. Being outside the training range alone does not imply that a local fit is undefined.

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 se = TRUE, standard errors of the regression estimates
beta	if betas = TRUE, estimates of the coefficients γ at the evaluation points
grad	estimated derivatives of the conditional mean with respect to the regressors in xdat; these correspond to the non-intercept smooth coefficient estimates at each evaluation point
gerr	if se = TRUE, asymptotic standard errors for grad
resid	if residuals = TRUE, training responses minus their fitted training values, restored to training rows under na.exclude. Evaluation responses, when supplied, are used for evaluation goodness-of-fit summaries, not to redefine these residuals.
R2	coefficient of determination (Doksum and Samarov (1995))
MSE	mean squared error
training.MSE	Known mean squared error against the training outcomes, retained independently of any evaluation-outcome override; NA when it is unavailable without another fit.
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

Book And Method Pointers

Smooth-coefficient moment solves use their own bounded numerical ridging; see npreg for its Seifert–Gasser motivation. The smooth-coefficient covariance is a separate moment calculation, not the ordinary-regression residual-normalized sandwich.

The smooth-coefficient model lets slopes vary with conditioning variables, typically written $Y = X'\beta(Z)+\epsilon$. The functions estimate the coefficient functions $\beta_j(z)$ using mixed-data kernel smoothing over Z, with the reported fitted values obtained by combining the estimated coefficient functions with the corresponding columns of X.

For book-length derivations, see Li and Racine (2007), Chapter 9 *Additive and Smooth (Varying) Coefficient Semiparametric Models*, especially Sections 9.3-9.3.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*, especially the varying-coefficient material.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, `npscoef(..., nomad=TRUE)` and `npscoefbw(..., nomad=TRUE)` expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

For plotting options for fitted smooth-coefficient objects and their bandwidth objects, see `plot.np`.

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

np.kernels, np.options, plot, plot.np, 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)
if (interactive()) 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)
if (interactive()) 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,
          ...)

## Default S3 method:
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          backfit.iterate,
          backfit.maxiter,
          backfit.tol,
          bandwidth.compute = TRUE,
          basis,
          bernstein.basis,
          bwmethod,
          bwscaling,
          bwtype,
          ckerbound,
          ckerlb,
          ckerorder,
          ckertype,
          ckerub,
          cv.iterate,
          cv.num.iterations,
          degree,
          degree.select = c("manual", "coordinate", "exhaustive"),
          search.engine = c("nomad+powell", "cell", "nomad"),
          nomad = FALSE,
          nomad.nmulti = 0L,
          degree.min = NULL,
          degree.max = NULL,
          degree.start = NULL,
          degree.restarts = 0L,
          degree.max.cycles = 20L,
          degree.verify = FALSE,
          nmulti,
          nomad.remin = FALSE,
          powell.remin = TRUE,
          okertype,
          optim.abstol,
```

```
            optim.maxattempts,
            optim.maxit,
            optim.method,
            optim.reltol,
            random.seed,
            regtype,
            ukertype,
            scale.factor.init.lower = 0.1,
            scale.factor.init.upper = 2.0,
            scale.factor.init = 0.5,
            lbd.init = 0.5,
            hbd.init = 1.5,
            dfac.init = 1.0,
            scale.factor.search.lower = NULL,
            ...,
            nomad.opts = list())

## S3 method for class 'scbandwidth'
npscoefbw(xdat = stop("invoked without data 'xdat'"),
            ydat = stop("invoked without data 'ydat'"),
            zdat = NULL,
            bws,
            backfit.iterate = FALSE,
            backfit.maxiter = 100,
            backfit.tol = .Machine$double.eps,
            bandwidth.compute = TRUE,
            cv.iterate = FALSE,
            cv.num.iterations = 1,
            nmulti,
            optim.abstol = .Machine$double.eps,
            optim.maxattempts = 10,
            optim.maxit = 500,
            optim.method = c("Nelder-Mead", "BFGS", "CG"),
            optim.reltol = sqrt(.Machine$double.eps),
            random.seed = 42,
            scale.factor.init.lower = 0.1,
            scale.factor.init.upper = 2.0,
            scale.factor.init = 0.5,
            lbd.init = 0.5,
            hbd.init = 1.5,
            dfac.init = 1.0,
            scale.factor.search.lower = NULL,
            ...,
            nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the smooth-coefficient data, formula interface, and whether bandwidths are supplied or computed.

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.

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.

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.

data
: This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

  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.

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

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.

subset
: an optional vector specifying a subset of observations to be used in the fitting process.

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.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search.

degree.max
: optional scalar or integer vector giving upper bounds for automatic degree search when degree.select != "manual".

degree.max.cycles
: positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for "manual" and "exhaustive" degree selection.

`degree.min` optional scalar or integer vector giving lower bounds for automatic degree search when `degree.select != "manual"`.

`degree.restarts` non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.select` character string controlling local-polynomial degree handling when `regtype="lp"`. `"manual"` (default) treats `degree` as fixed. `"coordinate"` performs cached coordinate-wise search over admissible degree vectors for the continuous `z` variables. `"exhaustive"` evaluates the full admissible degree grid when `search.engine="cell"`. For NOMAD-based search engines, any non-`"manual"` value requests direct joint search over degree and bandwidth coordinates.

`degree.start` optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous `z` variables.

`degree.verify` logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for `search.engine="cell"`.

Backfitting Controls: These controls tune the optional smooth-coefficient backfitting iterations.

`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`.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

`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

Categorical Search Initialization: These controls set categorical search starts.

dfac.init — deterministic fixed-bandwidth start factor for ordered and unordered categorical coordinates. Used only when bwtype="fixed". Defaults to 1.0. Values must not exceed 2.

hbd.init — upper bound for random fixed-bandwidth start factors for ordered and unordered categorical coordinates. Used only when bwtype="fixed". Defaults to 1.5. Must be greater than or equal to lbd.init and must not exceed 2.

lbd.init — lower bound for random fixed-bandwidth start factors for ordered and unordered categorical coordinates. Used only when bwtype="fixed". Defaults to 0.5. Values must not exceed 2 so that categorical fixed starts remain within lawful bandwidth bounds.

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

ckerbound — character string controlling continuous-kernel support handling. Can be set as none (default kernel on full support), range (use sample min/max), or fixed (use ckerlb/ckerub). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see np.kernels.

ckerlb — numeric scalar/vector of lower bounds for continuous variables used when ckerbound="fixed". Must satisfy lower-bound validity for each continuous variable (e.g., <= min(variable)). Use -Inf for unbounded below. See np.kernels for bounded-kernel normalization details.

ckerub — numeric scalar/vector of upper bounds for continuous variables used when ckerbound="fixed". Must satisfy upper-bound validity for each continuous variable (e.g., >= max(variable)). Use Inf for unbounded above. See np.kernels for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

scale.factor.init — deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to 0.5. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is max(scale.factor.init, scale.factor.search.lower). See Details.

scale.factor.init.lower — lower endpoint for random continuous scale-factor starts. Defaults to 0.1. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). See Details.

scale.factor.init.upper
: upper endpoint for random continuous scale-factor starts. Defaults to 2.0. It must be greater than or equal to the effective lower endpoint, max(scale.factor.init.lower, scale.factor.search.lower); otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Cross-Validation Iteration Controls: These controls tune iterative cross-validation behavior.

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.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels.

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.

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

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

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

Local-Polynomial Model Specification: These arguments control the local-polynomial estimator, basis, and fixed degree specification.

basis
: for regtype="lp", the polynomial basis family used for local polynomial fitting. Options are "glp" (default), "additive", and "tensor".

bernstein.basis
: for regtype="lp", logical flag selecting a stabilized representation. For basis="glp", TRUE selects the complete degree-graded, orthonormal shifted-Legendre definition given in npreg; additive and tensor bases retain their fixed-coordinate-degree representation. When automatic degree search is requested and bernstein.basis is not explicitly supplied, the search route defaults to TRUE for numerical stability. Explicit bernstein.basis=FALSE is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

degree for regtype="lp", polynomial degree specification for each continuous z variable.

regtype a character string specifying local smoothing type for the z surface. Options are "lc" (default), "ll", and "lp".

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

nomad logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When TRUE, any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype are filled with regtype="lp", search.engine="nomad+powell", degree.select="coordinate", bernstein.basis=TRUE, degree.min=0L, degree.max=10L, degree.verify=FALSE, and bwtype="fixed". Explicit incompatible settings error immediately; in particular, explicit nomad=TRUE currently requires regtype="lp", bwtype="fixed", automatic degree search, no explicit degree, and search.engine %in% c("nomad", "nomad+powell"). This shortcut does not change the meaning of nmulti or nomad.nmulti: nmulti remains the outer restart count, while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in bw$nomad.shortcut for a returned object bw; when available, nomad.time records the direct-search component. The legacy field powell.time records the subsequent family-native R optim refinement selected by optim.method.

nomad.nmulti non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when regtype="lp" and automatic degree search uses search.engine="nomad" or "nomad+powell". Defaults to 0L, which preserves the current one-start-per- restart behavior. This does not replace nmulti: nmulti controls outer restarts, while nomad.nmulti controls inner NOMAD multistarts within each outer restart.

nomad.opts optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. NULL is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as NB_THREADS_PARALLEL_EVAL > 1.

nomad.remin logical flag controlling the optional second NOMAD hot start. When TRUE, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to FALSE; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

search.engine character string controlling the automatic local-polynomial search backend when regtype="lp" and degree.select != "manual". The compatibility value "nomad+powell" (default) performs direct joint search over the zdat-side continuous bandwidth coordinates and degree vector using the native **crs** NOMAD C API, then applies one family-native R optim hot start from the NOMAD solution using optim.method. The value is retained for API compatibility; "nomad" omits this R-optimizer refinement. "cell" profiles the criterion over the admissible degree

grid using repeated fixed-degree bandwidth solves. NOMAD-based search supports `bwtype="fixed"`, `bwtype="generalized_nn"`, and `bwtype="adaptive_nn"`; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

Numerical Search Controls: These controls set search restart behavior.

`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(2,ncol(xdat))`.

`powell.remin` compatibility argument shared with bandwidth families that use the Powell direction-set solver. Smooth-coefficient fixed-degree and post-NOMAD refinement use R `optim` with `optim.method`; consequently, this argument does not alter their refinement behavior.

Optimization Controls: These arguments control outer optimization behavior for the semiparametric search.

`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 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.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`.

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

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via `newdata` does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state.

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start when that control is exposed. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval when exposed. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)` when both controls are present, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

`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.

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For smooth coefficient regression it expands any missing values to the equivalent long-form call

```
npscoefbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value nomad="auto" applies the same LP shortcut but leaves search.engine and degree.select eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the compatibility "nomad+powell" route. Explicit search.engine or degree.select choices are honored.

When regtype="lp" and degree.select != "manual", npscoefbw can jointly determine the zdat-side local polynomial degree vector together with the associated bandwidth coordinates. With search.engine="cell", the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search together with repeated fixed-degree bandwidth solves. With search.engine="nomad" or "nomad+powell", the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API. For this family, the compatibility value "nomad+powell" then performs one family-native R optim hot start using optim.method and keeps the better of the direct NOMAD and refined answers. This polynomial-adaptive joint-search route is motivated by Hall and Racine (2015). When bernstein.basis is not explicitly supplied, the automatic search route defaults to bernstein.basis=TRUE for numerical stability.

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.

Book And Method Pointers

The smooth-coefficient moment solves use their own bounded additive numerical-ridge policy. For the Seifert–Gasser motivation for ridging, see npreg; this does not impose its ordinary-regression uncertainty formula on smooth-coefficient estimation.

npscoefbw selects bandwidths for the smooth-coefficient model $Y = X'\beta(Z) + \epsilon$. The selected bandwidths control the mixed-data smoothing over Z used to estimate the coefficient functions $\beta_j(z)$.

For book-length derivations, see Li and Racine (2007), Chapter 9 *Additive and Smooth (Varying) Coefficient Semiparametric Models*, especially Sections 9.3-9.3.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*.

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(2,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.

Hall, P. and J.S. Racine (2015), "Infinite Order Cross-Validated Local Polynomial Regression," Journal of Econometrics, 185, 510-525.

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.

Li, A. and Q. Li and J.S. Racine (under revision), "Boundary Adjusted, Polynomial Adaptive, Nonparametric Kernel Conditional Density Estimation," Econometric Reviews.

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

See Also

np.kernels, np.options, plot, plot.np 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,
           B = 399,
           random.seed = 42)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data series, lag count, and statistic variant.

data	a vector containing the variable that can be of type numeric or ts. The sequence must be complete and regularly spaced. Missing values (including NaN) are rejected: omitting them would change the time lags. If necessary, explicitly select a contiguous complete segment; do not delete interior missing observations. For a numeric vector, its order defines time.
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 Controls: These arguments control bootstrap execution and reproducibility settings.

B	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
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.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

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.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

`npsdeptest` applies the dependence-test idea to a time series by comparing the joint law of current and lagged values with the product of their marginal laws. The lag and bootstrap choices determine the serial-dependence alternatives being probed and the null calibration.

For book-length background, see Li and Racine (2007), Chapter 13 *Nonsmoothing Tests*, especially Sections 13.4 and 13.5, and Chapter 12 *Model Specification Tests* for related distributional tests. For entropy and density context, see Racine (2019), Chapter 2.

Usage Issues

The `integration` version uses deterministic tensor-product, corrected composite trapezoidal quadrature. Along each axis the domain extends eight relevant bandwidths beyond the observed support and uses two evaluation points per smallest relevant bandwidth. A registered native callback evaluates the Gaussian marginal and joint densities in bounded tiles, so package-owned working memory does not grow as a dense sample-by-quadrature array. The integration version is generally more computationally demanding than `summation`, but avoids the density division that can make the summation statistic unstable.

The `summation` version fuses both marginal density sums and the joint density sum in one registered native pass with constant auxiliary storage. Bootstrap samples remain ordered index resamples: marginal counts alone cannot represent the lag pairs used by the statistic.

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.

See Also

`np.kernels`, `np.options`, `plot` `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,B=99,method="summation")

if (interactive()) Sys.sleep(5)

## Stationary independent time-series

yt <- ar.series(0.0,rnorm(n))
npsdeptest(yt,lag.num=2,B=99,method="summation")

## Stationary persistent time-series

yt <- ar.series(0.95,rnorm(n))
npsdeptest(yt,lag.num=2,B=99,method="integration")

if (interactive()) Sys.sleep(5)

## Stationary independent time-series

yt <- ar.series(0.0,rnorm(n))
npsdeptest(yt,lag.num=2,B=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 Input: Seed value used to reset the package C backend random number generator.

seed	a single finite numeric value that is representable as a non-negative integer after applying abs().

Details

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

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).

Invalid inputs such as missing values, non-finite values, non-scalars, and values that cannot be represented as integers are rejected with an explicit error.

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

np.kernels, np.options, plot set.seed

Examples

```
npseed(712)
x <- runif(10)
y <- x + rnorm(10, sd = 0.1)
bw <- npregbw(y~x)
```

npsemihat *Experimental Hat Operators for Semiparametric Estimators*

Description

Constructs hat operators for semiparametric estimators so that fitted values or bootstrap draws can be computed by matrix application in one step. These interfaces are currently experimental.

Usage

```
npindexhat(bws,
           txdat = stop("training data 'txdat' missing"),
           exdat = txdat,
           y = NULL,
           output = c("matrix", "apply", "constraint"),
           s = 0L,
           fd.step = NULL,
           ...)

npplreghat(bws,
           txdat = stop("training data 'txdat' missing"),
           tzdat = stop("training data 'tzdat' missing"),
           exdat = txdat,
           ezdat = tzdat,
           y = NULL,
           output = c("apply", "matrix", "constraint"),
           ...)

npscoefhat(bws,
           txdat = stop("training data 'txdat' missing"),
           tzdat = NULL,
           exdat = txdat,
           ezdat = tzdat,
           y = NULL,
           output = c("matrix", "apply", "constraint"),
           ridge = 0,
           iterate = FALSE,
           leave.one.out = FALSE,
           ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted bandwidth object. `npindexhat()` requires class `sibandwidth`, `npplreghat()` requires class `plbandwidth`, and `npscoefhat()` requires class `scbandwidth`.

`exdat` Evaluation `x` data.

`txdat` Training `x` data.

Operator Output And Derivatives: These arguments control operator output, derivative selection, finite-difference compatibility, and apply-mode right-hand sides.

`fd.step` Compatibility argument for `npindexhat(..., s=1)`. If supplied it must be a positive finite scalar, but the current `s=1` route uses the canonical exact derivative operator rather than finite-differencing the fit operator.

`output` Either `"matrix"` for the hat matrix, `"apply"` for direct application to `y`, or `"constraint"` for the row-weighted transpose `t(H) * y`. Note that `npplreghat()` retains `"apply"` as its default for historical compatibility.

`s` For `npindexhat`, `s=0` returns fit operator and `s=1` returns index-derivative operator.

`y` Optional response vector or matrix for apply mode. For `output = "constraint"`, `y` is required and must be a vector or one-column object. When the fitted response is a factor, factor payloads use its retained numerical level coding, including numeric level labels; payload levels are aligned to the fitted levels. Numeric vectors and matrices are used as supplied.

Partially Linear And Smooth-Coefficient Data: These arguments supply the additional `z` data used by partially linear and smooth-coefficient models.

`ezdat` Evaluation `z` data.

`tzdat` Training `z` data for partially linear and smooth-coefficient models.

Smooth-Coefficient Controls: These arguments control smooth-coefficient iteration, leave-one-out behavior, and ridge stabilization.

`iterate` Logical; `npscoefhat()` currently supports `iterate = FALSE` only.

`leave.one.out` Logical; leave-one-out kernel weights for `npscoefhat`. This currently requires evaluation `z` data to match training `z` data.

`ridge` Base ridge term for local linear solves in `npscoefhat`; must be a non-negative finite scalar. The ridge sequence starts at the supplied base (zero by default) and then increments by `1/n.train` as needed for stable solves. The ridge is expressed in the normalized kernel-weight units of the matrix output; categorical profile application converts it to the equivalent profile-system scale.

Additional Arguments: Reserved for future extensions.

`...` Reserved for future extensions.

Details

For local-constant and degree-zero local-polynomial `npindexhat` means, every finite nonzero kernel-weight sum is used without an epsilon floor. An explicitly supplied external evaluation row with all computed weights zero returns `NA` with a warning, preserving its position in matrix and applied outputs. This does not distinguish geometric absence from weights that have underflowed. Signed

cancellation and nonfinite normalizing sums are errors. Required internal and bootstrap computations do not omit undefined rows or replications.

These operators are intended for fixed-X workflows such as one-shot wild bootstrap calculations where many response draws are projected through the same operator. The implementation is intentionally conservative: class and scalar argument contracts are validated explicitly, and unsupported iterative `npscoefhat()` paths fail fast. `npscoefhat()` inherits `regtype`/LP-basis controls from the supplied `scbandwidth` object. For non-fixed `npscoef` bootstrap plotting, these operators can support a frozen approximation, but they do not remove the need to recompute the local smooth-coefficient vector itself for each resample: the local weighted systems depend on the resample weights/counts at each evaluation point, so unlike `npplreg` there is no single global coefficient vector to update once per draw.

Method-specific argument map: `npindexhat()` uses `s`; `fd.step` is accepted for compatibility but the current `s=1` route uses the canonical exact derivative operator; `npplreghat()` and `npscoefhat()` use `tzdat`/`ezdat`; `npscoefhat()` additionally uses `ridge`, `iterate`, and `leave.one.out`.

For `output = "constraint"`, the selected operator is first defined exactly as in `output = "matrix"` and the return value is `t(H) * y`. This is a convenience route for constrained quadratic-programming workflows; it does not solve a constrained estimation problem and it does not change the meaning of `output = "apply"`.

Value

If `output = "matrix"`, returns a hat matrix H. If `output = "apply"`, returns Hy (or HY for matrix right-hand-side input). If `output = "constraint"`, returns the constraint design matrix $\mathrm{diag}(y)H'$ as represented in R by `t(H) * y`.

Examples

```
## Not run:
set.seed(42)
n <- 100
x <- runif(n)
z <- runif(n)
y <- sin(2*pi*x) + 0.5 * z + rnorm(n, sd = 0.1)

tx <- data.frame(x = x)
tz <- data.frame(z = z)

ibw <- npindexbw(xdat = data.frame(x, x2 = x^2), ydat = y,
                 bws = c(0.5, 1.0, 1.0), bandwidth.compute = FALSE)
iH <- npindexhat(bws = ibw, txdat = data.frame(x, x2 = x^2), output = "matrix")
iH.fitted <- iH
iA <- npindexhat(bws = ibw, txdat = data.frame(x, x2 = x^2),
                 y = y, output = "constraint")
all.equal(iA, t(iH) * y)
ifit <- npindex(bws = ibw, txdat = data.frame(x, x2 = x^2), tydat = y)
head(cbind(fitted(ifit), iH.fitted), n = 2L)

pbw <- npplregbw(xdat = tx, zdat = tz, ydat = y,
                 bws = matrix(c(0.2, 0.2), nrow = 2L, ncol = 1L),
                 bandwidth.compute = FALSE)
```

```
pH <- npplreghat(bws = pbw, txdat = tx, tzdat = tz, output = "matrix")
pH.fitted <- pH
pfit <- npplreg(bws = pbw, txdat = tx, tydat = y, tzdat = tz)
head(cbind(fitted(pfit), pH.fitted), n = 2L)

sbw <- npscoefbw(xdat = tx, zdat = tz, ydat = y,
                 bws = 0.2, bandwidth.compute = FALSE)
sH <- npscoefhat(bws = sbw, txdat = tx, tzdat = tz,
                 output = "matrix", iterate = FALSE)
sH.fitted <- sH
sfit <- npscoef(bws = sbw, txdat = tx, tydat = y, tzdat = tz,
                iterate = FALSE)
head(cbind(fitted(sfit), sH.fitted), n = 2L)

## End(Not run)
```

---

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,
          ...,
          B = 399)

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

## 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"),
B = 399,
boot.method = c("iid","wild","wild-rademacher","pairwise"),
boot.type = c("I","II"),
pivot=TRUE,
joint=FALSE,
index = seq_len(ncol(xdat)),
random.seed = 42,
...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth object, formula/data interface, and explicit data inputs.

`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` This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

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`.

Bootstrap And Test Controls: These arguments control bootstrap execution, tested effects, and reproducibility settings.

`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`.

`B` 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"`.

`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`.

`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`.

`pivot` a logical value specifying whether to studentize the statistic. The default, `TRUE`, divides continuous derivatives and categorical finite differences by their asymptotic standard errors. This applies to individual and joint tests. `FALSE` requests the unstandardized statistic for every tested predictor. `NULL` is no longer accepted.

`random.seed` an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

`...` additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

`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 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.

Eligible individual tests with fixed bandwidths, IID residual resampling, and a supported local-polynomial or categorical-only local-constant engine evaluate bootstrap statistics in bounded native tiles. This is an internal computational optimization: it preserves bootstrap draws, random-number state, statistics, and P-values. Joint tests and configurations outside the qualified set continue through the incumbent bootstrap loop.

The categorical statistic of Racine, Hart, and Li (2006) is the unstandardized mean of squared finite-difference estimates, available with `pivot=FALSE`. The default extends this construction by studentizing categorical effects as well as continuous derivatives. For a joint test, the statistic is the mean of squared componentwise studentized effects over observations and tested predictors; it is not an inverse-covariance Wald statistic.

Studentized statistics use the residual-normalized gradient sandwich described in npreg, including its Ruppert et al. (1997) residual correction. Each bootstrap response supplies its own normalized training residuals; original-response errors are not held fixed across replications. Categorical errors use paired endpoint influence differences and retain endpoint covariance. Observed and bootstrap calculations use the same producer and information-availability rules. Eligible streamed calculations use the equivalent shared influence calculation without retaining a full influence matrix. This does not account for bandwidth or degree-selection uncertainty, or guarantee size calibration for every sample and configuration.

Studentization can reduce size distortion, but does not uniformly improve finite-sample calibration or power. In sparse designs it can be conservative and have lower power than the unstandardized statistic; rejection frequencies from an oversized unstandardized test are not size-adjusted power comparisons. The choice of pivot does not replace assessment of the fitted model and available local information.

Identical categorical endpoints contribute zero without changing the number of observations in the statistic's denominator. A native certificate of identically zero paired influence has the same interpretation; unavailable donor variance alone is not such a certificate. Unordered first-level reference rows have this property; ordered first-level rows instead use a forward difference. An entire finite observed effect vector that is exactly zero has statistic zero under the degenerate-zero convention. Because bootstrap statistics are nonnegative, the inclusive upper-tail P-value is one, without bootstrap fitting or bandwidth reselection. The bootstrap statistics need not also be zero. This applies to an individual test, or a joint test only when every tested observed effect vanishes. A zero component of a mixed joint test stays in the original denominator; it does not suppress the other components. The test has no separate bandwidth-based full-smoothing detector. However, the regression estimator's existing large-categorical-bandwidth approximation can make a categorical contrast exactly zero near the kernel's upper bandwidth bound, thereby triggering this convention. This can produce a discontinuity in the P-value at that approximation's cutoff; it is not evidence that the unapproximated contrast has vanished. The test respects the estimator option `np.largelambda`, including when the caller explicitly disables the approximation. Random draws are advanced to preserve later tests' random-number streams.

Otherwise a required zero, negative, non-finite or unavailable standard error does not define studentization. An isolated zero effect with zero standard error within a nonzero effect vector is not covered by the degenerate-zero convention unless its influence is separately certified as identically zero. The test stops with an informative diagnostic; it does not insert a variance floor, discard rows or failed replications, or substitute an unstandardized statistic. This rule applies equally to continuous and categorical effects and to observed and bootstrap responses.

When the supplied bandwidth object has `regtype="lp"`, every tested continuous coordinate must have polynomial degree at least one. A degree-zero local polynomial does not identify the derivative used by this significance test, so the function stops before fitting or bootstrap work and names the affected coordinate(s). If the degree was selected by NOMAD, refit the bandwidth object with `degree.min=1`. This restriction does not apply to an ordinary `regtype="lc"` bandwidth object.

With nearest-neighbor bandwidths, repeated continuous values can give a required radius of zero. In this case the test stops with the shared nearest-neighbor diagnostic, identifying the affected variable and the failing stage. This can occur during the unrestricted gradient or null-model evaluation, before bootstrap replications. It is not a blanket restriction on repeated values. Refit using `bwtype="fixed"` (the default) to avoid this nearest-neighbor radius limitation; see `np.kernels`.

The empirical upper-tail P-value counts bootstrap statistics equal to the observed statistic, that is, $P = B^{-1} \sum_{b=1}^{B} 1\{I_b^* \geq I\}$. Thus equality with the bootstrap critical value is non-rejection. In

particular, when smoothing makes both the observed statistic and every bootstrap statistic exactly zero, the returned P-value is one rather than zero.

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 empirical upper-tail P-values for each statistic in `In`; bootstrap values equal to the observed statistic are included in the upper tail.

`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`). An analytically skipped test has an `NA` column, not manufactured bootstrap draws.

`pivot` the requested logical pivot mode, `TRUE` or `FALSE`.

`pivot.effective`
a named logical vector recording the statistic actually used for each tested predictor. For a joint test all entries are identical.

`bootstrap.executed`
the number of bootstrap statistics actually computed for each test (zero for analytic non-rejection).

`bootstrap.reason`
the structural reason for skipping bootstrap computation, or `NA` when it was performed. The requested number of replications remains in `boot.num`.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

`npsigtest` assesses predictor relevance in nonparametric regression by comparing the fitted unrestricted regression with restrictions implied by excluding or smoothing over selected regressors. Bootstrap choices trade computational cost against finite-sample behavior.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests*, especially Sections 12.3 and 12.3.5, and Chapter 13 *Nonsmoothing Tests*, especially Section 13.3 where relevant. For the underlying conditional-mean setting, see Racine (2019), Chapter 6 *Conditional Mean Function Estimation*.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

Resampling can repeat continuous observations. With nearest-neighbor bandwidths, an inherited count can then give a zero literal radius, even when it was feasible in the preceding sample. Such a candidate is refused; the resulting error is propagated, not converted to a successful replicate or a different bootstrap method. Inspect ties and nearest-neighbor counts, or explicitly select fixed bandwidths (`bwtype="fixed"`) when this occurs.

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 `B=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"),
          B = 399,
          bw = NULL,
          boot.method = c("iid", "geom"),
          random.seed = 42,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, statistic variant, and any supplied bandwidth.

bw — a numeric (scalar) bandwidth. Defaults to plug-in (see details below).

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.

Bootstrap Controls: These arguments control bootstrap execution and reproducibility settings.

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.

B — 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: Further arguments are passed to the bandwidth-selection routines used by the test.

... — 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

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

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).

Missing observations are omitted before computing the statistic. For categorical data, declared levels, including unused levels, are retained in the pooled null sample and every bootstrap draw. The discrete entropy sum uses the full declared domain and the plug-in selector uses its category count. Kernel normalization and the existing reflection rule are unchanged; this does not define a new reflection for arbitrary nominal labels. If an observed or bootstrap reflection produces a category outside the declared domain, the call stops rather than discarding that observation. Numeric factor levels must therefore accommodate the existing reflection rule. No p-value is returned from incomplete bootstrap statistics.

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.

For numeric data using the fixed second-order Gaussian density estimator, the integration statistic uses deterministic corrected composite trapezoidal quadrature with four evaluation points per bandwidth and a domain extending eight bandwidths beyond the observed support. The density of each reflected bootstrap sample is obtained by reversing the density values on a grid symmetric about that sample's mean; the same reflection identity also avoids fitting a second density on the summation route. For summation, the selected iid or stationary bootstrap plan is generated once with the **boot** sampling law, converted to bounded chunks of multiplicities, and evaluated by a registered native routine without a sample-by-sample kernel matrix.

The accelerated route is also used when `...` contains only `bwtype="fixed"`, `ckertype="gaussian"`, and `ckerorder=2`. Other density options retain the general kernel-specific route. Generalized and adaptive nearest-neighbour bandwidths therefore continue to use the ordered, duplicate-preserving bootstrap sample and recompute realized neighbour radii rather than treating multiplicities as fixed-bandwidth weights.

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`.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

`npsymtest` tests symmetry by comparing the density or probability function of the observed data with that of the data reflected around the relevant center. The same entropy-distance logic used by the univariate density tests underlies the statistic.

For book-length background, see Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions* and Li and Racine (2007), Chapter 13 *Nonsmoothing Tests* and Chapter 12 *Model Specification Tests* where related density-feature tests are treated.

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

np.kernels, np.options, plot 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,B=99)

if (interactive()) Sys.sleep(5)

## Symmetric discrete probability distribution function

x <- factor(rbinom(n,2,.5))
npsymtest(x,B=99)

if (interactive()) Sys.sleep(5)

## Asymmetric continuous distribution function

y <- rchisq(n,df=2)
npsymtest(y,B=99)

if (interactive()) Sys.sleep(5)

## Symmetric continuous distribution function

y <- rnorm(n)
npsymtest(y,B=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,B=99,boot.method="geom")

## Symmetric time-series

yt <- ar.series(0.5,rnorm(n))
npsymtest(yt,B=99,boot.method="geom")


## End(Not run)
```

---

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,
        ...,
        se = FALSE)


## Default S3 method:
npudens(bws, tdat, ..., se = FALSE)
```

Arguments

se — Logical; compute and retain asymptotic standard errors when TRUE. The default FALSE skips uncertainty-only computation without changing the point estimates.

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

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.

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

tdat — a p-variate data frame of sample realizations (training data) used to estimate the density. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Evaluation Data: These arguments control where the fitted density is evaluated.

edat — a p-variate data frame of density evaluation points. By default, evaluation takes place on the data provided by tdat.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Additional Arguments: Further arguments are passed to npudensbw when bandwidths are computed internally, or used to interpret a numeric bws vector.

... — additional arguments supplied to npudensbw when npudens computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth selection controls such as bwmethod, bwtype, bwscaling, kernel/support controls such as ckertype, ckerorder, and ckerbound, categorical kernel controls such as ukertype and okertype, and search controls such as nmulti and scale.factor.search.lower are supplied. See npudensbw for the complete bandwidth-selection argument surface.

Details

With generalized nearest-neighbor bandwidths, query identity matters when selecting a radius: a training observation excludes its own occurrence, whereas explicit evaluation data remain external queries even if their values equal training observations. Consequently, predict(fit, newdata =

training.data) can differ from fitted(fit). This does not omit the training contribution from the fitted estimator. See np.kernels for the radius convention.

For native (non-formula) objects, named newdata columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over newdata.

For a native bandwidth object, newdata supplies edat; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

With bwtype="adaptive_nn" and se=TRUE, standard errors include the same-sample influence of each donor-centred nearest-neighbour radius. A two-sided rank-spacing pilot estimates the marginal density sum at the radius endpoints. Cross-coordinate and categorical-kernel covariance are retained in the joint observation influence. The selected NN ranks are treated as fixed; bandwidth-selection uncertainty is not included. This calculation supports Gaussian and Epanechnikov kernels of orders 2, 4, 6 and 8, and uniform kernels, without finite kernel bounds. Uniform density errors additionally include moving-support boundary influences with joint other-coordinate and categorical kernel marks. These use type-7 sample-quantile endpoints and deterministic two-sided rank-spacing pilots whose interior window grows as the two-thirds power of sample size. A strictly absent support face contributes zero; an exact probability endpoint or undefined finite-face pilot is unavailable. Finite kernel bounds (including beta kernels), extended or saturated NN ranks, and degenerate pilot spacings are not currently qualified: point estimates are retained and standard errors are NA, with one availability notice. No additional uncertainty computation is performed for se=FALSE. Requested adaptive-NN standard errors can take several times as long as the point fit because they include the donor-radius influence calculation; this cost is incurred only when standard errors are requested.

Documentation guide: see npudensbw for bandwidth selection and search controls, np.kernels for kernels, np.options for global options, and plot, plot.np for plotting options.

When bws is omitted, the formula and default methods call npudensbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a bandwidth object, npudens estimates with the stored bandwidth metadata in that object.

The beta associated density described in np.kernels is available for continuous coordinates at orders 2, 4, 6, and 8 with manual or cross-validated metric bandwidths. Such coordinates may be combined with unordered and ordered categorical coordinates. Fixed, generalized-NN, and adaptive-NN bandwidths are supported. Use ckertype="beta" and either ckerbound="fixed" with finite lower and upper bounds or ckerbound="range" for beta outer half-spacing sample bounds as defined in np.kernels. Every continuous training and evaluation value must lie within the resulting support. For example:

```
fhat <- npudens(tdat=data.frame(x), edat=data.frame(x=x.eval),
                bws=0.12, ckertype="beta", ckerorder=2,
                ckerbound="fixed", ckerlb=0, ckerub=1)
```

The beta contribution already includes its complete original-scale density normalization. No additional division by the bandwidth is performed. Beta density routes require at least one continuous coordinate and reject normal-reference selection. Cross-validated cv.ml and bounded target-coordinate cv.ls are available; the latter integrates the actual estimator and does not substitute the public beta-overlap operator. Orders 4, 6, and 8 are signed and can produce negative density estimates; no positivity repair is applied.

Because an associated beta kernel has target-dependent shape and roughness, its reported standard error does not substitute a translation-invariant $R(K)$ constant. If $K_i(x)$ denotes the full multivariate beta contribution, the implementation reports the ordinary standard error of their sample mean,

$$\widehat{\mathrm{se}}\{\hat{f}(x)\} = \sqrt{\frac{1}{n}\frac{1}{n-1}\sum_{i=1}^{n}\{K_i(x) - \hat{f}(x)\}^2},$$

with zero reported when $n = 1$. This contribution-based expression automatically respects the changing boundary shape.

When all variables are categorical, uncertainty uses the complete joint-category contributions $Z_i(x)$ produced by the selected kernel computation. With $\bar{Z}(x) = n^{-1}\sum_i Z_i(x)$, the empirical-population convention is

$$\widehat{\mathrm{se}}\{\hat{f}(x)\} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\{Z_i(x) - \bar{Z}(x)\}^2}.$$

Thus zero smoothing retains $\sqrt{\hat{p}(x)(1-\hat{p}(x))/n}$, while exactly constant contributions have zero standard error. Category-profile compression includes the original observation counts; it does not treat distinct profiles as equally frequent observations.

For ordinary kernels with mixed continuous and categorical data, fixed and generalized-nearest-neighbour standard errors retain the leading continuous- bandwidth approximation while using the squared joint categorical kernel product. Thus a categorical factor that multiplies the density by a constant also multiplies its standard error by that constant. This is not the sample variance of finite-bandwidth continuous kernel contributions. Generalized-NN errors use the existing frozen-query-radius approximation. The separate adaptive-NN influence calculation is described above.

Argument groups for bandwidth selection are documented on npudensbw. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls and numerical search controls if defaults need to be changed.

For S3 plotting help, see plot.np. You can list available plot methods with `methods("plot")`.

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

Standard errors are opt-in. To add them to an existing fit without repeating bandwidth selection, use fit <- npudens(bws=fit$bws, se=TRUE), supplying the original training data and desired evaluation data again when these cannot be recovered from the bandwidth object's call. The se() extractor reads retained errors only and gives a helpful refit message when they were not requested. Use predict(fit, se.fit=TRUE, ...) for prediction with standard errors.

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 when se=TRUE, otherwise an empty numeric vector.
se	Whether standard-error computation was requested.
log_likelihood	aggregate fitted-density log likelihood. Every represented strictly positive density estimate, including a positive subnormal value, contributes its literal logarithm. An exact zero or negative density estimate has no mathematical logarithm and is represented in this aggregate by the established computational placeholder log(DBL_MIN). Consequently this component is not a literal log likelihood if any fitted density is nonpositive.

Book And Method Pointers

The unconditional density target is $f(x)$. In the continuous fixed-bandwidth case the estimator has the usual kernel form $\hat{f}(x) = n^{-1} \sum_i K_h(X_i, x)$; with mixed data, npudens uses generalized product kernels so that the continuous, unordered, and ordered components each contribute their appropriate kernel factor. Nearest-neighbor bandwidth types change how the continuous bandwidth is indexed, not the statistical target.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.1, 1.3, 1.6, 1.8, and 1.9, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions* and Chapter 3 *Mixed-Data Probability Density and Cumulative Distribution Functions*.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

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.

Chen, S.X. (1999), "Beta kernel estimators for density functions," Computational Statistics & Data Analysis, 31(2), 131–145. doi:10.1016/S0167-9473(99)00010-9.

Igarashi, G. and Kakizawa, Y. (2020), "Higher-order bias corrections for asymmetric kernel density estimators," Journal of Nonparametric Statistics, 32(3), 716–752. doi:10.1080/10485252.2020.1770754.

See Also

np.kernels, np.options, plot, plot.np 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")
with(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...

if (interactive()) 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...

if (interactive()) Sys.sleep(5)

# Alternatively, you could use the plot() command (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems).

if (interactive()) plot(bw)

})

# 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")
with(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...

if (interactive()) 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...

if (interactive()) Sys.sleep(5)

# Alternatively, you could use the plot() command (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems).

if (interactive()) plot(bw)
```

```
})

# 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")
with(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).

if (interactive()) plot(bw, xtrim=-0.2)

})

# 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")
with(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).

if (interactive()) plot(bw, xtrim=-0.2)

})

## 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 'bandwidth'
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
          scale.factor.init.lower = 0.1,
          lbd.dir = 0.1,
          lbd.init = 0.1,
          nmulti,
          penalty.multiplier = 10,
          powell.remin = TRUE,
          bwsolver = c("powell", "mads", "mads+powell"),
```

```
            scale.init.categorical.sample = FALSE,
            scale.factor.search.lower = NULL,
            small = 1.490116e-05,
            tol = 1.490116e-04,
            transform.bounds = FALSE,
            eval.only = FALSE,
            ...,
            nomad.opts = list())

## Default S3 method:
npudensbw(dat = stop("invoked without input data 'dat'"),
            bws,
            bandwidth.compute = TRUE,
            bwmethod,
            bwscaling,
            bwtype,
            cfac.dir,
            scale.factor.init,
            ckerbound,
            ckerlb,
            ckerorder,
            ckertype,
            ckerub,
            dfac.dir,
            dfac.init,
            dfc.dir,
            ftol,
            scale.factor.init.upper,
            hbd.dir,
            hbd.init,
            initc.dir,
            initd.dir,
            invalid.penalty,
            itmax,
            lbc.dir,
            scale.factor.init.lower,
            lbd.dir,
            lbd.init,
            nmulti,
            okertype,
            penalty.multiplier,
            powell.remin,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample,
            scale.factor.search.lower = NULL,
            small,
            tol,
            transform.bounds,
```

```
            ukertype,
            ...,
            nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, and whether bandwidths are supplied or computed.

`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`. Beta kernels support both automatic cross-validation and explicit bandwidth objects.

`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.

`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.

`data`
: This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

  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.

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

`na.action`
: a function which indicates what should happen when the data contain `NA`s. 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`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

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, corresponding to exact cell-indicator kernels for categorical coordinates. 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

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

dfac.dir — stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

dfac.init — non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

hbd.dir — upper bound for direction set search for Powell's algorithm for categorical variables. See Details

hbd.init — upper bound for scale factors for categorical variables for Powell's algorithm. See Details

initd.dir — initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

lbd.dir — lower bound for direction set search for Powell's algorithm for categorical variables. See Details

lbd.init — lower bound 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

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

`cfac.dir` stretch factor for direction set search for Powell's algorithm for `numeric` variables. See Details

`dfc.dir` chi-square degrees of freedom for direction set search for Powell's algorithm for `numeric` variables. See Details

`initc.dir` initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`lbc.dir` lower bound for direction set search for Powell's algorithm for `numeric` variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

`ckerbound` character string controlling continuous-kernel support handling. Can be set as `none` (default kernel on full support), `range` (use sample min/max, except that beta kernels use the outer half-spacing rule in `np.kernels`), or `fixed` (use `ckerlb`/`ckerub`). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see `np.kernels`.

`ckerlb` numeric scalar/vector of lower bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy lower-bound validity for each continuous variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`ckerub` numeric scalar/vector of upper bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy upper-bound validity for each continuous variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

`scale.factor.init`
deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

`scale.factor.init.lower`
lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details.

`scale.factor.init.upper`
upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels.

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.

ckertype
: character string used to specify the continuous kernel type. Can be set as gaussian, epanechnikov, uniform, or beta; see Details for the finite-support beta contract. Defaults to gaussian.

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

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

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, and bounded search transformations.

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)).

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".

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

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

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

powell.remin
: logical flag controlling Powell restart-from-minimum behavior. When TRUE, the Powell-style search restarts from the located minimum for a minor gain in accuracy. Defaults to TRUE.

bwsolver
: bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

nomad.opts optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for MADS/NOMAD-backed bandwidth searches. NULL is treated as no extra NOMAD options. Non-list values and unsupported native options are rejected before solver entry.

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)).

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)).

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.

eval.only internal logical flag used by solver wrappers to evaluate an existing bandwidth candidate without running a bandwidth search. Ordinary users should leave this at its default FALSE.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

... additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via newdata does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state.

The scale.factor.* controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

scale.factor.init controls the deterministic first search start. scale.factor.init.lower and scale.factor.init.upper define the random multistart interval. scale.factor.search.lower is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is max(scale.factor.init, scale.factor.search.lower), and the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). scale.factor.init.upper must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When scale.factor.search.lower is NULL, an existing bandwidth object's stored floor is inherited when available; otherwise the package default 0.1 is used. Explicit bandwidths supplied for storage with bandwidth.compute = FALSE are not rewritten by the search floor.

For automatic fixed-bandwidth CVLS with a beta kernel, repeated complete observations at a declared endpoint require an explicitly positive resolved `scale.factor.search.lower`. The default `0.1` already satisfies this requirement. Endpoint repetitions can cause the criterion to prefer vanishing bandwidths; this is a conservative search-admissibility policy, not a claim that every such sample has a divergent criterion. Manual bandwidth evaluation and nearest-neighbor searches remain available. No positive floor is silently substituted for an explicitly requested zero.

A selected scale factor at this floor is a boundary solution of the constrained search, not evidence of an interior optimum. In particular, endpoint observations can make beta CVLS sensitive to the chosen floor. For a returned fixed-bandwidth object `bw`, inspect `bw$sfactor$x[bw$icon]` alongside `bw$scale.factor.search.lower`; the physical bandwidth is on a different scale and should not be compared directly with this dimensionless bound. Near-bound solutions merit the same sensitivity consideration, but proximity alone does not prove that the unconstrained criterion diverges. Changing the floor changes the admissible search problem.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

For `bwmethod="cv.ml"`, the ordinary contribution is based on $\log(\hat{f}_i)$. Compact-support kernels can yield zero leave-one-out density estimates, and higher-order kernels can yield negative estimates. In those cases the package optimizes a guarded CVML criterion rather than the literal log likelihood. Every represented strictly positive fitted value, including a positive subnormal value, uses the ordinary branch $-\log(\hat{f}_i)$. Let ϵ denote the positive machine floor used only to encode an invalid nonpositive contribution. Exact zero and negative values with magnitude no greater than ϵ use $-\log(\epsilon)$; negative values of larger magnitude use the smooth penalty $\log(-\hat{f}_i) - 2\log(\epsilon)$. The `guarded` count reported in bandwidth summaries records objective evaluations containing an invalid nonpositive contribution; a positive value is not guarded merely because it is small. The `bwmethod="cv.ls"` criterion remains well defined for negative kernel density estimates and is not modified by this CVML guard.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

The bandwidth-selection argument surface is easiest to read by decision group: data and existing bandwidth inputs; bandwidth criterion and representation; continuous kernel and support controls beginning with `cker*`; categorical kernel controls `ukertype` and `okertype`; and numerical search initialization, tolerances, and feasibility controls. Users who call `npudens` without a bandwidth object can pass these same bandwidth-selection controls through that function's `...`.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

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.

Exact ties can make the literal kth continuous nearest-neighbor radius zero. Such a bandwidth is undefined and is rejected; see `np.kernels` for the precise contract and remedies.

The Chen/Igarashi–Kakizawa beta family is available at orders 2, 4, 6, and 8 for fixed, generalized-NN, and adaptive-NN estimation of continuous variables on finite supports. Beta continuous coordinates may be combined with unordered and ordered categorical coordinates, which retain their selected categorical kernels. Set `ckertype = "beta"` and use either `ckerbound = "fixed"` with finite `ckerlb`/`ckerub` or `ckerbound = "range"`. Explicit bandwidth objects and automatic `cv.ml`/`cv.ls` selection are supported for continuous and mixed data; normal-reference selection is not. The bandwidth h_j is in the units of coordinate j; internally the beta precision is $\tau_j = ((b_j - a_j)/h_j)^2$. Higher orders use the signed multiscale coefficients documented in `np.kernels`. Bounded density `cv.ls` uses numerical target-coordinate quadrature of the estimator rather than the public observation-coordinate overlap. For univariate order-2 fixed-bandwidth beta `ckerbound="range"` CVLS searches with Powell, the ordinary result is also checked against the deterministic uniform-limit certificate documented in `np.kernels`. This does not replace or reduce the user's `nmulti` search, and the same check applies when automatic selection is restarted from an eligible beta range bandwidth object.

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

beta.range.certification — for an eligible automatic beta range CVLS search, a list recording the uniform-limit probe, trigger decision, optional support-width refinement, and selected source; absent otherwise

num.feval.certified — number of function evaluations attributable to that certification and optional refinement; absent otherwise

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.

Prediction follows a two-stage contract. Calling predict(bw) on the returned bandwidth object is a convenience forwarder for npudens(bws = bw) and returns a fitted npdensity object; it does not return a numeric vector of density estimates. Call fitted(fit) or predict(fit, newdata = ...) on that fitted object to obtain numeric results. The canonical workflow is

```
bw <- npudensbw(~ x, data = mydata)
fit <- predict(bw)             # equivalent to npudens(bws = bw)
fitted(fit)
predict(fit, newdata = newdata)
```

The functions summary and plot also support bandwidth objects.

Book And Method Pointers

npudensbw selects bandwidths for the unconditional density target $f(x)$ estimated by npudens. The continuous scale factors and discrete smoothing parameters are those used in the generalized product-kernel density estimator.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.3, 1.6, 1.8, and 1.9, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapters 2 and 3.

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(2, 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

np.kernels, np.options, plot, plot.np 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")
with(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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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)

})

# 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")
with(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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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)
```

```
})

## End(Not run)
```

`npudenshat` *Unconditional Density Hat Operator*

Description

Constructs the unconditional density hat operator associated with `npudens` bandwidth objects. The returned operator maps a right-hand side `y` to Hy; with `y = 1` this reproduces the fitted unconditional density.

Usage

```
npudenshat(bws,
           tdat = stop("training data 'tdat' missing"),
           edat,
           y = NULL,
           output = c("matrix", "apply"))
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

`bws`	A fitted unconditional density bandwidth object of class `"bandwidth"`.
`edat`	Optional evaluation data. If omitted, the operator is built on the training data.
`tdat`	Training data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

`output`	Either `"matrix"` for the hat matrix or `"apply"` for direct application to `y`.
`y`	Optional right-hand side vector or matrix with one row per training observation.

Details

For `output = "matrix"`, the return value is a matrix with class `c("npudenshat", "matrix")` and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For `output = "apply"`, the function returns `H y` directly. Matrix right-hand sides are applied column-wise.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class `"npudenshat"` or the applied result `H y`, depending on `output`.

Examples

```
## Not run:
data(cps71)
tx <- data.frame(age = cps71$age)

bw <- npudensbw(dat = tx, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = 1.0)

H <- npudenshat(bws = bw, tdat = tx)
dens.hat <- npudenshat(bws = bw, tdat = tx,
                       y = rep(1, nrow(tx)),
                       output = "apply")
dens.core <- fitted(npudens(bws = bw, tdat = tx))

head(cbind(dens.core, dens.hat), n = 2L)

## 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,
        ...,
        se = FALSE)


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

Arguments

se — Logical; compute and retain asymptotic standard errors when TRUE. The default FALSE skips uncertainty-only computation without changing the point estimates.

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

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.

data — This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use with(data, ...). Supplying data, including explicit data = NULL, to a native route that does not consume it is an error; see np-data-interfaces.

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.

tdat — a p-variate data frame of sample realizations (training data) used to estimate the cumulative distribution. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Evaluation Data: These arguments control where the fitted cumulative distribution is evaluated.

edat — a p-variate data frame of cumulative distribution evaluation points. By default, evaluation takes place on the data provided by tdat.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Additional Arguments: Further arguments are passed to npudistbw when bandwidths are computed internally, or used to interpret a numeric bws vector.

... — additional arguments supplied to npudistbw when npudist computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth selection controls such as bwmethod, bwtype, bwscaling, kernel/support controls such as ckertype, ckerorder, and ckerbound, categorical kernel controls such as okertype, and search controls such as nmulti, ngrid, and scale.factor.search.lower are supplied. See npudistbw for the complete bandwidth-selection argument surface.

Details

With generalized nearest-neighbor bandwidths, query identity matters when selecting a radius: a training observation excludes its own occurrence, whereas explicit evaluation data remain external queries even if their values equal training observations. Consequently, predict(fit, newdata =

training.data) can differ from fitted(fit). This does not omit the training contribution from the fitted estimator. See np.kernels for the radius convention.

For native (non-formula) objects, named newdata columns are matched to the retained variable names and reordered as needed. Extra columns are ignored; missing, duplicate or ambiguous names are errors. An unnamed matrix or vector remains positional for a single evaluation role. Explicit native evaluation arguments retain their positional contract and take precedence over newdata.

For a native bandwidth object, newdata supplies edat; explicitly supplied native evaluation arguments take precedence. This evaluates at the requested rows without repeating bandwidth selection.

With bwtype="adaptive_nn" and se=TRUE, standard errors include the same-sample influence of each donor-centred nearest-neighbour radius. A two-sided rank-spacing pilot estimates the marginal density sum at the radius endpoints. Cross-coordinate and categorical-kernel covariance are retained in the joint observation influence. The selected NN ranks are treated as fixed; bandwidth-selection uncertainty is not included. This calculation supports Gaussian and Epanechnikov kernels of orders 2, 4, 6 and 8, and uniform kernels, without finite kernel bounds. Uniform density errors additionally include moving-support boundary influences with joint other-coordinate and categorical kernel marks. These use type-7 sample-quantile endpoints and deterministic two-sided rank-spacing pilots whose interior window grows as the two-thirds power of sample size. A strictly absent support face contributes zero; an exact probability endpoint or undefined finite-face pilot is unavailable. Finite kernel bounds (including beta kernels), extended or saturated NN ranks, and degenerate pilot spacings are not currently qualified: point estimates are retained and standard errors are NA, with one availability notice. No additional uncertainty computation is performed for se=FALSE. Requested adaptive-NN standard errors can take several times as long as the point fit because they include the donor-radius influence calculation; this cost is incurred only when standard errors are requested.

For ordinary kernels with mixed continuous and ordered categorical data, fixed and generalized-nearest-neighbour standard errors retain the leading continuous-indicator approximation and the squared integrated categorical kernel product. The continuous integrated kernel remains to power one in that second moment; this is not a finite-smoothing sample-contribution variance. Purely continuous and purely categorical conventions are unchanged. Generalized-NN errors condition on the current query radii. The separate adaptive-NN influence calculation is described above.

Documentation guide: see npudistbw for bandwidth selection and search controls, np.kernels for kernels, np.options for global options, and plot, plot.np for plotting options.

When bws is omitted, the formula and default methods call npudistbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a dbandwidth object, npudist estimates with the stored bandwidth metadata in that object.

With finite continuous bounds, npudist uses the observation-centered truncated-kernel CDF documented in np.kernels. It is exactly zero and one at the declared lower and upper endpoints and has the linear uniform-CDF limit as the bandwidth diverges.

The beta associated CDF is available at orders 2, 4, 6, and 8 with manual or cross-validated metric bandwidths. Fixed, generalized-NN, and adaptive-NN modes are supported. Set ckertype="beta" and use either ckerbound="fixed" with finite ckerlb/ckerub, or ckerbound="range" for beta outer half-spacing sample bounds. The data must be continuous and all training and evaluation values must lie within the resulting supports. Categorical products and normal-reference selection are not supported.

For observation x_i, target x, support $[a, b]$, $L = b - a$, $z_i = (x_i - a)/L$, $t = (x - a)/L$, and $\tau = (L/h)^2$, the contribution is

$$I_t(1 + z_i\tau, 1 + (1 - z_i)\tau).$$

The returned estimate is the sample mean of these contributions; its reported standard error is the sample standard deviation of the multivariate contribution divided by $\sqrt{n}$. The implementation accumulates this variance in centered form to avoid cancellation near zero or one. This beta CDF is observation-centered and is not the target-variable integral of Chen's evaluation-centered beta density kernel. See `np.kernels` for the design rationale and references.

Fixed and adaptive order-2 beta CDF estimates are proper CDFs on the declared support. A generalized-NN order-2 curve retains exact endpoints and contribution-wise range but may be locally nonmonotone because its bandwidth changes with the target. Orders 4, 6, and 8 are signed and may overshoot $[0, 1]$ or be locally nonmonotone in any bandwidth mode. Returned values are not silently clipped or monotonized.

For purely ordered data, let $Z_i(x)$ be the complete multivariate CDF contribution and $\bar{Z}(x) = n^{-1} \sum_i Z_i(x)$. Standard errors use the empirical-population variance of these contributions divided by sample size,

$$\widehat{\mathrm{se}}\{\hat{F}(x)\} = \frac{1}{n} \sqrt{\sum_{i=1}^{n} \{Z_i(x) - \bar{Z}(x)\}^2}.$$

At zero smoothing this retains the usual $\sqrt{\hat{F}(x)(1 - \hat{F}(x))/n}$. At finite smoothing a contribution need not be an indicator, so the Bernoulli expression does not generally apply. Profile compression retains original observation multiplicities.

Argument groups for bandwidth selection are documented on `npudistbw`. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls, distribution-specific integral/grid controls, and numerical search controls if defaults need to be changed.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

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

Standard errors are opt-in. To add them to an existing fit without repeating bandwidth selection, use fit <- npudist(bws=fit$bws, se=TRUE), supplying the original training data and desired evaluation data again when these cannot be recovered from the bandwidth object's call. The se() extractor reads retained errors only and gives a helpful refit message when they were not requested. Use predict(fit, se.fit=TRUE, ...) for prediction with standard errors.

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 when se=TRUE, otherwise an empty numeric vector.
se	Whether standard-error computation was requested.

Book And Method Pointers

The unconditional distribution target is $F(x) = \Pr(X \leq x)$. The estimator replaces the density kernel factors by integrated distribution-kernel factors where appropriate, while retaining the same mixed-data product-kernel logic for continuous and ordered components. The returned standard errors are asymptotic standard errors for the estimated cumulative distribution values.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.4 and 1.5 for CDF estimation and bandwidth selection, together with Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions* and Chapter 3 *Mixed-Data Probability Density and Cumulative Distribution Functions*.

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.

For formula fits, character-valued missing-data policies are resolved through the `stats` model-frame owner; pass a caller-local policy as a function object rather than its name. With replacement training data, an explicit `na.action`, including `NULL`, takes precedence over the saved policy. Native fitting still requires complete rows, so a permissive policy does not guarantee the same retained sample as `stats::model.frame`.

When reusing a bandwidth object without explicit training or evaluation data, newly constructed bandwidth objects use retained training values. A fit with explicit replacement training data retains that sample in its returned bandwidth object; evaluation grids do not replace training data. Older objects without retained samples use their stored call environment when available and may require explicit training inputs after `saveRDS()` and `readRDS()`. Custom formula transformations used on new evaluation data must still be available.

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.

Chen, S.X. (1999), "Beta kernel estimators for density functions," Computational Statistics & Data Analysis, 31(2), 131–145. doi:10.1016/S0167-9473(99)00010-9.

Igarashi, G. and Kakizawa, Y. (2020), "Higher-order bias corrections for asymmetric kernel density estimators," Journal of Nonparametric Statistics, 32(3), 716–752. doi:10.1080/10485252.2020.1770754.

Mansouri, B., Rastin, A., and Allah Mombeni, H. (2024), "Beta Kernel Estimator for a Cumulative Distribution Function with Bounded Support," Journal of Sciences, Islamic Republic of Iran, **34**(4), 349–361 (volume dated 2023; accepted and DOI registered in 2024). doi:10.22059/jsciences.2024.367933.1007836.

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

np.kernels, np.options, plot, plot.np 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")
with(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()

Fhat <- 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, Fhat, 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...

if (interactive()) Sys.sleep(5)

contour(as.integer(levels(year.seq)),
        gdp.seq,
        Fhat,
        xlab="Year",
        ylab="GDP",
        main = "Cumulative Distribution Contour Plot",
        col=topo.colors(100))

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

if (interactive()) Sys.sleep(5)

# Alternatively, you could use the plot() command (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems).

if (interactive()) plot(bw)

})

# 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")
with(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()

Fhat <- 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, Fhat, 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...

if (interactive()) Sys.sleep(5)

contour(as.integer(levels(year.seq)),
        gdp.seq,
        Fhat,
        xlab="Year",
        ylab="GDP",
        main = "Cumulative Distribution Contour Plot",
        col=topo.colors(100))

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

if (interactive()) Sys.sleep(5)

# Alternatively, you could use the plot() command (<ctrl>-C will
# interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems).

if (interactive()) plot(bw)

})

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

```
library("datasets")
data("faithful")
with(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).

if (interactive()) plot(bw, xtrim=-0.2)

})

# 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")
with(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).

if (interactive()) plot(bw, xtrim=-0.2)

})

## 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 'dbandwidth'
npudistbw(dat = stop("invoked without input data 'dat'"),
          bws,
          gdat = NULL,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          do.full.integral = FALSE,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
          scale.factor.init.lower = 0.1,
          lbd.dir = 0.1,
          lbd.init = 0.1,
          memfac = 500.0,
          ngrid = 100,
          nmulti,
          penalty.multiplier = 10,
          powell.remin = TRUE,
          bwsolver = c("powell", "mads", "mads+powell"),
          scale.init.categorical.sample = FALSE,
          scale.factor.search.lower = NULL,
          small = 1.490116e-05,
          tol = 1.490116e-04,
```

```
            transform.bounds = FALSE,
            eval.only = FALSE,
            ...,
            nomad.opts = list())

## Default S3 method:
npudistbw(dat = stop("invoked without input data 'dat'"),
            bws,
            gdat,
            bandwidth.compute = TRUE,
            bwmethod,
            bwscaling,
            bwtype,
            cfac.dir,
            scale.factor.init,
            ckerbound,
            ckerlb,
            ckerorder,
            ckertype,
            ckerub,
            dfac.dir,
            dfac.init,
            dfc.dir,
            do.full.integral,
            ftol,
            scale.factor.init.upper,
            hbd.dir,
            hbd.init,
            initc.dir,
            initd.dir,
            invalid.penalty,
            itmax,
            lbc.dir,
            scale.factor.init.lower,
            lbd.dir,
            lbd.init,
            memfac,
            ngrid,
            nmulti,
            okertype,
            penalty.multiplier,
            powell.remin,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample,
            scale.factor.search.lower = NULL,
            small,
            tol,
            transform.bounds,
```

```
...,
nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, optional integration grid, and whether bandwidths are supplied or computed.

`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`.

`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.

`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 (ordered factors), or some combination thereof.

`data`
: This argument belongs to the formula interface (including supported formula-bearing bandwidth-object refits). Native calls must supply their vectors, matrices or data frames directly, or use `with(data, ...)`. Supplying `data`, including explicit `data = NULL`, to a native route that does not consume it is an error; see `np-data-interfaces`.

  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.

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

`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).

`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).

`na.action`
: a function which indicates what should happen when the data contain `NA`s. 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`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

`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

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

`dfac.dir` stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

`dfac.init` non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

`hbd.dir` upper bound for direction set search for Powell's algorithm for categorical variables. See Details

`hbd.init` upper bound for scale factors for categorical variables for Powell's algorithm. See Details

`initd.dir` initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.dir` lower bound for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.init` lower bound 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

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

cfac.dir
: stretch factor for direction set search for Powell's algorithm for numeric variables. See Details

dfc.dir
: chi-square degrees of freedom for direction set search for Powell's algorithm for numeric variables. See Details

initc.dir
: initial non-random values for direction set search for Powell's algorithm for numeric variables. See Details

lbc.dir
: lower bound for direction set search for Powell's algorithm for numeric variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

ckerbound
: character string controlling continuous-kernel support handling. Can be set as none (default kernel on full support), range (use sample min/max, except that beta kernels use the outer half-spacing rule in np.kernels), or fixed (use ckerlb/ckerub). For distribution estimation the selected continuous kernel is integrated after observation-centered truncation and normalization on the chosen support; see np.kernels.

ckerlb
: numeric scalar/vector of lower bounds for continuous variables used when ckerbound="fixed". Must satisfy lower-bound validity for each continuous variable (e.g., <= min(variable)). Use -Inf for unbounded below. See np.kernels for bounded-kernel normalization details.

ckerub
: numeric scalar/vector of upper bounds for continuous variables used when ckerbound="fixed". Must satisfy upper-bound validity for each continuous variable (e.g., >= max(variable)). Use Inf for unbounded above. See np.kernels for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

scale.factor.init
: deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to 0.5. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is max(scale.factor.init, scale.factor.search.lower). See Details.

scale.factor.init.lower
: lower endpoint for random continuous scale-factor starts. Defaults to 0.1. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). See Details.

scale.factor.init.upper
: upper endpoint for random continuous scale-factor starts. Defaults to 2.0. It must be greater than or equal to the effective lower endpoint, max(scale.factor.init.lower, scale.factor.search.lower); otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Distribution Integral And Grid Controls: These controls tune the distribution-function integral and grid calculations.

do.full.integral
: a logical value which when set as TRUE evaluates the moment-based criterion on the empirical training support. Defaults to FALSE, which uses a faster row-aligned marginal-quantile path controlled by ngrid. For multivariate data, use do.full.integral = TRUE when you want the criterion evaluated on the empirical joint support and can tolerate the additional cost.

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.

ngrid
: integer number of row-aligned marginal-quantile grid points to use when gdat is not supplied and do.full.integral = FALSE. In multivariate problems this is a one-dimensional path through the marginal quantiles, not a Cartesian product grid. Defaults to 100.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels.

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.

ckertype
: character string used to specify the continuous kernel type. Can be set as gaussian, epanechnikov, or uniform, or beta. See Details for the finite-support beta contract. Defaults to gaussian.

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

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, and bounded search transformations.

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)).

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".

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

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

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

powell.remin logical flag controlling Powell restart-from-minimum behavior. When TRUE, the Powell-style search restarts from the located minimum for a minor gain in accuracy. Defaults to TRUE.

bwsolver bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

nomad.opts optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for MADS/NOMAD-backed bandwidth searches. NULL is treated as no extra NOMAD options. Non-list values and unsupported native options are rejected before solver entry.

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)).

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)).

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.

eval.only internal logical flag used by solver wrappers to evaluate an existing bandwidth candidate without running a bandwidth search. Ordinary users should leave this at its default FALSE.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

... additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Formula bandwidth objects retain the prepared training sample (after alignment, subsetting and missing-value handling) and the resolved missing-value policy. Later refits use that sample even if constructor variables are rebound. Supplying explicit training data to the estimator replaces the training data; evaluation via `newdata` does not. Retention requires storage proportional to the sample size times the number of variables, not a pairwise weight matrix. Older saved objects without this retained state still rely on their original data bindings; rebuild their bandwidth object from the intended data before changing those bindings. User-defined transformation and missing-value functions should not rely on subsequently modified closure state.

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)`, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot`, `plot.np` for plotting options.

The bandwidth-selection argument surface is easiest to read by decision group: data, grid, and existing bandwidth inputs; bandwidth criterion and representation; continuous kernel and support controls beginning with `cker*`; ordered categorical kernel controls such as `okertype`; distribution-specific integral/grid controls such as `gdat`, `gdata`, `do.full.integral`, and `ngrid`; and numerical search initialization, tolerances, and feasibility controls. Users who call `npudist` without a bandwidth object can pass these same bandwidth-selection controls through that function's `....`

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

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.

Nearest-neighbor cross-validation excludes the held-out observation from both the kernel sum and the sample defining the neighborhood. An empirical evaluation-grid row retains its training-query identity in that deleted sample. An explicitly supplied grid is an external evaluation grid, even when its coordinates equal observed values.

Exact ties can make the literal kth continuous nearest-neighbor radius zero. Such a bandwidth is undefined and is rejected; see np.kernels for the precise contract and remedies.

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 beta associated CDF supports orders 2, 4, 6, and 8, explicit bandwidth objects and automatic cv.cdf selection, and fixed, generalized-NN, and adaptive-NN bandwidths. Set ckertype="beta" and use ckerbound="fixed" with finite ckerlb/ckerub, or ckerbound="range". Data must be continuous and lie within those supports. Normal-reference selection and categorical products are not supported. Unsupported specifications fail explicitly. The bandwidth h remains in the coordinate's original units and maps to beta precision $((b-a)/h)^2$. See np.kernels and npudist for the exact observation-centered CDF contribution. For univariate order-2 fixed-bandwidth beta ckerbound="range" CDF searches with Powell, the ordinary result is also checked against the deterministic uniform-limit certificate documented in np.kernels. This does not replace or reduce the user's nmulti search, and the same check applies when automatic selection is restarted from an eligible beta range bandwidth object.

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

beta.range.certification — for an eligible automatic beta range CDF search, a list recording the uniform-limit probe, trigger decision, optional support-width refinement, and selected source; absent otherwise

num.feval.certified — number of function evaluations attributable to that certification and optional refinement; absent otherwise

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.

Prediction follows a two-stage contract. Calling predict(bw) on the returned bandwidth object is a convenience forwarder for npudist(bws = bw) and returns a fitted npdistribution object; it does not return a numeric vector of distribution estimates. Call fitted(fit) or predict(fit, newdata = ...) on that fitted object to obtain numeric results. The canonical workflow is

```
bw <- npudistbw(~ x, data = mydata)
fit <- predict(bw)             # equivalent to npudist(bws = bw)
fitted(fit)
predict(fit, newdata = newdata)
```

The functions summary and plot also support dbandwidth objects.

Book And Method Pointers

npudistbw selects bandwidths for the unconditional distribution target $F(x) = \Pr(X \leq x)$ estimated by npudist. The selected bandwidths control the integrated kernel factors used in the mixed-data CDF estimator.

By default, data-driven bandwidth selection evaluates the CDF criterion on a row-aligned marginal-quantile path of length `ngrid`. This default is fast and useful for routine selection. To evaluate the criterion on the empirical joint support, set `do.full.integral = TRUE`; to use a user-chosen evaluation cloud or grid, supply `gdat`. For multivariate data, a Cartesian product of marginal quantiles can be supplied explicitly through `gdat` when appropriate, but it is not the default because its size grows exponentially with dimension.

Higher-order continuous CDF kernels can yield estimates that are not globally monotone and may fall outside $[0, 1]$ at some evaluation points. Returned values are not silently clamped. Fixed and adaptive order-2 beta CDFs remain monotone and in $[0, 1]$; generalized-NN order-2 curves can be locally nonmonotone because the bandwidth changes with the target. Signed beta orders 4, 6, and 8 may additionally leave $[0, 1]$ or be locally nonmonotone in any bandwidth mode.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.4 and 1.5, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapters 2 and 3.

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(2, 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

np.kernels, np.options, plot, plot.np 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")
with(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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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)

})

# 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")
with(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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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...

if (interactive()) 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)

})

## End(Not run)
```

---

npudisthat *Unconditional Distribution Hat Operator*

Description

Constructs the unconditional distribution hat operator associated with npudist bandwidth objects. The returned operator maps a right-hand side y to Hy; with y = 1 this reproduces the fitted unconditional distribution function.

Usage

```
npudisthat(bws,
           tdat = stop("training data 'tdat' missing"),
           edat,
           y = NULL,
           output = c("matrix", "apply"))
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted unconditional distribution bandwidth object of class "dbandwidth".
edat	Optional evaluation data. If omitted, the operator is built on the training data.
tdat	Training data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

output	Either "matrix" for the hat matrix or "apply" for direct application to y.
y	Optional right-hand side vector or matrix with one row per training observation.

Details

For output = "matrix", the return value is a matrix with class c("npudisthat", "matrix") and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For output = "apply", the function returns H y directly. Matrix right-hand sides are applied column-wise.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class "npudisthat" or the applied result H y, depending on output.

Examples

```
## Not run:
data(cps71)
tx <- data.frame(age = cps71$age)

bw <- npudistbw(dat = tx, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = 1.0)

H <- npudisthat(bws = bw, tdat = tx)
dist.hat <- npudisthat(bws = bw, tdat = tx,
                       y = rep(1, nrow(tx)),
                       output = "apply")
dist.core <- fitted(npudist(bws = bw, tdat = tx))

head(cbind(dist.core, dist.hat), n = 2L)

## 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 = 1,
                  proper = FALSE)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify evaluation points, observations, support bounds, and optional bandwidths.

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 $\max(X)$)
h	an optional positive bandwidth. For "beta1" and "beta2", h is dimensionless on the rescaled support $[0,1]$, not in the original data units. For "beta2", $0 < h \leq 1/4$ is required.
X	a required numeric vector of training data lying in $[a,b]$
Y	an optional numeric vector of evaluation data lying in $[a,b]$

Bandwidth Search Controls: These arguments control the boundary-corrected bandwidth search.

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", in the same units and admissible domain as h
kertype	an optional kernel specification (defaults to "gaussian1")
nmulti	number of multi-starts used when cv="numeric" (defaults to 1 for the univariate boundary problem)

Proper Density Control: This argument controls optional proper-density repair.

proper	an optional logical value indicating whether to enforce proper density and distribution function estimates over the range $[a,b]$

Details

Reported standard errors use the number of training observations in `X`, not the number of requested evaluation points.

Beta kernels require finite support bounds. Their bandwidths and automatic search starts refer to $(X - a)/(b - a)$; changing data units does not rescale `h`. The `"beta2"` bound prevents its left and right boundary regions from overlapping. Inadmissible explicit bandwidths or grid values are rejected rather than clipped.

`F` integrates the fitted density from `a` to each evaluation point, including when `a=-Inf`. It is independent of the order, multiplicity and extent of the requested evaluation grid. Continuous quadrature shares an adaptive integration representation across evaluation points; infinite supports are transformed without truncating their tails. The ordinary unbounded Gaussian case uses its analytic distribution function. Accurate CDFs on sparse evaluation grids can require substantially more work than evaluating the density at those few points. This additional work occurs after bandwidth selection, not within the cross-validation objective.

With `proper=TRUE`, the positive part of the raw fitted density is normalized by its finite positive mass over the entire declared support. The same normalization is used for `f` and `F`; the CDF is not separately clipped or rescaled on the requested grid. This replaces the former constant upward shift, which could create infinite mass on unbounded support. Failed numerical convergence or invalid normalizing mass is reported as an error. This normalization does not alter the raw `proper=FALSE` kernel definitions or cross-validation objectives.

The `"gaussian2"` kernel uses the continuous limit of its truncated- moment ratios at the support midpoint. The same stable algebra also avoids loss of these ratios when both boundary Gaussian ordinates underflow; no different kernel is substituted. At arbitrarily large bandwidths, moments are scaled to the declared support before taking ratios, retaining the Gaussian2 limit rather than replacing it by a uniform kernel.

Sampled total integrals used elsewhere in this helper collapse duplicate integration coordinates, not training observations. Endpoint correction applies only on uniform grids; irregular grids use the trapezoidal rule.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

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 `Y`, or `X` if `Y` is omitted
`F`	integral of the fitted density from `a` to each evaluation point
`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

np.kernels, np.options, plot 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$f)))
if (interactive()) plot(X,dgp,ylab="Density",ylim=ylim,type="l")
lines(X,model.g1$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)
on.exit(par(oldpar), add = TRUE)
par(mfrow=c(1,2))
ylim=range(c(model$f,model$f+1.96*model$sd.f,model$f-1.96*model$sd.f,dbeta(X,5,3)))
if (interactive()) plot(X,model$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")

if (interactive()) 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)
with(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")
})

## 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

Data, Bandwidth Inputs And Formula Interface: These arguments identify evaluation points, observations, support bounds, and optional bandwidths.

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

Additional Arguments: Further arguments are passed to npudensbw and npudens.

...	optional arguments passed to npudensbw and npudens

Details

Reported standard errors use the number of training observations in X, not the number of requested evaluation points.

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

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

`np.kernels`, `np.options`, `plot` 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)))
if (interactive()) 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)
on.exit(par(oldpar), add = TRUE)
par(mfrow=c(1,2))
ylim=range(c(model$f,model$f+1.96*model$sd.f,model$f-1.96*model$sd.f,dbeta(X,5,3)))
if (interactive()) plot(X,model$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")

if (interactive()) 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)
with(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")
})

## 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

Data, Bandwidth Inputs And Formula Interface: These arguments identify evaluation points, observations, support bounds, and optional bandwidths.

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)
h	a bandwidth (> 0)
X	a required numeric vector of training data lying in $[a, b]$
Y	an optional numeric vector of evaluation data lying in $[a, b]$

Density Bounds: These arguments set optional lower and upper density bounds used with constraint = \"density\".

lb	an optional scalar lower bound (≥ 0) to be used in conjunction with constraint="density"
ub	an optional scalar upper bound (≥ 0 and, when supplied, $\geq$ lb) to be used in conjunction with constraint="density"

Grid And Distance Controls: These arguments control grid construction, distance metric, and mass preservation.

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)
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, taking the integral at the rightmost coordinate regardless of input order. Raw density

derivatives receive the same normalization; derivatives of the log-density do not change under multiplication of the density by a constant.

`num.grid` number of additional grid points (in addition to `X` and `Y`) placed on an equi-spaced 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`)

Shape Constraint: This argument chooses the monotonicity, convexity, or log-shape constraint.

`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

Reported standard errors use the number of training observations in `X`, not the number of requested evaluation points.

Numerical integration collapses duplicate integration coordinates, not training observations, and restores the evaluation order and multiplicity. Sampled total integrals use a uniform-grid endpoint correction only when the distinct coordinates are uniformly spaced; otherwise they use the nonuniform trapezoidal rule. The `F` and `F.sc` calculations instead integrate the retained fitted density functions from the declared support lower bound, including infinite support. Integration does not add QP constraint nodes or refit the coefficients. The sampled-range normalization selected by `integral.equal` is preserved; these distribution integrals do not separately force whole-support unit mass or impose shape constraints away from the QP grid. Log-shaped fits are integrated on the density scale. Accurate absolute integration can add noticeable work when only a few evaluation points are requested; its adaptive representation is reused across points and failed convergence is reported explicitly.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

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

np.kernels, np.options, plot 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$f))
if (interactive()) plot(X,foo$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)
on.exit(par(oldpar), add = TRUE)
par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
if (interactive()) 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))
if (interactive()) 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))
if (interactive()) 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))
if (interactive()) 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))
if (interactive()) 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))

if (interactive()) 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))
if (interactive()) 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))
if (interactive()) 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)))
if (interactive()) 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))
if (interactive()) 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")

## 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,
          B = 399,
          bw.x = NULL,
          bw.y = NULL,
          random.seed = 42,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the two samples, statistic variant, and any supplied bandwidths.

bw.x, bw.y	numeric (scalar) bandwidths. Defaults to plug-in (see details below).
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 Controls: These arguments control bootstrap execution and reproducibility settings.

B	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
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.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

...	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

Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

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).

Missing observations are omitted independently from the two samples before bandwidth selection and overlap checks. Each sample must retain at least one observation; automatic bandwidth selection can require more.

For categorical data, the declared factor levels define the finite domain of the probability comparison, including levels not observed in a sample. The two samples share the union of their declared levels, retained in all bootstrap samples. Plug-in bandwidths use that category count, and the discrete statistic sums over the entire common domain, not only observed categories. Existing kernel

normalization is unchanged. Numeric ordered levels retain their numeric scale. Qualitative ordered samples must specify the same level order or an order-consistent subset; otherwise declare a common ordered level set explicitly before calling the test.

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.

For numeric data using the fixed second-order Gaussian density estimator, the integration statistic uses deterministic corrected composite trapezoidal quadrature. The integration domain extends eight bandwidths beyond the pooled observed support and uses four evaluation points per smallest bandwidth. During a default numeric iid bootstrap, resampling counts are evaluated in bounded chunks on the same quadrature grid for `integration` or on the original support for `summation`. This avoids repeated adaptive integration and repeated reconstruction of resampled data. The summation density ratios are evaluated by a registered native routine without forming a sample-by-sample kernel matrix.

The accelerated route is also used when `...` contains only options explicitly equivalent to the defaults: `bwtype="fixed"`, `ckertype="gaussian"`, and `ckerorder=2`. Other density options retain the corresponding general estimator and numerical integration route. In particular, generalized and adaptive nearest-neighbour bandwidths use duplicate-preserving resamples and recompute their realized neighbour radii; bootstrap counts are not treated as fixed-bandwidth observation weights.

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`.

Bootstrap P-Values

The bootstrap upper-tail p-value is the proportion of bootstrap statistics greater than or equal to the observed statistic. Ties count as at least as extreme, giving the conservative tie convention for discrete statistics. This is the ordinary bootstrap proportion, without an add-one correction; a reported zero is possible with a finite number of replications.

Book And Method Pointers

`npunitest` uses an entropy-distance measure to compare univariate density or probability functions. The statistic is based on the normalized Hellinger-type distance described in the entropy-test literature referenced below.

For book-length background, see Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions*, especially the entropy and density-test material. For related density and testing background, see Li and Racine (2007), Chapters 1, 12, and 13.

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 non-linear processes", Journal of Time Series Analysis, 25, 649-669.

Maasoumi, E. and J.S. Racine (2002), "Entropy and predictability of stock market returns," Journal of Econometrics, 107, 2, pp 291-312.

See Also

np.kernels, np.options, plot 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)

if (interactive()) Sys.sleep(5)

## Conduct the test for this data

npunitest(x,y,B=99)

if (interactive()) 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,B=99)

if (interactive()) 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
Fstat <- 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))
if (interactive()) plot(fx,xlim=xlim,ylim=ylim,xlab="Data",main="f(x), f(y)")
lines(fy$x,fy$y,col="red")

if (interactive()) Sys.sleep(5)

## Test for equality of log(wage) distributions

data(wage1)
with(wage1, {
lwage.male <- lwage[female=="Male"]
lwage.female <- lwage[female=="Female"]

npunitest(lwage.male,lwage.female,B=99)

if (interactive()) 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"))

})

if (interactive()) 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,B=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")
with(oecdpanel, {
summary(oecdpanel)
})
```

`plot.np` *General Purpose Plotting of Nonparametric Objects*

Description

Plotting is provided via `plot` S3 methods, which generate 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. `plot(...)` is the supported public interface.

Usage

```
## S3 method for class 'bandwidth'
plot(x, ...)

## S3 method for class 'conbandwidth'
plot(x, ...)

## S3 method for class 'plbandwidth'
plot(x, ...)

## S3 method for class 'rbandwidth'
plot(x, ...)

## S3 method for class 'scbandwidth'
plot(x, ...)

## S3 method for class 'sibandwidth'
plot(x, ...)
```

Arguments

Plot Object: This argument identifies the object to plot.

`x` an object generated by the **np** package. The most convenient inputs are fitted estimator objects returned by invocations of `npudens`, `npcdens`, `npreg`, `npconmode`, `npplreg`, `npindex`, or `npscoef`. Bandwidth objects returned by `npudensbw`, `npcdensbw`, `npregbw`, `npplregbw`, `npindexbw`, or `npscoefbw` are also supported when plotting should construct the fitted object.

Plot Options Passed Through . . . : Named options passed via `...` include:

Data and evaluation controls

`data` replacement training data for the partially linear formula-bandwidth plot route, when explicit training inputs are omitted. Other plot routes do not consume this argument and reject it, including explicit `NULL`. Refit with the intended data before plotting; see `np-data-interfaces`.

`grid.control` optional detailed grid controls created by `np_grid_control`.

`neval` the number of evaluation points used for each plotted continuous margin, defaulting to `50L`. Continuous numeric variables use this grid even when their observed values are integers or have few distinct values. Categorical margins use their levels. For `npudens`/`npudist` perspective surfaces with numeric ordered margins, the grid uses the ordered levels observed in the training data. Their one-dimensional categorical slices use all declared levels, including unused levels. A surface with two continuous margins has `neval^2` evaluation points.

`xdat`, `ydat`, `zdat` training data used when plotting bandwidth objects. `xdat` is the regressor/conditioning data, `ydat` is the response or conditional outcome data, and `zdat` is the auxiliary semiparametric covariate block when relevant.

`xq`, `yq`, `zq` quantiles used to hold non-plotted variables fixed when constructing slices.

`xtrim`, `ytrim`, `ztrim` trimming/expansion controls for the evaluation support.

Plot content controls

`coef`, `coef.index` coefficient-function plotting controls for semiparametric families where supported.

`common.scale` a logical value specifying whether multiple panels should share a common plotting scale when feasible.

`gradient.order`, `gradients` derivative plotting controls for families that support gradients/derivatives. For `npreg` local-polynomial objects, `gradient.order` requests the continuous-predictor derivative order to generate on the plot evaluation grid; if omitted the first derivative/effect is plotted. Location-scale quantile-regression `nplsqreg` objects use the same coordinate-wise local-polynomial contract while preserving their existing `tau`-indexed output; no mixed partials are generated. When a requested continuous derivative order exceeds that coordinate's fitted degree, its values remain `NA` and its plot panel remains present but empty if another derivative or categorical effect is available. Plotting stops with an informative error only when no requested component is available. Ordered and unordered categorical panels remain first-order finite differences, denoted by Δ labels. For `npconmode` objects, `gradients=TRUE` plots stored class-probability gradients/effects for the selected response level, computed by `npconmode(..., gradients=TRUE, level=...)`.

level response level to plot for class-probability displays when supported, notably plot.conmode. If omitted, the default for npconmode objects is the base/reference response level levels(y)[1]. These plots require probabilities stored by npconmode(..., probabilities=TRUE). Currently plot.conmode displays stored object-fed probability/effect payloads only; probability/effect interval bands and surface grids are not yet implemented for this route.

layout "auto" or TRUE requests automatic par(mfrow=...) management for multi-panel plots. "current" or FALSE preserves the user's current graphics layout.

output one of "plot", "data", or "both". "both" is the public spelling for the historical "plot-data" behavior.

proper, proper.control, proper.method controls for proper conditional density/distribution handling where implemented.

tau the quantile index or indices used for quantile-regression-related plotting routes when relevant. A vector of tau values overlays the corresponding conditional quantile curves and reuses the stored bandwidth object. With errors="asymptotic" or errors="bootstrap", intervals are constructed separately for each requested tau; bootstrap work is shared across tau values within each resample. For npqreg objects whose conditional-distribution bandwidths were selected with nomad=TRUE, plotting reuses the stored LP regtype.engine and selected degree.engine metadata.

Interval and bootstrap controls

alpha the nominal significance level used for interval/band construction. The default is 0.05, corresponding to a nominal variability level of $1-\alpha = 0.95$. This nominal level describes the construction; it is not by itself a claim that smoothing bias is negligible or that the displayed bounds have literal confidence coverage for the unknown target.

B the bootstrap replication count.

band one of "pmzsd", "pointwise", "bonferroni", "simultaneous", or "all". The common bandwidth-engine default is "pmzsd"; direct npcopula and npconmode plots retain their method-specific "pointwise" default. The meaning of a band name depends on whether errors="bootstrap" or errors="asymptotic"; see Details.

boot.control optional detailed bootstrap controls created by np_boot_control. This controls non-fixed bootstrap behavior, wild-bootstrap multipliers, and dependent-bootstrap block length.

bootstrap the bootstrap resampling scheme. Regression-family plots support "wild", "inid", "fixed", and "geom"; density/distribution-family plots support "inid", "fixed", and "geom".

boxplot.outliers, factor.boxplot boxplot-style bootstrap summary controls for factor/categorical displays when available.

center one of "estimate" or "bias-corrected". When bootstrap errors are requested, center="bias-corrected" centers plotted uncertainty bands or bars on a bootstrap bias-corrected estimate. The package chooses the validated correction engine for the requested bootstrap route. Wild bootstrap regression routes use the standard wild-bootstrap correction. Pairwise and dependent-bootstrap routes "inid", "fixed", and "geom" use an oversmoothed pilot correction for fixed-bandwidth estimators where that route has been implemented and validated. Unsupported combinations fail clearly rather than silently using a weak correction. For one-dimensional regression curves, the ordinary estimate remains visible and the selected bias-corrected estimate is drawn as an additional center line when available. Fixed-bandwidth regression gradient plots use the same returned gradient-bias convention described below.

errors one of "none", "bootstrap", or "asymptotic". The default is "none". Bootstrap errors can use either a symmetric normal-quantile rule based on bootstrap standard deviations or full bootstrap quantile/rank distributions, according to band. Asymptotic errors use pointwise standard errors and normal quantiles where available.

Rendering and view controls

data.overlay logical; when TRUE, overlay raw data on regression, quantile-regression, single-index, partially linear, and smooth coefficient plot surfaces and expand plot limits to include the data. For npqreg level plots the data layer is drawn first by default, so quantile curves are overlaid on top; ordered and factor conditioning variables use the natural boxplot-style display from plot(). The option is ignored for derivative/gradient plots and coefficient plots. Overlay points default to small, light markers and can be customized with standard points() arguments passed via ... (for example pch, cex, col, and bg). Default is TRUE.

legend legend control for legends drawn by the package, including vector-tau npqreg level plots and band="all" uncertainty legends. TRUE draws the default legend, FALSE, NULL, or NA suppresses it, a single character string supplies the legend position, and a list is merged into the default legend call for custom labels, placement, or styling. For plot routes with more than one package-drawn legend, named sublists such as legend=list(tau=list(x="topright"), bands=FALSE) may be used when a route distinguishes those legend roles.

data.rug logical; when TRUE, draw a rug-style support cue for the plotted coordinates. For ordinary 1D base plots this behaves like a standard rug. For eligible 2D surface plots, the base renderer draws short floor-rug ticks and renderer="rgl" draws the analogous 3D floor-rug ticks on supported surface routes. In this rollout tranche, data.rug=TRUE is implemented across the current base plot families and across supported rgl surface routes; unsupported renderer/geometry combinations still fail clearly rather than being ignored silently. On mixed base plot layouts, rug marks are drawn for continuous panels and omitted for categorical panels. This option is distinct from data.overlay: data.rug shows where observations lie in the plotted covariate support, while data.overlay is the regression-only 3D response overlay used on supported surface plots. On supported regression surfaces, both options may be used together. Default is FALSE.

perspective a logical value specifying whether bivariate continuous surfaces are displayed with persp-style plots when relevant.

phi, theta, view view controls for bivariate continuous surface plots.

render.control optional detailed rendering controls created by np_render_control, including band-versus-bar uncertainty display controls.

renderer the surface-rendering backend for eligible bivariate continuous plots. "base" uses the existing base R persp path. "rgl" is currently a fail-closed backend for supported plain 2D surface routes in npreg/npregbw, npplreg/npplregbw, npscoef/npscoefbw, npudens/npudensbw, npudist/npudistbw, npcdens/npcdensbw, and npcdist/npcdistbw. In this tranche, renderer="rgl" requires perspective=TRUE, view="fixed" and output="plot"; unsupported combinations still error rather than falling back silently. The suggested packages rgl and htmlwidgets must be installed when renderer="rgl" is requested. When drawn, the rgl surface is displayed via the active rgl backend, which on this rollout path is typically the RStudio Viewer or a browser/WebGL window rather than the base graphics plot pane. The base and rgl renderers share the same viridis palette mapper and neutral-gray three-face reference-grid default. When col is omitted, the rgl surface uses a value-based hcl.colors(...,

"viridis") gradient. When the inherited base-view defaults theta=0 and phi=20 are left unchanged, the rgl backend uses theta=0, phi=-70, and fov=55 to approximate the base perspective in the square interactive viewport. The widget automatically uses up to two backing-store pixels per CSS pixel on high-DPI displays. Axis titles retain rgl's native screen-facing text orientation while the scene is rotated. For the currently supported npreg, npplreg, and plain mean-surface npscoef surface routes, data.overlay=TRUE adds observed data as an interactive point cloud. For all currently supported rgl surface routes, errors != "none" adds interval surfaces; band="all" draws pointwise, simultaneous, and bonferroni band families with distinct, colorblind-safer colors when those bands are available for the route. Unsupervised and conditional density/distribution surface routes do not use observed-data point overlays in this tranche. For npscoef/npscoefbw, renderer="rgl" currently applies only to the plain mean-surface route; coefficient-mode plots remain on the base renderer. Common rgl display controls may be supplied directly in ... on supported routes, including surface arguments such as alpha, back, and front, and view arguments such as fov and zoom. Additional backend-specific arguments may be passed through using prefixed names in ..., namely rgl.persp3d.<arg>, rgl.view3d.<arg>, rgl.par3d.<arg>, rgl.grid3d.<arg>, and rgl.widget.<arg>, and rgl.legend3d.<arg>. For the new overlay layers, rgl.points3d.<arg> customizes observed-point overlays and rgl.surface3d.<arg> customizes interval surfaces. When band="all" is used on supported rgl routes, a matching construction-aware interactive legend is drawn using the same default labels as the base renderer, and rgl.legend3d.<arg> may be used to customize it, for example rgl.legend3d.bty="o" to restore a legend box or rgl.legend3d.magnify=1 to request the original legend texture resolution. By default, these legends follow the base-graphics uncertainty legend convention and are drawn without a surrounding box.

Graphical parameters

border, **cex.axis**, **cex.lab**, **cex.main**, **cex.sub**, **col**, **lty**, **lwd**, **main**, **sub**, **type**, **xlab**, **xlim**, **ylab**, **ylim**, **zlab**, **zlim** standard graphical controls forwarded to the underlying plotting primitives when relevant.

Reproducibility controls

random.seed an optional random seed used to make bootstrap plotting reproducible.

Additional Arguments: Further plotting controls are passed through ... to the relevant plot method.

... additional arguments supplied to control plotting behavior or passed through to underlying plotting helpers where supported. Supported named plotting controls are summarized above; standard graphical parameters are forwarded where relevant.

Details

Documentation guide: see np.kernels for kernels and np.options for global options.

The preferred public interface is plot on fitted or bandwidth objects (e.g., plot(fit) or plot(bw)).

plot 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 plot directly: plotting is handled by class-specific S3 plot methods for objects generated by the **np** package.

Visualizing one and two dimensional datasets is a straightforward process. The default behavior of `plot` is to generate a standard 2D plot to visualize univariate data, and a perspective plot for bivariate data. When visualizing higher dimensional data, `plot` 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. `plot` uses an analogous procedure to extend the upper limit of the interval.

Numeric estimator output follows R's extended-real conventions: mathematically defined one-sided limits are retained as `-Inf` or `Inf`, while unavailable or indeterminate values are returned as `NA_real_`. Plotting does not rewrite the estimator object or values returned with `output="data"`. Instead, a temporary graphics-only copy maps every non-finite coordinate to missing geometry. Consequently lines break at an infinite or unavailable point, interval segments at that point are omitted, and axis limits are computed from finite values. If a panel has no finite geometry, a finite fallback scale is used unless the user supplies `ylim`. This allows exact beta endpoint derivatives and other extended-real estimates to retain their mathematical meaning without passing invalid coordinates or limits to a graphics backend.

Plot variability-band constructions are:

```
errors       band             construction
bootstrap    pmzsd            center +/- z_(1-alpha/2) * bootstrap SD
bootstrap    pointwise        pointwise bootstrap quantiles
bootstrap    bonferroni       Bonferroni-adjusted bootstrap quantiles
bootstrap    simultaneous     rank-based joint bootstrap band
bootstrap    all              all three bootstrap quantile/rank families
asymptotic   pmzsd/pointwise  center +/- z_(1-alpha/2) * asymptotic SE
asymptotic   bonferroni       normal rule with alpha replaced by alpha/m
asymptotic   simultaneous     finite-grid Gaussian rule for supported targets
asymptotic   all              marginal families plus Gaussian where supported
```

where m is the number of evaluation points used in the plotted curve/surface ($m =$ `neval` for univariate curves, typically $m =$ `neval`^{2} for full 2D perspective surfaces).

Thus `errors="bootstrap"`, `band="pmzsd"` summarizes each bootstrap distribution by its standard deviation and imposes a symmetric normal-quantile shape on the displayed band. In contrast, bootstrap `"pointwise"` and `"bonferroni"` bands use the corresponding lower and upper empirical bootstrap quantiles separately and may be asymmetric at each evaluation point. Bootstrap `"simultaneous"` uses a joint rank construction rather than either pointwise standard deviations or pointwise tail quantiles. With `band="all"`, these distinct bootstrap families are displayed together where supported.

For asymptotic errors, "pmzsd" and "pointwise" are two names for the same symmetric normal construction. Let $T(x)$ denote the plotted functional (mean, gradient, density, distribution, etc.) and $\widehat{se}(x)$ its asymptotic standard error. Both use $T(x) \pm z_{1-\alpha/2}\widehat{se}(x)$. "bonferroni" applies the same rule with α/m in place of α.

Regression, single-index and smooth-coefficient mean and gradient curves support asymptotic Gaussian simultaneous variability bands within each panel's finite evaluation grid. This supports fixed, generalized nearest-neighbor and adaptive nearest-neighbor bandwidths with second-order Gaussian or Epanechnikov kernels and the existing LC, LL or LP specification. Numeric, unordered-factor and ordered-factor panels are supported, including mixed-data models and all-categorical smoothing under its existing local-constant specification. When there are no continuous smoothing variables, continuous-kernel settings do not restrict joint calibration. Regression, partially linear and smooth-coefficient mean surfaces support the same bands with both perspective and renderer="rgl" rendering. Gradients retain their slice plots; single-index plots retain their existing scalar-index geometry. Other continuous kernels retain a warning and unavailable simultaneous limits. With band="all", supported curves retain the pointwise and Bonferroni families and add the Gaussian simultaneous family.

For partially linear plots, a material influence reconstruction discrepancy (more than 1% of a standard error, with a roundoff allowance) signals numerical instability in the plotted fit. A warning identifies the affected points. Pointwise and Bonferroni limits are NA only at those points, and simultaneous limits are NA for the entire affected panel. The fitted curve is preserved; other panels retain their available bands.

Density and distribution plots (npudens, npudist, npcdens and npcdist) also support Gaussian simultaneous variability bands for ordinary panels and complete perspective/rgl level grids. Supported specifications include fixed, generalized nearest-neighbor and adaptive nearest-neighbor bandwidths with second-order Gaussian or Epanechnikov kernels. Conditional LC, LL and LP fits retain their selected degree and basis. Available conditional continuous derivatives and categorical first differences are supported in slice plots; higher generalized-NN derivatives retain their existing restrictions. Unconditional adaptive-NN uniform kernels are supported on regular spacing and support-face domains. Fixed continuous conditional LC specifications also support beta kernels. Proper-projected density/CDF plots and other continuous kernel specifications retain their documented simultaneous-band restrictions. Conditional quantile plots have a separate finite-grid influence construction; see npqreg for its supported level and gradient targets and conditioning assumptions.

These density-family bands use the incumbent marginal standard errors and the correlation of centered observation-level empirical influence rows. Writing D for the diagonal matrix of those standard errors and R for that correlation, the working covariance is DRD. The conditional ratio includes numerator-denominator dependence; LP uses the accepted local solve, including smooth ridge-scale and intercept compensation terms. Categorical contrasts subtract endpoint influence rows before computing covariance. Conditional NN geometry is held fixed, and unconditional generalized-NN uses its frozen-query-radius approximation; unconditional adaptive-NN includes the same-sample random-radius contribution. All three band families share the same marginal scale. Simultaneity is within the complete displayed panel/grid, not across separate panels.

The Gaussian-multiplier calibration is based on the framework developed by Victor Chernozhukov, Denis Chetverikov and Kengo Kato (2013, 2015). Related Gaussian-process and simultaneous-inference results for local regression distribution estimators are due to Matias D. Cattaneo, Michael Jansson and Xinwei Ma (2024); for local polynomial conditional density estimators, see Matias D. Cattaneo, Rajita Chandak, Michael Jansson and Xinwei Ma (2024). These works provide the

related methodological foundation; the implementations here retain this package's kernel estimators, marginal standard errors and density-specific influence contracts described above. They do not implement the distinct estimators or robust bias-correction procedures in those papers. This is finite-grid variability conditional on the selected model and smoothing specification, not bias correction, bandwidth-search uncertainty or honest continuous-domain confidence coverage. Gaussian approximation can be poor with small nearest-neighbor neighborhoods or sparse evaluation points, including grids extending to sample extremes. Increasing the number of Gaussian draws reduces simulation error but does not correct these limitations. Nonregular ANN geometry, unavailable marginal uncertainty and resource limits produce a warning and NA simultaneous limits, preserving other available bands. Unexpected calculation errors and material target reconstruction failures remain errors.

The separately named plot argument `gaussian.draws` has a fixed default of 9999 and accepts positive integer counts subject to representability and workspace limits, without a fixed one-million upper cap. It applies only to asymptotic simultaneous/all requests; `B` remains bootstrap-only. The usual `random.seed` (default 42) controls calibration without changing the caller's random-number state. A count below `ceiling(qnorm(.975)^2 * (1-alpha) / (.1^2 * alpha))` warns and is used unchanged. This is an approximate 95 percent Monte Carlo target for relative error within 10 percent in the simulated maximum's upper-tail probability. It is about 7299 draws at `alpha=0.05` and 38031 at `alpha=0.01`. The maximum's tail probability is α for either a slice or a full `neval^2` perspective/rgl grid; the number of points affects the maximum distribution and computational cost, but does not divide this tail probability. This recommendation does not guarantee critical-value accuracy or statistical coverage and is not a draw-count prescription in Chernozhukov, Chetverikov and Kato (2013). Explicit counts are never increased silently. Simulation uses bounded blocks and the bootstrap progress mechanism to report completed draws, elapsed time and estimated time remaining during interactive computation. It uses a conservative 64 MiB workspace budget; reduce `neval` or `gaussian.draws` if the workspace is too large. Regression, single-index and smooth-coefficient slice covariance calibration is limited to 512 grid points. Density-family plots switch to an observation-level factor above 512 points and budget up to 128 MiB for influence construction. This admission accounts for training size as well as grid size and LP workspaces: with the default 50 by 50 density-family surface, the factor/calibration allowance alone limits admission to about 1677 training observations; additional workspaces can lower that limit. Reducing `neval` reduces the surface grid quadratically. An over-budget request warns and retains the available marginal bands with NA simultaneous limits. Surfaces and partially linear plots instead use observation-level influence factors, avoiding a grid-by-grid covariance matrix and that grid-size limit.

Write S for the training smoother and L for its evaluation influence map. The covariance is $L\,\mathrm{diag}(z_i^2)L^{\mathsf{T}}$, where $z_i = ((I-S)y)_i/\sqrt{\sum_j(\delta_{ij}-S_{ij})^2}$. For regression derivatives, replace L by the corresponding derivative influence map; the training smoother and normalized residuals remain those of the fitted mean. Available higher-order continuous partials use the existing `gradient.order` contract. For categorical regression gradients, use the difference of the two endpoint influence rows. Unordered factors compare each level with the first retained level; ordered factors use adjacent-level differences, with the first forward difference repeated at the first level. This retains the covariance between endpoints, gives zero width for an unordered reference contrast, and calibrates jointly over all displayed levels. Simultaneity remains within each panel, not across separate panels. A simulated maximum of the absolute standardized joint Gaussian coordinates supplies one multiplier for the existing standard errors. Its simulated critical value is bounded by the analytical pointwise and Bonferroni critical values, ensuring that these three band widths remain ordered despite Monte Carlo variation. The raw simulated critical value is retained as

`critical.raw` in the calibration metadata for nonzero-variance panels. For regression, single-index and smooth-coefficient plots, if the joint calculation cannot reproduce the plotted target or standard errors, has unidentified residual information, or exceeds its workspace/grid budget, the plot warns and returns unavailable (NA) simultaneous limits. With `band="all"`, pointwise and Bonferroni intervals are retained. This can occur in sparse, locally ridged tails even after bandwidth selection. Invalid arguments, invalid covariance objects and unexpected computation errors still fail explicitly.

This Gaussian/multiplier connection follows the framework developed by Victor Chernozhukov, Denis Chetverikov and Kengo Kato (2013); see also their Gaussian comparison and anti-concentration results (2015). Conditional on the data and the fitted smoothing specification, simulating this covariance has the same Gaussian law as applying independent standard normal multipliers to the observation-level influence contributions. The covariance is assembled once per panel, so calibration requires no repeated estimator fits. These references concern the Gaussian approximation and multiplier framework; the covariance above retains this package's estimator-specific standard-error convention and does not by itself establish bias correction or honest confidence coverage.

Density-family output objects retain `band.calibration` and, for `band="all"`, `band.errors`. With conditional gradients these are lists indexed by the explanatory-variable component. Calibration records the draw count, grid size, critical value and working dependence contract.

Single-index bands additionally condition on the fitted index coefficients; gradient panels multiply the scalar-index derivative by the corresponding fitted index coefficient and its widths by that coefficient's absolute value; negative coefficients exchange lower and upper widths for asymmetric bootstrap intervals before scaling. They retain the scalar-index regression standard-error convention. Smooth-coefficient bands describe the noniterated plotted mean or existing slope-coefficient gradient selected by `coef.index`, and use fitted raw training residuals in place of the normalized residuals above. These gradients hold the smoothing coordinates fixed, including when explanatory and smoothing variables overlap; they are not total derivatives through those coordinates. The separate `coef=TRUE` display remains excluded. A joint request reports unavailable simultaneous limits if its influence map cannot reproduce the incumbent plotted target and standard errors, including incompatible locally deficient solve configurations. These three families retain their existing pointwise intervals.

Partially linear mean plots use the complete response map of the fitted estimator, including coefficient estimation. Write $R = X - S_x X$ and $R_e = X_e - L_x X$, with each column using its own nuisance smoother, and let $K = R^{+}(I - S_y)$. The evaluation map is $H = L_y + R_e K$; the training map is $S = S_y + RK$. The plot covariance is $H\,\mathrm{diag}(z_i^2)H^{\mathsf{T}}$, using the complete training map in the normalized-residual formula above. Thus the shared-observation cross-covariances are retained. This corrects the previous partially linear plot helper, which added component variances separately; its pointwise and Bonferroni plot widths consequently change as well. Coefficient estimates and their reported covariance are unchanged. With raw residuals in place of normalized residuals, this response map gives exactly the conditional covariance of independent fixed-design wild-bootstrap multipliers. Partially linear curves and surfaces use influence-factor calibration. Their nuisance influence projections reuse the scalar fit's accepted local factorization or global inverse; the fitted estimator and its ridge policy are unchanged. Reconstruction differences below one percent of a plotted standard error are treated as negligible on the interval scale. Material discrepancies raise an error.

Nearest-neighbor radii are held at the fitted design for this conditional asymptotic calculation. Training residuals retain the estimator's training evaluation convention, which can differ from explicit evaluation at the same coordinates. A blocked response-map calculation preserves that distinction without retaining a full training smoother. This can add computation for NN and partially linear

asymptotic bands; it does not repeat bandwidth selection or propagate its sampling variability.

Bandwidths and the polynomial specification are held fixed. This construction neither corrects smoothing bias nor guarantees finite-sample confidence coverage. It does not cover points between the grid or jointly cover different panels. Returned data include compact `band.calibration` metadata; `band="all"` also returns the three width matrices in `band.errors`. Surface widths follow the flattened evaluation-grid order. For a single-index gradient object, each width family is an evaluation-point by bound by coefficient array, aligned with the returned gradient columns. Factor-based calibration reports `rank=NA` because no eigendecomposition is computed.

On an eligible single-panel plot rendered by base graphics, a missing `sub` argument allows the plot method to add a concise variability subtitle after finite bounds have actually been drawn. This includes individual figures in a sequence with `options(plot.par.mfrow=FALSE)` when the active graphics layout is a full-frame single panel. Levels and supported gradients/effects follow the same rule. Categorical interval bars use interval rather than band terminology. The wording distinguishes symmetric z-times-standard-error bands from bootstrap quantile and rank constructions. For example, at the default `alpha=0.05`, labels include `"95% symmetric variability band (normal quantile × asymptotic SE)"` and `"95% pointwise bootstrap variability band (quantiles)"`. Bias-corrected bootstrap labels also identify their center.

The automatic subtitle uses the effective subtitle typography (including `cex.sub`) and only appears when an approved one-line wording fits the existing plot width and bottom-margin clearance. It does not enlarge margins or reduce the plotting region; when no wording fits, it is silently omitted. Supplying `sub`, including `sub=""` or `sub=NULL`, suppresses the automatic label without changing that route's existing handling of the supplied subtitle.

Package-managed multi-panel base figures instead receive one shared annotation at the top of each page, using existing whitespace without changing margins, panel sizes, or fonts. Simultaneous and Bonferroni wording identifies the within-panel scope; for multiple quantile curves, the scope is within each quantile curve, not joint across quantiles. An annotation is omitted if its text does not fit or clearance from a main title cannot be established. Caller-managed multi-panel layouts and partial figure regions are not labelled with a figure-wide annotation. Bootstrap boxplots and coefficient displays of plus/minus one standard error are not labelled as nominal variability intervals. The global `plot.par.mfrow` option also governs automatic layout for class-probability plots.

On these eligible base plots, with `band="all"`, default legend labels also identify the construction, for example `"Pointwise (quantiles)"` and `"Simultaneous (rank-based)"` for bootstrap bands, or `"Pointwise (z × SE)"` and `"Bonferroni (adjusted z × SE)"` for asymptotic bands. A default legend omits any family for which no finite band was drawn. Explicit user-supplied legend labels are preserved without relabelling or filtering.

These automatic annotations apply to the base renderer. The `"rgl"` renderer uses the same construction-aware default labels in its existing `band="all"` legend, but retains its sparse interactive presentation: it adds no automatic footer or subtitle, and panning, zooming, responsive sizing, and widget geometry are unchanged.

When `center="bias-corrected"` and `output="data"` (or `output="both"`) is requested, returned plot objects that support bootstrap bias correction include `bias` and `bias.corrected` components where available. The sign convention is `bias = estimate - bias.corrected`, so the corrected estimate can be recovered as `estimate - bias`. For example, regression plot objects use `mean - bias`; density objects use `dens - bias`; and distribution objects use `dist - bias`. Regression-gradient plot objects use the analogous convention `gbias = grad - gradient.bias.corrected`, so the corrected gradient can be recovered as `grad - gbias`. Two-dimensional plots do not draw

a second corrected surface in this tranche; use `output="data"` to recover the corrected surface values.

Kernel estimators are generally biased because smoothing trades variance reduction for bias. For this reason, pointwise and simultaneous error bounds centered on the fitted kernel estimate are often best interpreted as variability bounds rather than literal confidence intervals for the unknown target function. Confidence-interval interpretation requires the smoothing bias to be negligible relative to sampling variability, or to be corrected with sufficient accuracy.

When bootstrap errors are requested, `center="bias-corrected"` recenters the plotted bootstrap bounds on a method-specific Efron-Tibshirani bootstrap bias-corrected estimate, where available. This produces bounds that are closer in spirit to confidence intervals than variability bounds centered on the original biased estimate. Bias correction for nonparametric estimators is difficult, however, and no universal correction is optimal across estimators, bandwidths, supports, and data types.

For wild-bootstrap regression routes, `center="bias-corrected"` uses the standard wild-bootstrap correction, without a second oversmoothed pilot bandwidth. This follows the wild-bootstrap regression literature and the package's finite-sample evidence that an additional pilot can underadjust in this route. For supported fixed-bandwidth regression-style routes with `bootstrap="inid"`, `"fixed"`, or `"geom"`, `center="bias-corrected"` estimates the bias center from a second pilot bootstrap matrix while retaining the ordinary bootstrap matrix for variability widths. These combinations can behave differently in finite samples, so the returned `bias` and `bias.corrected` components should be inspected when bias correction is scientifically important. For supported fixed regression gradient plots, inspect `gbias`, `gradient.bias.corrected`, or route-specific gradient-bias components such as `gc1bias`; different bootstrap schemes can give materially different derivative-bias corrections.

The pilot rule is family-specific. For fixed-bandwidth regression plots, the non-wild pilot uses the full oversmoothing-rate rule:

$$g_j = h_j n^{2r/((p+2r)(p+4r))},$$

where h_j is the selected bandwidth, r is the continuous kernel order, and p is the number of continuous smoothing variables in that regression route. For ordinary second-order kernels with one continuous smoothing variable this reduces to $g_j = h_j n^{4/45}$. Categorical variables do not enter this continuous-rate exponent. Supported semiparametric regression and single-index plot routes use the square-root version of this rule.

For unconditional density and distribution plots with continuous variables, including mixed-data routes supported by the estimator, the smooth-bootstrap perturbation rule uses $g_j = h_j$, following the density smooth-bootstrap literature. Continuous variables are perturbed and categorical variables are resampled row-wise without perturbation. The smooth perturbation law is drawn from the selected ordinary continuous kernel when that kernel defines a probability law: second-order Gaussian, uniform, and second-order Epanechnikov kernels are supported. Epanechnikov perturbations use the package's standardized support $[-\sqrt{5}, \sqrt{5}]$ before multiplication by the pilot bandwidth. Higher-order Gaussian and Epanechnikov kernels remain valid estimation kernels, but their signed equivalent-kernel forms do not define ordinary random perturbations, so `center="bias-corrected"` with pair/block/geometric bootstrap fails clearly for those higher-order smooth-bootstrap perturbation routes. For conditional density/distribution plots, the continuous dependent-variable side uses the same density-style rule $g_y = h_y$; categorical dependent variables are resampled row-wise without perturbation. The conditioning-variable side uses a full regression-style pilot: continuous conditioning bandwidths are enlarged, and categorical conditioning lambdas are moved toward their admissible upper bounds without crossing them. Categorical conditioning values are drawn independently given the selected donor, using the selected categorical kernel

normalized over the declared possible levels for that donor. Factor levels and their ordering are retained. For conditional density this is

$$g_{xj} = h_{xj} n^{2r/((p+2r)(p+4r))},$$

and for conditional distribution this is

$$g_{xj} = h_{xj} n^{r/((p+r)(p+2r))},$$

where here p is the total number of continuous dependent and explanatory variables. The same conditional pilot rule is used for supported fixed-bandwidth conditional density/distribution gradient plots, including mixed-data routes. In mixed-data conditional gradients, continuous dependent variables are perturbed by the density-style rule, categorical dependent variables are resampled row-wise without perturbation, continuous conditioning variables use the conditional regression-style pilot, and categorical conditioning lambdas are moved toward their admissible upper bounds as in the corresponding level route and actually used in the categorical draws. The conditional pilot uses donor-truncated continuous probability laws when finite domain bounds are supplied, with the same normalization in its reference calculation.

Each conditional pilot resample is refitted at the original estimation bandwidths and the selected LC/LL/LP specification. The bias estimate subtracts the known conditional target of the generating pilot law from the mean pilot-bootstrap estimate, not the original fitted estimate. Conditional PDF/CDF levels, continuous derivatives and categorical first differences use this same reference contract; the endpoints of a categorical contrast share each resampled dataset. Ordinary bootstrap draws still determine variability widths. A pilot reference without support, or a derivative at a non-smooth compact-kernel knot, fails explicitly rather than silently changing the requested centering method. These are practical, evidence-backed rules-of-thumb guided by the smoothed-bootstrap and regression two-bandwidth literatures, not claims of universal optimality.

In practice, users who want confidence-style bounds should prefer bootstrap errors with `center="bias-corrected"`. The ordinary fitted curve or surface remains the default estimate because bias correction can increase variance and may have larger mean squared error than the uncorrected estimator.

Asymptotic standard errors are taken from fitted-object components such as `merr`, `gerr`, `derr`, `conderr`, and `congerr` where implemented. Ordinary regression, single-index mean/gradient and transformed-response least-squares quantile errors inherit the residual normalization described in `npreg`. This does not replace other families' covariance formulas.

Bootstrap resampling is conducted pairwise on (y, X, Z) (i.e., by resampling rows of (y, X) or (y, X, Z) as appropriate). Bootstrap method support differs by estimator family:

```
Regression-family (npreg/npindex/npscoef/npplreg): wild, inid, fixed, geom
Density/distribution-family (npudens/npudist/npcdens/npcdist/npconmode): inid, fixed, geom
```

hence `"wild"` is only available for regression-family plotting. Bias-corrected bootstrap centering is currently restricted to supported fixed-bandwidth plot routes. Non-fixed, adaptive-nearest-neighbor, and generalized-nearest-neighbor bandwidth routes fail closed for `center="bias-corrected"`. Conditional quantile `npqreg` bootstrap plots currently require `center="estimate"`; `center="bias-corrected"` fails clearly for this route. Copula `npcopula` and instrumental-variable `npregiv` bias-corrected bootstrap plot routes are not claimed in this release tranche and fail clearly when unsupported bias-corrected bootstrap controls are requested. Partially linear `npplreg` level plots support the

regression-family bootstrap methods above; gradients=TRUE is not a partially linear plot route and fails clearly.

Implementation notes for speed:

```
wild            : fast np*hat linear-operator bootstrap path
inid/fixed/geom : fast direct helper path (no internal bandwidth search)
```

For non-fixed density/distribution bootstrap, an explicit experimental approximation is available via boot.control = np_boot_control(nonfixed = "exact") or boot.control = np_boot_control(nonfixed = "frozen"). The default "exact" route recomputes the non-fixed geometry for each resample. The experimental "frozen" route reuses the original-sample non-fixed geometry throughout the bootstrap run. This option is currently implemented only for unconditional and conditional density/distribution bootstrap routes and remains off by default. For generalized/adaptive nearest-neighbor runs, "frozen" is an approximation that can alter interval/band width by holding the original-sample nearest-neighbor geometry fixed across bootstrap resamples; "exact" remains the recommended setting for production inference. This approximation can be more noticeable for conditional density/distribution plotting than for the regression-style plot families because the conditional bootstrap paths freeze both numerator and denominator nearest-neighbor geometry before recombining them. In practice, conditional distribution bands are often closer, while conditional density bands can differ more materially from "exact" under generalized/adaptive nearest-neighbor bandwidths. For smooth coefficient plots (npscoef) under non-fixed bandwidths, "exact" can also be much more expensive than "frozen" on large jobs, because the coefficient field must be recomputed for each bootstrap resample rather than reusing the original-sample geometry. This recomputation cannot in general be avoided without a more aggressive approximation: for npscoef the local weighted systems that define the coefficient vector depend on the bootstrap resample weights/counts at each evaluation point, so unlike npplreg there is no single global coefficient vector that can be updated once per draw. 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 local polynomial conditional density/distribution plotting (npcdens/npcdist with regtype="ll" or regtype="lp") and proper=TRUE, the plotted estimate is rendered proper slice-by-slice on the fixed evaluation grid: each conditional density slice is projected to be nonnegative and to integrate to one using trapezoidal quadrature weights from the evaluation y-grid, while each conditional distribution slice is projected to be monotone and bounded in $[0, 1]$. When errors="bootstrap", the bootstrap resample surfaces are computed first on that same fixed grid and then properized resample-by-resample using the same grid geometry before "pointwise", "bonferroni", "simultaneous", and "all" bands are constructed. Thus the bootstrap distribution used to form these bands is built from properized resample surfaces. The final lower/upper band surfaces are interval envelopes and are not themselves separately re-projected to satisfy the density/distribution shape constraints.

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.

The default bootstrap replication count is B=9999, including copula plots. Class-probability plots retain their default B=399. Estimator and test bootstrap defaults are separate. For pointwise percentile intervals the bare tail-resolution reference is $B \geq \lceil 2/\alpha - 1 \rceil$. For "bonferroni", "simultaneous", and "all", the warning uses $B_{\min} = \lceil 2m/\alpha - 1 \rceil$, where m is the actual within-panel evaluation count, including $m = \texttt{neval}^2$ on a full perspective grid. For rank-based simultaneous intervals this is a conservative Bonferroni reference. These are bare rank-scale floors, not integrality requirements, Monte Carlo precision targets or coverage guarantees. Meeting a floor may still leave noisy endpoints; larger B can stabilize them. At alpha=0.05, a 50-point curve has a floor of 1999, while a 50 by 50 surface has a floor of 99999. Below the floor the plot warns and uses the requested count unchanged. Bootstrap standard-deviation bands ("pmzsd") do not estimate empirical tail quantiles and are not subject to this tail-resolution warning.

Typical plotting calls:

```
## Asymptotic pointwise/bonferroni intervals
plot(fit, errors="asymptotic", band="pointwise")
plot(fit, errors="asymptotic", band="bonferroni")

## Regression-family bootstrap (wild available)
plot(fit, errors="bootstrap", bootstrap="wild")

## Density/distribution-family bootstrap (use inid/fixed/geom)
plot(fit, errors="bootstrap", bootstrap="inid")
```

Compound public plot argument names use dots, consistent with the package interface. Retired underscore spellings are not accepted. Use errors, bootstrap, B, band, alpha, center, output, data.overlay, data.rug, layout, and the control constructors documented in np_boot_control, np_grid_control, and np_render_control.

Value

Setting output will instruct plot what data to return. Option summary:
plot: instruct plot to just plot the data and return NULL
both: instruct plot 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 plot 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 plot 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.

Cao-Abad, R. (1991), "Rate of convergence for the wild bootstrap in nonparametric regression," The Annals of Statistics, 19, 2226-2231.

Cattaneo, Matias D., Rajita Chandak, Michael Jansson and Xinwei Ma (2024), "Boundary adaptive local polynomial conditional density estimators," Bernoulli, 30(4), 3193-3223. doi:10.3150/23-BEJ1711.

Cattaneo, Matias D., Michael Jansson and Xinwei Ma (2024), "Local regression distribution estimators," Journal of Econometrics, 240(2), 105074. doi:10.1016/j.jeconom.2021.01.006.

Chernozhukov, Victor, Denis Chetverikov and Kengo Kato (2013), "Gaussian approximations and multiplier bootstrap for maxima of sums of high-dimensional random vectors," The Annals of Statistics, 41(6), 2786-2819. doi:10.1214/13AOS1161.

Chernozhukov, Victor, Denis Chetverikov and Kengo Kato (2015), "Comparison and anti-concentration bounds for maxima of Gaussian random vectors," Probability Theory and Related Fields, 162, 47-70. doi:10.1007/s0044001405659.

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.

Hall, P. (1992a), "Effect of bias estimation on coverage accuracy of bootstrap confidence intervals for a probability density," The Annals of Statistics, 20, 675-694.

Hall, P. (1992b), "On bootstrap confidence intervals in nonparametric regression," The Annals of Statistics, 20, 695-711.

Hardle, W. and J.S. Marron (1991), "Bootstrap simultaneous error bars for nonparametric regression," The Annals of Statistics, 19, 778-796.

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.

Silverman, B.W. and G.A. Young (1987), "The bootstrap: to smooth or not to smooth?" Biometrika, 74, 469-479.

De Angelis, D. and G.A. Young (1992), "Smoothing the bootstrap," International Statistical Review, 60, 45-56.

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

np.kernels, np.options np.options

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")
with(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)
fit <- npudens(tdat=data, bws=bw)

# 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, plot 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(fit)

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

plot(fit, errors = "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")
fit <- npreg(bws=bw)

plot(fit)

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

Sys.sleep(5)

# Now plot the gradients...

plot(fit, 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(fit,
     errors="bootstrap",
     center="bias-corrected",
     band="simultaneous")

# EXAMPLE 3: This example demonstrates how to retrieve plotting data from
# plot(). When plot() is called with the arguments
# `output="both"' (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", package = "np")

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

bw <- npregbw(xdat=cps71$age, ydat=cps71$logwage,
              regtype="ll", bwmethod="cv.aic")
fit <- npreg(bws=bw)

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

plot.out <- plot(fit,
                 perspective=FALSE,
                 errors="bootstrap",
                 B=25,
                 output="both")

# Now grab the r1 object that plot 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...

with(cps71, 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)

# Bias-corrected bootstrap output includes the estimated bias and the
# bias-corrected center when available. The sign convention is
# bias = estimate - bias.corrected, so bias.corrected = estimate - bias.

plot.out <- plot(fit,
                 perspective=FALSE,
```

```
                  errors="bootstrap",
                  center="bias-corrected",
                  B=25,
                  output="data")

logwage.eval <- fitted(plot.out$r1)
logwage.bias <- plot.out$r1$bias
logwage.bc <- plot.out$r1$bias.corrected
stopifnot(all.equal(logwage.bc, logwage.eval - logwage.bias))

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

plot.out <- plot(fit,
                  perspective=FALSE,
                  errors="bootstrap",
                  B=25,
                  output="both",
                  gradients=TRUE)

# Now grab object that plot() 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, se = 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)

# EXAMPLE 4: Variations on local polynomial conditional density
# estimation with proper = TRUE.

data("Italy")

Italy2 <- within(Italy, {
  year <- as.numeric(as.character(year))
})
```

```
# Plot only: make the plotted surface proper on the plot evaluation grid.

fhat <- npcdens(gdp ~ year, data = Italy2,
                regtype = "lp", degree = 3, nmulti = 1)

plot(fhat, proper = TRUE)

# Fit an object whose fitted values are themselves proper.

ctrl_fit <- list(
  mode = "slice",
  apply = "fitted",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_fit <- npcdens(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_fit
)

fit_proper <- fitted(fhat_fit)
fit_raw <- fhat_fit$condens.raw

# Display the repaired and raw fitted values for cases where the raw
# fitted density is negative.

head(cbind(fit_proper, fit_raw)[which(fit_raw < 0), ])

# Predict on a common explicit y-grid for several years, and render
# those predictions proper.

g.grid <- seq(min(Italy2$gdp), max(Italy2$gdp), length.out = 200)

nd_grid <- expand.grid(
  gdp = g.grid,
  year = c(1955, 1975, 1995)
)

pred_grid <- predict(fhat, newdata = nd_grid, proper = TRUE)

# Predict on paired rows with different gdp grids by year, and still
# make the predictions proper via slice mode.

g1 <- seq(quantile(Italy2$gdp, 0.10),
          quantile(Italy2$gdp, 0.60), length.out = 60)
g2 <- seq(quantile(Italy2$gdp, 0.30),
          quantile(Italy2$gdp, 0.90), length.out = 35)
```

```
nd_slice <- rbind(
  data.frame(gdp = g1, year = rep(1960, length(g1))),
  data.frame(gdp = g2, year = rep(1985, length(g2)))
)

pred_slice <- predict(
  fhat,
  newdata = nd_slice,
  proper = TRUE,
  proper.control = list(mode = "slice")
)

# One object that carries properization for fitted values and for later
# predict() calls.

ctrl_both <- list(
  mode = "slice",
  apply = "both",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_both <- npcdens(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_both
)

fit_both <- fitted(fhat_both)
pred_both <- predict(
  fhat_both,
  newdata = nd_slice,
  proper.control = ctrl_both
)
plot(fhat_both)
})

## End(Not run)
```

---

se *Extract Standard Errors*

Description

se is a generic function which extracts standard errors from objects.

Usage

```
se(x)
```

Arguments

Object Input: Object to interrogate for standard errors.

x	an object for which the extraction of standard errors is meaningful.

Details

This function extracts stored standard errors; it does not select bandwidths, refit a model, or run a bootstrap. For estimators with a `se` fitting control, request `se = TRUE` when those errors will be needed. With `se = FALSE`, errors are deliberately absent and this accessor explains how to refit with `se = TRUE`, reusing the bandwidth object without repeating bandwidth selection. Requested but unavailable standard errors may remain `NA`; extraction does not replace or recompute them.

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), se = TRUE))
```

---

uocquantile *Compute Quantiles*

Description

`uocquantile` is a function which computes quantiles of an unordered, ordered or continuous variable `x`.

Usage

```
uocquantile(x, prob)
```

Arguments

Variable And Probability Inputs: Variable and requested probability for mixed-type quantile extraction.

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")
with(wage1, {
summary(wage1)
})
```

Index

∗ **datasets**
cps71, 5
Engel95, 7
Italy, 11
oecdpanel, 378
wage1, 401
∗ **density**
npcdenshat, 63
npudenshat, 332
∗ **distribution**
npcdisthat, 93
npudisthat, 358
∗ **hplot**
np.pairs, 28
np_plot_controls, 29
∗ **instrument**
npregiv, 252
npregivderiv, 262
∗ **misc**
np.options, 25
∗ **models**
np-data-interfaces, 16
∗ **nonparametric**
b.star, 3
gradients, 9
np.kernels, 17
npcdens, 31
npcdensbw, 46
npcdenshat, 63
npcdist, 65
npcdistbw, 78
npcdisthat, 93
npcmstest, 95
npconmode, 100
npcopula, 113
npdeneqtest, 123
npdeptest, 127
npindex, 131
npindexbw, 144
npksum, 157
nplsqreg, 169
nplsqregbw, 175
npplreg, 178
npplregbw, 187
npqcmstest, 198
npqreg, 202
npquantile, 212
npreg, 215
npregbw, 231
npreghat, 248
npscoef, 269
npscoefbw, 275
npsdeptest, 288
npseed, 291
npsemihat, 293
npsigtest, 296
npsymtest, 303
npudens, 307
npudensbw, 317
npudenshat, 332
npudist, 333
npudistbw, 342
npudisthat, 358
npuniden.boundary, 359
npuniden.reflect, 365
npuniden.sc, 367
npunitest, 373
plot.np, 379
se, 399
uocquantile, 400
∗ **package**
np, 12
∗ **regression**
nplsqreg, 169
nplsqregbw, 175
npreghat, 248
npsemihat, 293
∗ **semiparametric**

npsemihat, 293
∗ **smooth**
npkernels, 17
npuniden.boundary, 359
npuniden.reflect, 365
npuniden.sc, 367
∗ **univar**
b.star, 3
npdeptest, 127
npsdeptest, 288
npsymtest, 303
npunitest, 373
uocquantile, 400

as.data.frame, *32*, *49*, *66*, *81*, *96*, *102*, *120*, *132*, *146*, *159*, *180*, *189*, *199*, *204*, *216*, *234*, *270*, *278*, *297*, *308*, *319*, *334*, *345*
as.data.frame.npcopula *(*npcopula*)*, 113

b.star, 3, *304*
boxplot.stats, *214*
bw *(*oecdpanel*)*, 378
bw.all *(*wage1*)*, 401
bw.nrd, *62*, *92*, *275*, *328*, *354*
bw.SJ, *62*, *92*, *275*, *304*, *328*, *354*, *374*
bw.subset *(*wage1*)*, 401

cbind, *13*, *38*, *61*, *72*, *91*, *98*, *109*, *121*, *126*, *137*, *155*, *163*, *174*, *184*, *195*, *201*, *210*, *223*, *245*, *274*, *286*, *301*, *312*, *327*, *338*, *353*, *392*
coef, *11*, *135*, *154*, *182*, *273*, *400*
colMeans, *163*
contour, *118, 119*
cps71, 5

data.frame, *13*, *38*, *61*, *72*, *91*, *98*, *109*, *121*, *126*, *137*, *155*, *163*, *174*, *184*, *195*, *201*, *210*, *223*, *245*, *274*, *286*, *301*, *312*, *327*, *338*, *353*, *392*
density, *313*, *339*
dimBS, 6

ecdf, *214*
Engel95, 7
expand.grid, *118*
extendrange, *115*, *213*

factor, *13*, *35*, *59*, *70*, *89*, *161*, *182*, *193*, *219*, *242*, *305*, *311*, *325*
fitted, *11*, *37*, *71*, *108*, *120*, *135*, *173*, *182*, *209*, *222*, *259*, *267*, *273*, *311*, *337*, *400*
fitted.lsqregression *(*nplsqreg*)*, 169
fitted.npcopula *(*npcopula*)*, 113
fitted.npregiv *(*npregiv*)*, 252
fitted.npregivderiv *(*npregivderiv*)*, 262
fivenum, *214*

glm, *96*, *98*
gradients, 9, *11*, *37*, *71*, *108*, *135*, *173*, *209*, *222*, *259*, *267*, *400*
gradients.npregiv *(*npregiv*)*, 252
gradients.npregivderiv *(*npregivderiv*)*, 262

hcl.colors, *116*, *382*
hist, *62*, *92*, *275*, *328*, *354*

image, *118, 119*
Italy, 11

legend, *382*
lm, *13*, *96*, *98*
loess, *225*
lsqregressionbandwidth *(*nplsqregbw*)*, 175

na.action, *49*, *81*, *146*, *159*, *189*, *234*, *278*, *319*, *345*
na.fail, *49*, *81*, *146*, *159*, *189*, *234*, *278*, *319*, *345*
na.omit, *49*, *81*, *146*, *159*, *189*, *234*, *278*, *319*, *345*
nlm, *160*
np, *11*, 12, *27*, *35*, *49*, *59*, *70*, *81*, *89*, *146*, *161*, *181, 182*, *189*, *193*, *217*, *219*, *234*, *242*, *271*, *278*, *284*, *291, 292*, *304*, *311*, *319*, *325*, *337*, *345*, *351*, *400*
np-data-interfaces, 16
np-package *(*np*)*, 12
npkernels, *14*, *16*, 17, *25*, *27–29*, *34*, *39*, *51, 52*, *58, 59*, *62*, *68, 69*, *73*, *83, 84*, *88, 89*, *92*, *97*, *99*, *106*, *110*, *118*, *121*, *124*, *126*, *128*, *130*, *135*, *138*, *152*, *160–162*, *174*, *178*, *182*, *184*, *193*, *196*, *200, 201*, *206*, *211*, *213, 214*, *218*, *220*, *225*, *236, 237*, *241, 242*,

246, *257*, *261*, *266*, *268*, *275*, *280*, *284*, *287*, *289, 290*, *292*, *298, 299*, *304*, *306*, *309*, *313*, *321*, *324, 325*, *328*, *335, 336*, *339*, *347*, *350, 351*, *354*, *361*, *363*, *365, 366*, *369, 370*, *374*, *376*, *383*, *394*
np.options, *14*, *16, 17*, 25, *25*, *28, 29*, *34*, *39*, *58*, *62*, *69*, *73*, *88*, *92*, *97*, *99*, *106*, *110*, *118*, *121*, *124*, *126*, *128*, *130*, *135*, *138*, *152*, *160*, *174*, *178*, *182*, *184*, *193*, *196*, *200, 201*, *206*, *211*, *213, 214*, *218*, *225*, *241*, *246*, *257*, *261*, *266*, *268*, *275*, *284*, *287*, *289, 290*, *292*, *298*, *304*, *306*, *309*, *313*, *324*, *328*, *335*, *339*, *350*, *354*, *361*, *363*, *365, 366*, *369, 370*, *374*, *376*, *383*, *394*
np.pairs, 28
np_boot_control, *117*, *381*, *392*
np_boot_control *(*np_plot_controls*)*, 29
np_grid_control, *380*, *392*
np_grid_control *(*np_plot_controls*)*, 29
np_plot_controls, 29
np_render_control, *382*, *392*
np_render_control *(*np_plot_controls*)*, 29
npcdens, 31, *61*, *63*, *103*, *106*, *109*, *380*
npcdensbw, *25*, *32–35*, 46, *102–106*, *109, 110*, *292*, *380*
npcdenshat, 63
npcdist, 65, *91*, *93*, *206*
npcdistbw, *25*, *66*, *68, 69*, 78, *177*, *203–206*, *209, 210*
npcdisthat, 93
npcmstest, 95, *292*
npconmode, 100, *380*
npcopula, *17*, 113
npdeneqtest, 123, *130*, *290*, *306*, *376*
npdeptest, *126*, 127, *290*, *306*, *376*
npindex, 131, *380*
npindexbw, *132*, *134, 135*, *138*, 144, *380*
npindexhat *(*npsemihat*)*, 293
npksum, *14*, *21*, *25*, *153*, 157, *265*
nplsqreg, 169, *175*, *177, 178*
nplsqregbw, *170, 171*, *174*, 175
npplreg, 178, *194*, *380*
npplregbw, *179–182*, 187, *292*, *380*
npplreghat *(*npsemihat*)*, 293
npqcmstest, 198, *292*
npqreg, *72*, *91*, *169*, *174*, *177*, 202, *292*, *385*
npquantile, 212, *213*
npreg, *11*, *13*, *17*, *28, 29*, *36*, *54*, *61*, *70*, *85*, *91*, *137*, *148*, *154*, *171–174*, *176–178*, *182*, *184*, *194*, *196*, 215, *238*, *241, 242*, *244*, *246*, *249, 250*, *257*, *261*, *265, 266*, *268*, *273*, *281*, *286, 287*, *299*, *380*, *390*
npregbw, *20*, *25*, *97*, *99*, *160*, *176–178*, *181*, *182*, *184*, *189*, *192–194*, *196*, *200, 201*, *216–218*, *222*, *224*, 231, *287*, *292*, *297*, *380*
npreghat, *64*, *95*, 247
npregiv, *17*, 252, *268*
npregivderiv, *261*, 262
npscoef, 269, *380*
npscoefbw, *270–272*, 275, *275*, *380*
npscoefhat *(*npsemihat*)*, 293
npsdeptest, *126*, *130*, 288, *306*, *376*
npseed, 291
npsemihat, 293
npsigtest, *221*, *292*, 296
npsymtest, *126*, *130*, *290*, 303, *376*
npudens, *21*, *28, 29*, *39*, *62*, *73*, *92*, *121*, *275*, 307, *324*, *326*, *328*, *332*, *365*, *380*
npudensbw, *25*, *115, 116*, *118*, *121*, *125*, *275*, *292*, *308–310*, *313*, 317, *365*, *380*
npudenshat, 332
npudist, *21*, *62*, *92*, *121*, *213*, *275*, *328*, 333, *337*, *350–352*, *354*, *358*
npudistbw, *25*, *115, 116*, *118*, *121*, *212–214*, *334–336*, *339*, 342
npudisthat, 358
npuniden.boundary, 359, *366*, *370*
npuniden.reflect, *363*, 364
npuniden.sc, 367
npunitest, *126*, *130*, *290*, *306*, 373
numeric, *13*, *128*, *162*, *212*, *288*, *305*

oecdpanel, 378
on.exit, *27*
optim, *26*, *149–151*, *153*, *255*, *283*
options, *27*
ordered, *13*, *35*, *59*, *70*, *89*, *161*, *182*, *193*, *219*, *242*, *311*, *325*, *337*, *351*

persp, *117–119*, *382*
plot, *12*, *14*, *16, 17*, *25*, *27–29*, *34*, *37*, *39*, *58*, *60*, *62*, *69*, *71*, *73*, *88*, *91, 92*, *97*, *99*,

106, *108*, *110*, *118*, *120, 121*, *124*, *126*, *128*, *130*, *135*, *138*, *152*, *154*, *160*, *173*, *182*, *184*, *193*, *196*, *200, 201*, *206*, *209*, *211*, *213, 214*, *218*, *222*, *225*, *241*, *244*, *246*, *257*, *259*, *261*, *266–268*, *273*, *275*, *284*, *286, 287*, *289, 290*, *292*, *298*, *304*, *306*, *309*, *311*, *313*, *324*, *326*, *328*, *335*, *337*, *339*, *350*, *352*, *354*, *361*, *363*, *365, 366*, *369, 370*, *374*, *376*, *378, 379*, *383*, *402*
plot.bandwidth *(*plot.np*)*, 379
plot.conbandwidth *(*plot.np*)*, 379
plot.conmode, *103*
plot.conmode *(*npconmode*)*, 100
plot.lsqregression *(*nplsqreg*)*, 169
plot.np, *20*, *30*, *34, 35*, *39*, *58*, *62*, *69*, *73*, *88, 89*, *92*, *105*, *135*, *138*, *152*, *174*, *182*, *184*, *193*, *196*, *205, 206*, *211*, *218*, *225*, *241*, *246*, *274, 275*, *284*, *287*, *309, 310*, *313*, *324*, *328*, *335, 336*, *339*, *350*, *354*, 379
plot.npcopula *(*npcopula*)*, 113
plot.npregiv *(*npregiv*)*, 252
plot.npregivderiv *(*npregivderiv*)*, 262
plot.plbandwidth *(*plot.np*)*, 379
plot.qregression *(*npqreg*)*, 202
plot.rbandwidth *(*plot.np*)*, 379
plot.scbandwidth *(*plot.np*)*, 379
plot.sibandwidth *(*plot.np*)*, 379
predict, *37*, *71*, *106*, *108*, *120*, *135*, *154*, *173*, *182*, *206*, *209*, *222*, *273*, *286*, *311*, *337*
predict.conmode *(*npconmode*)*, 100
predict.lsqregression *(*nplsqreg*)*, 169
predict.npcopula *(*npcopula*)*, 113
predict.npreghat *(*npreghat*)*, 248
predict.qregression *(*npqreg*)*, 202
print, *173*, *259*, *267*
print.lsqregression *(*nplsqreg*)*, 169
print.lsqregressionbandwidth *(*nplsqregbw*)*, 175
print.npcopula *(*npcopula*)*, 113
print.npreghat *(*npreghat*)*, 248
print.summary.npregiv *(*npregiv*)*, 252
print.summary.npregivderiv *(*npregivderiv*)*, 262

qkde, *214*
qlogspline, *214*
quantile, *173*, *209*, *213, 214*, *401*
quantile.lsqregression *(*nplsqreg*)*, 169

residuals, *11*, *135*, *173*, *182*, *222*, *259*, *267*, *273*, *400*
residuals.lsqregression *(*nplsqreg*)*, 169
residuals.npregiv *(*npregiv*)*, 252
residuals.npregivderiv *(*npregivderiv*)*, 262
rq, *199*

se, *11*, *37*, *71*, *120*, *135*, *173*, *209*, *222*, *273*, *311*, *337*, 399
se.lsqregression *(*nplsqreg*)*, 169
se.npcopula *(*npcopula*)*, 113
set.seed, *292*
summary, *37*, *60*, *71*, *91*, *98*, *108*, *120*, *125*, *129*, *135*, *154*, *173*, *200*, *209*, *222*, *244*, *259*, *267*, *273*, *286*, *289*, *300*, *305*, *311*, *326*, *337*, *352*, *375*
summary.lsqregression *(*nplsqreg*)*, 169
summary.lsqregressionbandwidth *(*nplsqregbw*)*, 175
summary.npcopula *(*npcopula*)*, 113
summary.npregiv *(*npregiv*)*, 252
summary.npregivderiv *(*npregivderiv*)*, 262

ts, *288*

uocquantile, 400

vcov, *135*, *137*, *182*

wage1, 401