npregress kernel — Nonparametric kernel regression

Description

`npregress kernel` performs nonparametric local–linear and local–constant kernel regression. Like linear regression, nonparametric regression models the mean of the outcome conditional on the covariates, but unlike linear regression, it makes no assumptions about the functional form of the relationship between the outcome and the covariates. `npregress kernel` may be used to model the mean of a continuous, count, or binary outcome.

Quick start

Nonparametric regression of `y` on `x` and discrete covariate `a` using the Epanechnikov kernel for `x` and the Li–Racine kernel for `a`

```
npregress kernel y x i.a
```

Same as above, but use 500 replications and compute bootstrap standard errors and percentile confidence intervals

```
npregress kernel y x i.a, reps(500)
```

Same as above, but use a Gaussian kernel for `x`

```
npregress kernel y x i.a, reps(500) kernel(gaussian)
```

Same as above, but use the improved AIC to find the optimal bandwidth

```
npregress kernel y x i.a, reps(500) kernel(gaussian) imaic
```

Same as above, but additionally specify that only the mean of the outcome be computed

```
npregress kernel y x i.a, reps(500) kernel(gaussian) imaic noderivatives
```

Specify `h` as the vector of bandwidths

```
npregress kernel y x i.a, bwidth(h)
```

Menu

Statistics > Nonparametric analysis > Nonparametric kernel regression

Syntax

`npregress kernel` *depvar indepvars* [*if*] [*in*] [, *options*]

options	Description
Model	
`estimator(linear | constant)`	use the local-linear or local-constant kernel estimator
`kernel(`*kernel*`)`	kernel density function for continuous covariates
`dkernel(`*dkernel*`)`	kernel density function for discrete covariates
`predict(`*prspec*`)`	store predicted values of the mean and derivatives using variable names specified in *prspec*
`noderivatives`	suppress derivative computation
`imaic`	use improved AIC instead of cross-validation to compute optimal bandwidth
`unidentsample(`*newvar*`)`	specify name of variable that marks identification problems
Bandwidth	
`bwidth(`*specs*`)`	specify kernel bandwidth for all predictions
`meanbwidth(`*specs*`)`	specify kernel bandwidth for the mean
`derivbwidth(`*specs*`)`	specify kernel bandwidth for the derivatives
SE	
* `vce(`*vcetype*`)`	*vcetype* may be `none` or `bootstrap`
`reps(#)`	equivalent to `vce(bootstrap, reps(#))`
`seed(#)`	set random-number seed to #; must also specify `reps(#)`
`bwreplace`	vary bandwidth with each bootstrap replication; seldom used
Reporting	
`level(#)`	set confidence level; default is `level(95)`
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
`citype(`*citype*`)`	method to compute bootstrap confidence intervals; default is `citype(percentile)`
Maximization	
maximize_options	control the maximization process
`coeflegend`	display legend instead of statistics

indepvars may contain factor variables; see **[U] 11.4.3 Factor variables**.

`bootstrap`, `by`, `collect`, and `jackknife` are allowed; see **[U] 11.1.10 Prefix commands**.

* `vce(bootstrap)` reports percentile confidence intervals instead of the normal-based confidence intervals reported when `vce(bootstrap)` is specified with other estimation commands.

`coeflegend` does not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

kernel	Description
`epanechnikov`	Epanechnikov kernel function; the default
`epan2`	alternative Epanechnikov kernel function
`biweight`	biweight kernel function
`cosine`	cosine trace kernel function
`gaussian`	Gaussian kernel function
`parzen`	Parzen kernel function
`rectangle`	rectangle kernel function
`triangle`	triangle kernel function

dkernel	Description
`liracine`	Li–Racine kernel function; the default
`cellmean`	cell means kernel function

citype	Description
`percentile`	percentile confidence intervals; the default
`bc`	bias-corrected confidence intervals
`normal`	normal-based confidence intervals

Options

Model

`estimator(linear | constant)` specifies whether the local-constant or local-linear kernel estimator should be used. The default is `estimator(linear)`.

`kernel(`*kernel*`)` specifies the kernel density function for continuous covariates for use in calculating the local-constant or local-linear estimator. The default is `kernel(epanechnikov)`.

`dkernel(`*dkernel*`)` specifies the kernel density function for discrete covariates for use in calculating the local-constant or local-linear estimator. The default is `dkernel(liracine)`; see Methods and formulas for details on the Li–Racine kernel. When `dkernel(cellmean)` is specified, discrete covariates are weighted by their cell means.

`predict(`*prspec*`)` specifies that `npregress kernel` store the predicted values for the mean and derivatives of the mean with the specified names. *prspec* is the following:

`predict(`*varlist* | *stub*`*` [`, replace noderivatives`]`)`

The option takes a variable list or a *stub*. The first variable name corresponds to the predicted outcome mean. The second name corresponds to the derivatives of the mean. There is one derivative for each *indepvar*.

When `replace` is used, variables with the names in *varlist* or *stub*`*` are replaced by those in the new computation. If `noderivatives` is specified, only a variable for the mean is created. This will increase computation speed but will add to the computation burden if you want to obtain marginal effects after estimation.

`noderivatives` suppresses the computation of the derivatives. In this case, only the mean function is computed.

`imaic` specifies to use the improved AIC instead of cross-validation to compute optimal bandwidths.

`unidentsample(`*newvar*`)` specifies the name of a variable that is 1 if the observation violates the model identification assumptions and is 0 otherwise. By default, this variable is a system variable (`_unident_sample`).

`npregress kernel` computes a weighted regression for each observation in our data. An observation violates identification assumptions if the regression cannot be performed at that point. The regression formula, which is discussed in detail in *Methods and formulas*, is given by

$$\widehat{\gamma} = (\mathbf{Z}'\mathbf{W}\mathbf{Z})^{-1}\,\mathbf{Z}'\mathbf{W}\mathbf{y}$$

`npregress kernel` verifies that the matrix $(\mathbf{Z}'\mathbf{W}\mathbf{Z})$ is full rank for each observation to determine identification. Identification problems commonly arise when the bandwidth is too small, resulting in too few observations within a bandwidth. Independent variables that are collinear within the bandwidth can also cause a problem with identification at that point.

Observations that violate identification assumptions are reported as missing for the predicted means and derivatives.

Bandwidth

`bwidth(`*specs*`)` specifies the half-width of the kernel at each point for the computation of the mean and the derivatives of the mean function. If no bandwidth is specified, one is chosen by minimizing the integrated mean squared error of the prediction.

specs specifies bandwidths for the mean and derivative for each *indepvar* in one of three ways: by specifying the name of a vector containing the bandwidths (for example, `bwidth(H)`, where `H` is a properly labeled vector); by specifying the equation and coefficient names with the corresponding values (for example, `bwidth(Mean:x1=0.5 Effect:x1=0.9)`); or by specifying a list of values for the means, standard errors, and derivatives for *indepvars* given in the order of the corresponding *indepvars* and specifying the `copy` suboption (for example, `bwidth(0.5 0.9, copy)`).

`skip` specifies that any parameters found in the specified vector that are not also found in the model be ignored. The default action is to issue an error message.

`copy` specifies that the list of values or the vector be copied into the bandwidth vector by position rather than by name.

`meanbwidth(`*specs*`)` specifies the half-width of the kernel at each point for the computation of the mean function. If no bandwidth is specified, one is chosen by minimizing the integrated mean squared error of the prediction. For details on how to specify the bandwidth, see the description of `bwidth()`, above.

`derivbwidth(`*specs*`)` specifies the half-width of the kernel at each point for the computation of the derivatives of the mean. If no bandwidth is specified, one is chosen by minimizing the integrated mean squared error of the prediction. For details on how to specify the bandwidth, see the description of `bwidth()`, above.

SE

`vce(vcetype)` specifies the type of standard error reported, which may be either that no standard errors are reported (`none`; the default) or that bootstrap standard errors are reported (`bootstrap`); see [R] ***vce_option***.

We recommend that you select the number of replications using `reps(#)` instead of specifying `vce(bootstrap)`, which defaults to 50 replications. Be aware that the number of replications needed to produce good estimates of the standard errors varies depending on the problem.

When `vce(bootstrap)` is specified, `npregress kernel` reports percentile confidence intervals as recommended by Cattaneo and Jansson (2018) instead of reporting the normal-based confidence intervals that are reported when `vce(bootstrap)` is specified with other commands. Other types of confidence intervals can be obtained by using the `citype(`*citype*`)` option.

`reps(#)` specifies the number of bootstrap replications to be performed. Specifying this option is equivalent to specifying `vce(bootstrap, reps(#))`.

`seed(#)` sets the random-number seed. You must specify `reps(#)` with `seed(#)`.

`bwreplace` computes a different bandwidth for each bootstrap replication. The default is to compute the bandwidth once and keep it fixed for each bootstrap replication. This option is seldom used.

Reporting

`level(#)`; see [R] **Estimation options**.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(`*%fmt*`)`, `pformat(`*%fmt*`)`, `sformat(`*%fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

`citype(`*citype*`)` specifies the type of confidence interval to be computed. By default, bootstrap percentile confidence intervals are reported as recommended by Cattaneo and Jansson (2018). *citype* may be one of `percentile`, `bc`, or `normal`.

Maximization

maximize_options: `iterate(#)`, `[no]log`, `trace`, `showstep`, `tolerance(#)`, `ltolerance(#)`, `from(`*init_specs*`)`; see [R] **Maximize**. These options are seldom used.

The following option is available with `npregress kernel` but is not shown in the dialog box:

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

This entry assumes that you are already familiar with nonparametric regression. For an introduction to the nonparametric kernel regression methods used in `npregress kernel`, see [R] **npregress intro**.

Remarks are presented under the following headings:

Overview

`npregress kernel` implements local-constant and local-linear regression. The covariates may be continuous or discrete. You can use `npregress kernel` to nonparametrically estimate a conditional mean. `npregress kernel` also allows you to estimate covariate effects after estimation and, in models with one covariate, to plot the mean function by using `npgraph` after estimation.

The word “nonparametric” refers to the fact that the parameter of interest, the mean as a function of the covariates, is given by the unknown function $g(\mathbf{x}_i)$, which is an element of an infinite-dimensional space of functions. In contrast, in a parametric model, the mean for a given value of the covariates, $E(y_i|\mathbf{x}_i) = f(\mathbf{x}_i, \boldsymbol{\beta})$, is a known function that is fully characterized by the parameter of interest, $\boldsymbol{\beta}$, which is a finite-dimensional real vector (Shao 2003).

The regression model of outcome y_i given the k-dimensional vector of covariates $\mathbf{x}_i$ is given by

$$y_i = g(\mathbf{x}_i) + \varepsilon_i \tag{1}$$

$$E(\varepsilon_i|\mathbf{x}_i) = 0 \tag{2}$$

where ε_i is the error term. The covariates may include discrete and continuous variables. Equations (1) and (2) imply that

$$E(y_i|\mathbf{x}_i) = g(\mathbf{x}_i)$$

Once we account for the information in the covariates, the error term provides no information about the mean of our outcome. The conditional mean function is therefore given by $g(\mathbf{x}_i)$. By estimating $E(y_i|\mathbf{x}_i = \mathbf{x})$ for all points $\mathbf{x}$ in our data, we obtain an estimate of $E(y_i|\mathbf{x}_i)$.

`npregress kernel`, by default, estimates a local-linear regression. Local-linear regression estimates a regression for a subset of observations for each point in our data. See Fan and Gijbels (1996) for a good reference on local-linear regression. Local-linear regression, for each point $\mathbf{x}$, solves the minimization problem given by

$$\min_{\boldsymbol{\gamma}} \sum_{i=1}^{n} \left\{y_i - \gamma_0 - \boldsymbol{\gamma}_1' (\mathbf{x}_i - \mathbf{x})\right\}^2 K(\mathbf{x}_i, \mathbf{x}, \mathbf{h}) \tag{3}$$

where $\boldsymbol{\gamma} = (\gamma_0, \boldsymbol{\gamma}_1')'$.

Equation (3) and its solution are similar to parametric ordinary least squares. The slope and the constant in (3), however, have a different interpretation. The constant in (3), γ_0, is the conditional mean at a specific point $\mathbf{x}$. The slope parameter, $\boldsymbol{\gamma}_1$, is the derivative of the mean function with respect to $\mathbf{x}$. The solution to this least-squares problem gives us the mean function and its derivative for each one of the elements of $\mathbf{x}$. Repeating this optimization for each point $\mathbf{x}$ gives us the entire mean function and its derivatives.

Another difference between (3) and the minimization problem of parametric ordinary least squares is how the optimization is weighted. The weights are given by the kernel function $K(\mathbf{x}_i, \mathbf{x}, \mathbf{h})$. The kernel function assigns weights to observations $\mathbf{x}_i$ based on how much they differ from $\mathbf{x}$ and based on the bandwidth, $\mathbf{h}$. The smaller $\mathbf{h}$ is, the larger the weight assigned to points between $\mathbf{x}_i$ and $\mathbf{x}$.

The bandwidth also determines the bias and variance of the mean function estimator. `npregress kernel` selects the bandwidth using cross-validation, as suggested by Li and Racine (2004), or if the `imaic` option is specified, with the improved AIC proposed by Hurvich, Simonoff, and Tsai (1998). Both methods minimize the tradeoff between bias and variance.

`npregress kernel` computes a conditional mean for each observation in the data and, for each one of these computations, verifies whether identification conditions are fulfilled. The observations for which the regression identification assumptions are not satisfied are dropped from the estimation sample. Additionally, whenever there is a violation of the identification assumption, `npregress kernel` generates a system variable or a variable with a name provided in `noidsample(`*newvar*`)`. This variable is 1 for observations violating the identification assumption and is 0 otherwise. `npregress kernel` also issues a warning, letting you know the number of observations for which the identification assumption is not satisfied.

Estimation and effects

The output of `npregress kernel` reports averages of the mean function and the effects of the mean function. An average effect from nonparametric regression may be either 1) an average marginal effect, in the case of the mean of derivatives for continuous covariates or 2) the mean of contrasts for discrete covariates.

Some well-established literature estimates these average effects directly and uses an optimal bandwidth for this computation; see Powell, Stock, and Stoker (1989) and Powell and Stoker (1996). By taking averages of the local-linear estimates, `npregress kernel` is more in line with the approach in Li, Lu, and Ullah (2003). Intuitively, choosing the optimal bandwidth for the derivative produces a more efficient estimator than using the bandwidth that is optimal for the function. Both estimators are consistent for the average effect. Cattaneo and Jansson (2018) formally justify the average effect using the function-optimal bandwidth.

`npregress kernel` also reports an approximation of $n|\mathbf{h}|$ as the expected kernel observations. This statistic rounds the product of the continuous kernel bandwidth values and the number of observations used for estimation. For instance, if the estimation sample was 500 and the bandwidth was 0.246, the expected kernel observations would be 123 ($= 500 \times 0.246$). The expected kernel observation number of 123 tells us that, on average, 123 observations are used to compute each one of the 500 regressions performed by `npregress kernel`.

▷ Example 1: Nonparametric regression estimation and graphing

`dui.dta` contains information about the number of monthly drunk driving citations in a local jurisdiction (`citations`). Suppose we want to know the effect of increasing fines on the number of citations. Because `citations` is a count variable, we could consider fitting the model with `poisson` or `nbreg`. However, both of these estimators make assumptions about the distribution of the data. If these assumptions are not true, we will obtain inconsistent estimates.

By using `npregress kernel`, we do not have to make any assumptions about how `citations` is distributed. We use `npregress kernel` to estimate the mean of `citations` as a function of the value of the fines imposed for drunk driving (`fines`).

```
. use https://www.stata-press.com/data/r19/dui
(Fictional data on monthly drunk driving citations)

. npregress kernel citations fines

Computing mean function

Minimizing cross-validation function:

Iteration 0:  Cross-validation criterion =  35.478784
Iteration 1:  Cross-validation criterion =  4.0147129
Iteration 2:  Cross-validation criterion =  4.0104176
Iteration 3:  Cross-validation criterion =  4.0104176
Iteration 4:  Cross-validation criterion =  4.0104176
Iteration 5:  Cross-validation criterion =  4.0104176
Iteration 6:  Cross-validation criterion =  4.0104006

Computing optimal derivative bandwidth

Iteration 0:  Cross-validation criterion =  6.1648059
Iteration 1:  Cross-validation criterion =  4.3597488
Iteration 2:  Cross-validation criterion =  4.3597488
Iteration 3:  Cross-validation criterion =  4.3597488
Iteration 4:  Cross-validation criterion =  4.3597488
Iteration 5:  Cross-validation criterion =  4.3597488
Iteration 6:  Cross-validation criterion =  4.3595842
Iteration 7:  Cross-validation criterion =  4.3594713
Iteration 8:  Cross-validation criterion =  4.3594713

Bandwidth
-------------+----------------------
             |      Mean     Effect
-------------+----------------------
       fines |  .5631079    .924924
-------------+----------------------

Local-linear regression                    Number of obs      =         500
Kernel   : epanechnikov                    E(Kernel obs)      =         282
Bandwidth: cross-validation                R-squared          =      0.4380
-------------+--------------------------------------------------------------
   citations |   Estimate
-------------+--------------------------------------------------------------
Mean         |
   citations |   22.33999
-------------+--------------------------------------------------------------
Effect       |
       fines |  -7.692388
-------------+--------------------------------------------------------------
Note: Effect estimates are averages of derivatives.
Note: You may compute standard errors using vce(bootstrap) or reps().
```

The first table displays the bandwidths used to estimate the mean function and the derivative of the mean function. Each of these bandwidths is estimated by minimizing a function that trades off bias and variance; the corresponding iteration logs are displayed also. The expected number of observations used to estimate the mean function at each point is reported in `E(Kernel obs)` as 282.

Unlike other estimation commands, npregress kernel does not report standard errors, test statistics, and confidence intervals by default. In example 2, we demonstrate how to obtain these statistics and further discuss the output.

◁

▷ Example 2: Bootstrapping standard errors

We can estimate the standard errors by using the bootstrap; see Cattaneo and Jansson (2018) for formal results. We use the reps(400) option, which is equivalent to vce(bootstrap, reps(400)) and specifies that 400 bootstrap replications be used instead of the default 50 replications that are used when we specify vce(bootstrap).

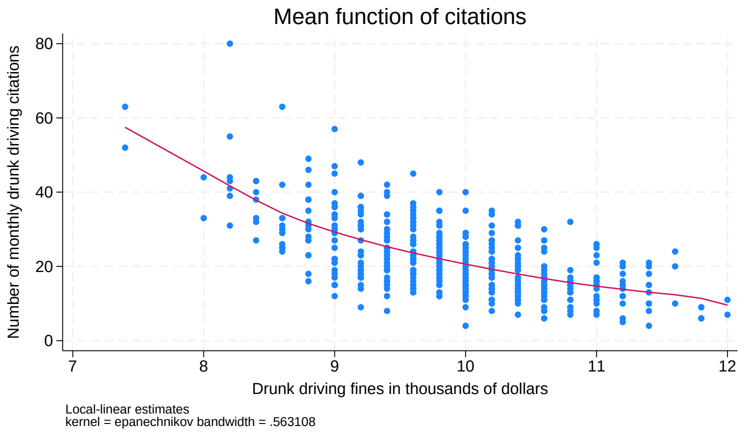
Each estimation problem requires a different number of replications to produce good estimates of the standard errors. In example 3, we explain how we decided to use 400 replications. Note that nonparametric estimation and the bootstrap are computationally intensive, so running this example and others that compute bootstrap standard errors will take a while.

```
. npregress kernel citations fines, reps(400) seed(12)
(running npregress on estimation sample)

Bootstrap replications (400): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170.......
> .180.........190.........200.........210.........220.........230.........240.
> ........250.........260.........270.........280.........290.........300......
> ...310.........320.........330.........340.........350.........360.........37
> 0.........380.........390.........400 done

Bandwidth
─────────────┬──────────────────────
             │      Mean     Effect
─────────────┼──────────────────────
       fines │  .5631079    .924924
─────────────┴──────────────────────

Local-linear regression                    Number of obs      =          500
Kernel   : epanechnikov                    E(Kernel obs)      =          282
Bandwidth: cross-validation                R-squared          =       0.4380
─────────────┬──────────────────────────────────────────────────────────────
             │   Observed   Bootstrap                         Percentile
   citations │   estimate   std. err.      z    P>|z|     [95% conf. interval]
─────────────┼──────────────────────────────────────────────────────────────
Mean         │
   citations │   22.33999   .4588298    48.69   0.000     21.48622    23.35956
─────────────┼──────────────────────────────────────────────────────────────
Effect       │
       fines │  -7.692388    .491884   -15.64   0.000    -8.693068   -6.757721
─────────────┴──────────────────────────────────────────────────────────────
Note: Effect estimates are averages of derivatives.
```

The coefficient table now reports the average of the predicted means and the average of the predicted derivatives of the mean function with bootstrap standard errors. The average of the observation-level predicted (citations) is 22.34. The average of the observation-level marginal effects is −7.69, which indicates that increasing fines reduces the mean number of citations.

We use npgraph to graph the estimated conditional mean function.

```
. npgraph
```



The graph shows the negative association between fines and the number of drunk driving citations.

◁

npregress kernel generates system variables for the mean function and the derivative of the mean function. To see the variables that npregress kernel generated for example 1, we type

```
. describe *_*, fullnames

Variable      Storage   Display    Value
    name         type    format    label      Variable label
─────────────────────────────────────────────────────────────────────────────────

_Mean_citations double  %10.0g                Mean function
_d_Mean_citations_dfines
                double  %10.0g                derivative of mean function w.r.t
                                                fines
```

To specify a name for each system variable, we can use the predict() option.

```
. npregress kernel citations fines, predict(mean deriv)
  (output omitted)
. describe mean deriv

Variable      Storage   Display    Value
    name         type    format    label      Variable label
─────────────────────────────────────────────────────────────────────────────────

mean            double  %10.0g                Mean function
deriv           double  %10.0g                derivative of mean function w.r.t
                                                fines
```

Alternatively, we can use the same *stub* for all the variable names by typing predict(hatvar*), which would generate variables hatvar1 and hatvar2.

You may add noderivatives to the option, as in predict(hatvar*, noderivatives), to specify that no derivatives be generated. You save memory when you use noderivatives, but you add to the computational burden. As you will see below, an important feature of npregress kernel is the availability of the margins command after estimation. margins must compute the derivatives and their optimal bandwidth.

▷ Example 3: Selecting the number of bootstrap replications

We start by fitting the model using 200 bootstrap replications. We want to find the number of replications for which the confidence intervals do not change much.

```
. npregress kernel citations fines, reps(200) seed(12)
(running npregress on estimation sample)
Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170.......
> .180.........190.........200 done
Bandwidth
─────────────┬──────────────────────
             │      Mean     Effect
─────────────┼──────────────────────
       fines │  .5631079    .924924
─────────────┴──────────────────────

Local-linear regression                    Number of obs      =          500
Kernel    : epanechnikov                   E(Kernel obs)      =          282
Bandwidth: cross-validation                R-squared          =       0.4380
─────────────┬─────────────────────────────────────────────────────────────
             │   Observed   Bootstrap                         Percentile
   citations │   estimate   std. err.      z    P>|z|     [95% conf. interval]
─────────────┼─────────────────────────────────────────────────────────────
Mean         │
   citations │   22.33999   .4769389    46.84   0.000     21.49744    23.42156
─────────────┼─────────────────────────────────────────────────────────────
Effect       │
       fines │  -7.692388   .5088819   -15.12   0.000    -8.742081    -6.77816
─────────────┴─────────────────────────────────────────────────────────────
Note: Effect estimates are averages of derivatives.
```

For 200 replications, the confidence interval for the mean ranges from 21.50 to 23.42. For the effect of fines, this range is −8.74 to −6.78.

We repeat the estimation using 300 replications and the same seed as in the previous case.

```
. npregress kernel citations fines, reps(300) seed(12)
(running npregress on estimation sample)
Bootstrap replications (300): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170.......
> .180.........190.........200.........210.........220.........230.........240.
> .......250.........260.........270.........280.........290.........300 done
Bandwidth
─────────────┬──────────────────────
             │      Mean     Effect
─────────────┼──────────────────────
       fines │  .5631079    .924924
─────────────┴──────────────────────
```

```
Local-linear regression                    Number of obs      =          500
Kernel   : epanechnikov                    E(Kernel obs)      =          282
Bandwidth: cross-validation                R-squared          =       0.4380

------------------------------------------------------------------------------
             |   Observed   Bootstrap                          Percentile
   citations |   estimate   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
Mean         |
   citations |   22.33999   .4570611    48.88   0.000     21.49359    23.36299
-------------+----------------------------------------------------------------
Effect       |
       fines |  -7.692388   .4981956   -15.44   0.000    -8.673813   -6.720508
------------------------------------------------------------------------------
Note: Effect estimates are averages of derivatives.
```

The confidence interval for the mean ranges from 21.49 to 23.36. For the effect of fines, this range is −8.67 to −6.72. There are some differences so we try estimation with 400 replications.

```
. npregress kernel citations fines, reps(400) seed(12)
(running npregress on estimation sample)
Bootstrap replications (400): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170.......
> .180.........190.........200.........210.........220.........230.........240.
> ........250.........260.........270.........280.........290.........300......
> ...310.........320.........330.........340.........350.........360.........37
> 0.........380.........390.........400 done
Bandwidth

-----------------------------------
             |      Mean     Effect
-------------+---------------------
       fines |  .5631079    .924924
-----------------------------------

Local-linear regression                    Number of obs      =          500
Kernel   : epanechnikov                    E(Kernel obs)      =          282
Bandwidth: cross-validation                R-squared          =       0.4380

------------------------------------------------------------------------------
             |   Observed   Bootstrap                          Percentile
   citations |   estimate   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
Mean         |
   citations |   22.33999   .4588298    48.69   0.000     21.48622    23.35956
-------------+----------------------------------------------------------------
Effect       |
       fines |  -7.692388    .491884   -15.64   0.000    -8.693068   -6.757721
------------------------------------------------------------------------------
Note: Effect estimates are averages of derivatives.
```

The confidence interval for the mean ranges from 21.49 to 23.36. In the case of the effect of fines, these ranges are −8.69 to −6.76. The changes are small so we decide to use 400 replications.

◁

▷ Example 4: Estimating the effect of a percentage change in a covariate

Nonparametric estimation and the bootstrap are computationally intensive, so we use only 200 replications here.

We now extend example 2. In addition to `fines`, we model `citations` as a function of whether the jurisdiction taxes alcoholic beverages (`taxes`); whether the city is small, medium, or large (`csize`); and whether there is a college in the jurisdiction (`college`).

```
. npregress kernel citations fines i.taxes i.csize i.college, nolog
> reps(200) seed(12)
(running npregress on estimation sample)
Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done
Bandwidth

─────────────────────────┬────────────────────
                         │      Mean     Effect
─────────────────────────┼────────────────────
                   fines │  .4471373   .6537197
                   taxes │  .4375656   .4375656
                   csize │  .3938759   .3938759
                 college │   .554583    .554583
─────────────────────────┴────────────────────

Local-linear regression                    Number of obs      =          500
Continuous kernel : epanechnikov           E(Kernel obs)      =          224
Discrete kernel   : liracine               R-squared          =       0.8010
Bandwidth         : cross-validation
─────────────────────────┬──────────────────────────────────────────────────────────────────
                         │   Observed   Bootstrap                         Percentile
               citations │   estimate   std. err.      z    P>|z|     [95% conf. interval]
─────────────────────────┼──────────────────────────────────────────────────────────────────
Mean                     │
               citations │   22.26306   .4642464   47.96   0.000     21.46204     23.2516
─────────────────────────┼──────────────────────────────────────────────────────────────────
Effect                   │
                   fines │  -7.332833   .3316656  -22.11   0.000    -8.013487   -6.741899
                         │
                   taxes │
                    (Tax │
                      vs │
                 No tax) │  -4.502718      .5012   -8.98   0.000    -5.437733   -3.544934
                         │
                   csize │
                 (Medium │
                      vs │
                  Small) │   5.300524   .2687413   19.72   0.000     4.758121    5.797119
                  (Large │
                      vs │
                  Small) │   11.05053    .502633   21.99   0.000     10.00169    11.94311
                         │
                 college │
                (College │
                      vs │
             Not coll..) │   5.953188    .461057   12.91   0.000     5.086511     6.88612
─────────────────────────┴──────────────────────────────────────────────────────────────────
Note: Effect estimates are averages of derivatives for continuous covariates
      and averages of contrasts for factor covariates.
```

The mean number of `citations` predicted by the mean estimates is 22.26. The average marginal effect of `fines` is −7.33, slightly less in magnitude than the −7.69 that we estimated in example 2.

The average marginal effect tells us the result of an infinitesimal change in fines on `citations`. Instead of talking about infinitesimal changes, we want to know the effect of increasing `fines` by 15%. We can use `margins` to estimate the mean number of citations that would occur if fines were increased by 15%.

```
. margins, at(fines=generate(fines*1.15)) reps(200) seed(12)
(running margins on estimation sample)

Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done

Predictive margins                                        Number of obs = 500
                                                          Replications  = 200

Expression: Mean function, predict()
At: fines = fines*1.15

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Percentile
             |     margin   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       _cons |   14.00818    .866694    16.16   0.000     11.39967    15.00145
------------------------------------------------------------------------------
```

The estimated mean number of `citations` with the new level of `fines` is 14.01, which is smaller than the mean 22.26 that was estimated with the observed `fines`. We can formally compare this estimate with the mean at the original level of `fines`. We use the `contrast()` option with `margins` to estimate the difference in these means.

```
. margins, at(fines=generate(fines)) at(fines=generate(fines*1.15))
> contrast(atcontrast(r) nowald) reps(200) seed(12)
(running margins on estimation sample)

Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done
Contrasts of predictive margins
                                                 Number of obs = 500
                                                 Replications  = 200

Expression: Mean function, predict()
1._at: fines =      fines
2._at: fines = fines*1.15

--------------------------------------------------------------
             |   Observed   Bootstrap          Percentile
             |   contrast   std. err.     [95% conf. interval]
-------------+------------------------------------------------
         _at |
   (2 vs 1)  |  -8.254875    .8058215     -10.44121   -7.381583
--------------------------------------------------------------
```

We find that increasing fines by 15% reduces the average number of drunk driving citations by 8.25.

◁

▷ Example 5: Estimating the effect of a change in level

Now, we estimate the effect of increasing `fines` from $10,000 to $11,000 for fixed levels of the other covariates. The other covariate values identify a jurisdiction with a set of characteristics of interest: of medium size, with a college, and taxes alcohol.

First, we use `margins` to estimate the means for a jurisdiction with the characteristics of interest for the two levels of fines.

```
. margins, at(fines=10 taxes=1 csize=2 college=1)
> at(fines=11 taxes=1 csize=2 college=1) reps(200) seed(12)
(running margins on estimation sample)

Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done

Adjusted predictions                                    Number of obs = 500
                                                        Replications  = 200

Expression: Mean function, predict()
1._at: fines   = 10
       taxes   =  1
       csize   =  2
       college =  1
2._at: fines   = 11
       taxes   =  1
       csize   =  2
       college =  1

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Percentile
             |     margin   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         _at |
          1  |   23.17242   .5746008    40.33   0.000     21.95222    24.30412
          2  |   15.90157    .972558    16.35   0.000     13.87449     17.7134
------------------------------------------------------------------------------
```

For a medium-sized jurisdiction that taxes alcohol and has a college, the estimated mean of citations when fines are $10,000 is 23.17, and the estimated mean of citations when fines are $11,000 is 15.90.

We now use `margins` to estimate the difference in these means.

```
. margins, at(fines=10 taxes=1 csize=2 college=1)
> at(fines=11 taxes=1 csize=2 college=1)
> contrast(atcontrast(r) nowald) reps(200) seed(12)
(running margins on estimation sample)
Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done
Contrasts of predictive margins
                                                 Number of obs = 500
                                                 Replications  = 200

Expression: Mean function, predict()
1._at: fines   = 10
       taxes   =  1
       csize   =  2
       college =  1
2._at: fines   = 11
       taxes   =  1
       csize   =  2
       college =  1

──────────────────────────────────────────────────────────────
             │  Observed   Bootstrap          Percentile
             │  contrast   std. err.     [95% conf. interval]
─────────────┼────────────────────────────────────────────────
         _at │
   (2 vs 1)  │ -7.270858    1.003861     -9.096777   -5.162513
──────────────────────────────────────────────────────────────
```

In these jurisdictions, increasing fines from $10,000 to $11,000 reduces the average number of citations by 7.27.

◁

▷ Example 6: Population-averaged covariate effects

In example 5, we estimated the means for two values of fines for a medium-sized jurisdiction with a college and taxes on alcohol. We specified values for each covariate in our model. In this example, we will now estimate population-averaged means instead of means at specific levels of all covariates.

We first estimate the means for two levels of fines. We do not specify values for `csize`, `college`, or `taxes`, so the estimated means are unconditional on these covariates. We use `margins` to estimate means of citations when fines are $10,000 and when fines are $11,000:

```
. margins, at(fines=10) at(fines=11) reps(200) seed(12)
(running margins on estimation sample)

Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done

Predictive margins                                      Number of obs = 500
                                                        Replications  = 200

Expression: Mean function, predict()
1._at: fines = 10
2._at: fines = 11

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Percentile
             |     margin   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         _at |
          1  |   20.50161   .3281821    62.47   0.000     19.90257    21.08954
          2  |   14.97432   .3815647    39.24   0.000     14.14858    15.59955
------------------------------------------------------------------------------
```

The estimated mean of citations when fines are $10,000 is 20.50, and the estimated mean of citations when fines are $11,000 is 14.97. We now use `margins` to estimate the difference in these means:

```
. margins, at(fines=10) at(fines=11)
> contrast(atcontrast(r) nowald) reps(200) seed(12)
(running margins on estimation sample)

Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done
Contrasts of predictive margins
                                                Number of obs = 500
                                                Replications  = 200

Expression: Mean function, predict()
1._at: fines = 10
2._at: fines = 11

--------------------------------------------------------------
             |   Observed   Bootstrap         Percentile
             |   contrast   std. err.     [95% conf. interval]
-------------+------------------------------------------------
         _at |
    (2 vs 1) |  -5.527288   .3529352     -6.277903   -4.925523
--------------------------------------------------------------
```

When fines increase from $10,000 to $11,000, the mean number of citations is estimated to decrease by 5.53.

Next, we consider the effect of taxing alcoholic beverages. We first estimate the population-averaged number of citations with and without such taxes.

```
. margins taxes, reps(200) seed(12)
(running margins on estimation sample)

Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done

Predictive margins                                       Number of obs = 500
                                                         Replications  = 200

Expression: Mean function, predict()

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Percentile
             |     margin   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       taxes |
     No tax  |   25.47052    .6445729    39.52   0.000     24.17515     26.6114
        Tax  |   20.96781    .4448277    47.14   0.000     20.17071    21.88565
------------------------------------------------------------------------------
```

The estimated mean number of citations is 25.47 when there are no alcohol taxes and 20.97 when there are alcohol taxes. We again use `margins` to estimate the difference in these means.

```
. margins r.taxes, reps(200) seed(12)
(running margins on estimation sample)

Bootstrap replications (200): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100.........110...
> ......120.........130.........140.........150.........160.........170........
> .180.........190.........200 done
Contrasts of predictive margins                  Number of obs = 500
                                                 Replications  = 200

Expression: Mean function, predict()

------------------------------------------------
             |         df        chi2     P>chi2
-------------+----------------------------------
       taxes |          1       80.71     0.0000
------------------------------------------------

-----------------------------------------------------------------
                 |   Observed   Bootstrap         Percentile
                 |   contrast   std. err.     [95% conf. interval]
-----------------+-----------------------------------------------
           taxes |
(Tax vs No tax)  |  -4.502719   .5011999     -5.437733   -3.544934
-----------------------------------------------------------------
```

The mean number of citations is estimated to decrease by 4.50 when alcohol sales are taxed.

◁

Visualizing covariate effects

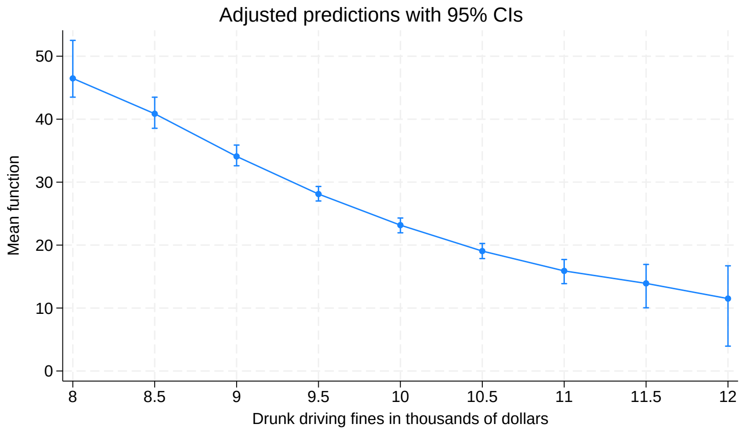
▷ Example 7: Using margins to visualize the mean function and covariate effects

We can also estimate the mean function for the jurisdiction with characteristics of interest over a range of observed fines. We simply add a range of fines to our `margins` specification from example 4.

```
. margins, at(fines=(8(0.5)12) taxes=1 csize=2 college=1) reps(200) seed(12)
  (output omitted)
```

We graph these results using `marginsplot`.

```
. marginsplot
  (output omitted)
```



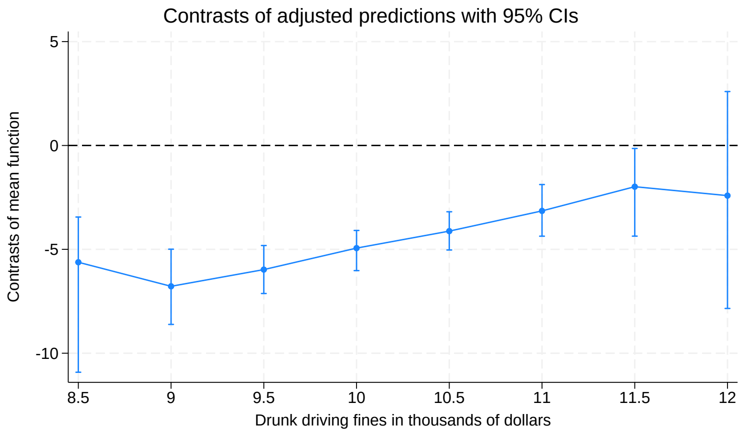
We estimated the mean when fines are $8,000, $8,500, and so on. From these estimated means, we can estimate the effect of a $500 increase for each of these levels of fines.

We simply reissue our `margins` command and specify a reverse adjacent contrast that subtracts the current level from the next level for each level of fines.

```
. margins, at(fines=(8(0.5)12) taxes=1 csize=2 college=1)
> contrast(atcontrast(ar)) reps(200) seed(12)
  (output omitted)
```

We again graph the results, adding a reference line at 0 that designates no change in citations:

```
. marginsplot, yline(0)
  (output omitted)
```



For each level of fines between \$8,500 and \$11,500, the effect of a \$500 increase reduces the mean number of drunk driving incidents. Between \$11,500 and \$12,000, the difference of a \$500 increase is not statistically different from 0.

It would be easy to construct a similar graph for the population-averaged effects in example 6. Simply omit the terms that set the other covariates at fixed values.

◁

Stored results

`npregress kernel` stores the following in `e()`:

Scalars	
`e(N)`	number of observations
`e(mean)`	mean of mean function
`e(r2)`	R^2
`e(nh)`	expected kernel observations
`e(converged_effect)`	1 if effect optimization converged, `0` otherwise
`e(converged_mean)`	1 if mean optimization converged, `0` otherwise
`e(converged)`	1 if effect and mean optimization converged, `0` otherwise
Macros	
`e(cmd)`	`npregress`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(estimator)`	`linear` or `constant`
`e(kname)`	name of continuous kernel
`e(dkname)`	name of discrete kernel
`e(bselector)`	criterion function for bandwidth selection
`e(title)`	title in estimation output
`e(vce)`	*vcetype* specified in `vce()`
`e(properties)`	`b` (or `b V` if `reps()` specified)
`e(datasignaturevars)`	variables used in calculation of checksum
`e(datasignature)`	the checksum
`e(estat_cmd)`	program used to implement `estat`
`e(predict)`	program used to implement `predict`
`e(marginsok)`	predictions allowed by `margins`
`e(marginsprop)`	signals to the `margins` command
Matrices	
`e(b)`	coefficient vector
`e(bwidth)`	bandwidth for all predictions
`e(derivbwidth)`	bandwidth for the derivative
`e(meanbwidth)`	bandwidth for the mean
`e(ilog_mean)`	iteration log for mean (up to 20 iterations)
`e(ilog_effect)`	iteration log for effects (up to 20 iterations)
Functions	
`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

The regression model of outcome y_i given the k-dimensional vector of covariates $\mathbf{x}_i$ was defined in (1) and (2) of *Remarks and examples* and repeated here:

$$y_i = g(\mathbf{x}_i) + \varepsilon_i \tag{4}$$

$$E(\varepsilon_i|\mathbf{x}_i) = 0 \tag{5}$$

where ε_i is the error term. The covariates may include discrete and continuous variables. Equations (4) and (5) imply that

$$E(y_i|\mathbf{x}_i) = g(\mathbf{x}_i)$$

`npregress kernel`, by default, estimates a local-linear regression; see Fan and Gijbels (1996) for a good reference on local-linear regression. As we discussed in *Remarks and examples*, local-linear regression estimates a regression for a subset of observations for each point in our data and solves the minimization problem given by

$$\min_{\boldsymbol{\gamma}} \sum_{i=1}^{n} \left\{ y_i - \gamma_0 - \boldsymbol{\gamma}_1' (\mathbf{x}_i - \mathbf{x}) \right\}^2 K(\mathbf{x}_i, \mathbf{x}, \mathbf{h}) \tag{6}$$

where $\boldsymbol{\gamma} = (\gamma_0, \boldsymbol{\gamma}_1')'$ and $K(\mathbf{x}_i, \mathbf{x}, \mathbf{h})$ is the product of the kernels for each covariate.

$$K(\mathbf{x}_i, \mathbf{x}, \mathbf{h}) = \prod_{j=1}^{k} K_j(x_{ij}, x_j, h_j)$$

The kernel for a continuous covariate is of the form

$$K_j(x_{ij}, x_j, h_j) = k_j\left(\frac{x_{ij} - x_j}{h_j}\right)$$

where $k_j(\cdot)$ is one of the kernels listed in [R] **kdensity**. For discrete covariates, `npregress kernel` uses the Li–Racine kernel given by

$$K_j(x_{ij}, x_j, h_j) = \begin{cases} 1 & \text{if } x_{ij} = x_j \\ h_j & \text{otherwise} \end{cases}$$

By estimating $E(y_i|\mathbf{x}_i = \mathbf{x})$ for all points $\mathbf{x}$ in our data, we obtain an estimate of $E(y_i|\mathbf{x}_i)$. For a given $\mathbf{x}$, the solution to the minimization problem in (6) is given by

$$\widehat{\boldsymbol{\gamma}} = (\mathbf{Z}'\mathbf{W}\mathbf{Z})^{-1} \mathbf{Z}'\mathbf{W}\mathbf{y}$$

where $\widehat{\boldsymbol{\gamma}} = (\widehat{\gamma}_0, \widehat{\boldsymbol{\gamma}}_1')'$, $\mathbf{Z}$ is an $n \times (k+1)$ matrix with an ith row given by $\{1, (\mathbf{x}_i - \mathbf{x})'\}'$, $\mathbf{W}$ is an $n \times n$ diagonal matrix with an ith diagonal given by $K(\mathbf{x}_i, \mathbf{x}, \mathbf{h})$, and $\mathbf{y}$ is the $n \times 1$ outcome vector. $\widehat{\gamma}_0$ is an estimate of $g(\mathbf{x})$, whereas $\widehat{\boldsymbol{\gamma}}_1$ is an estimate of the derivative of $g(\mathbf{x})$ with respect to $\mathbf{x}$. When the matrix $(\mathbf{Z}'\mathbf{W}\mathbf{Z})$ is not full rank, the parameter $\boldsymbol{\gamma}$ is not identified. The observations for which this is true are dropped from the estimation sample.

The local-constant estimator of $g(\mathbf{x})$ is a special case of (6) with $\gamma_1 = \mathbf{0}$. In this case, the solution to the optimization problem is given by

$$\frac{\sum_{i=1}^{n} y_i K(\mathbf{x}_i, \mathbf{x}, \mathbf{h})}{\sum_{i=1}^{n} K(\mathbf{x}_i, \mathbf{x}, \mathbf{h})}$$

This is also known as the Nadaraya–Watson kernel estimator, for Nadaraya (1965) and Watson (1964).

`npregress kernel` and `margins`, when used after `npregress kernel`, use a bootstrap estimate of the standard errors for all the estimated effects and report percentile confidence intervals by default. Cattaneo and Jansson (2018) formally justify this use of the bootstrap and provide a definitive reference for semiparametric estimation and inference using kernel-based estimators. Their work demonstrates that the percentile bootstrap provides better coverage than a normal-based confidence interval for statistics based on kernel estimates. See *Methods and formulas* in [R] **bootstrap** for confidence interval formulas.

The rate of convergence of nonparametric regression estimates is given by the product of the sample size and the bandwidths $\sqrt{n|\mathbf{h}|}$, where $|\mathbf{h}|$ is the product of the bandwidths for each covariate. As the sample size increases, the bandwidth decreases. Thus, the rate of convergence of the estimator is slower than the parametric rate $\sqrt{n}$. Another way of thinking about $n|\mathbf{h}|$ is that, because we are not using all our observations to estimate the mean at each point, we require more data to get more reliable estimates; the convergence rate is thus slower. The rate of convergence also decreases as the number of covariates increases, because $|\mathbf{h}|$ decreases. This is referred to as the curse of dimensionality; see Li and Racine (2007, chap. 2) and Stinchcombe and Drukker (2013) for details.

The convergence rate for the derivative of the mean function is different from the convergence rate of the mean function. Therefore, the bandwidth and bandwidth computation for the derivative are different. `npregress kernel` computes the bandwidth for the derivative function by using cross-validation, as suggested by Henderson et al. (2015).

Acknowledgments

We thank Matias Cattaneo of Princeton University and Qi Li of Texas A&M University for many helpful conversations. We thank Xinrong Li of Central University of Finance and Economics in China for research assistance during her summer internship at StataCorp during the summer of 2008.

References

Cattaneo, M. D., and M. Jansson. 2018. Kernel-based semiparametric estimators: Small bandwidth asymptotics and bootstrap consistency. *Econometrica* 86: 955–995. https://doi.org/10.3982/ECTA12701.

Chetverikov, D., D. Kim, and D. Wilhelm. 2018. Nonparametric instrumental-variable estimation. *Stata Journal* 18: 937–950.

Cox, N. J. 2005. Speaking Stata: Smoothing in various directions. *Stata Journal* 5: 574–593.

———. 2007. Kernel estimation as a basic tool for geomorphological data analysis. *Earth Surface Processes and Landforms* 32: 1902–1912. https://doi.org/10.1002/esp.1518.

Eubank, R. L. 1999. *Nonparametric Regression and Spline Smoothing*. 2nd ed. New York: Dekker. https://doi.org/10.1201/9781482273144.

Fan, J., and I. Gijbels. 1996. *Local Polynomial Modelling and Its Applications*. London: Chapman and Hall. https://doi.org/10.1201/9780203748725.

Gutierrez, R. G., J. M. Linhart, and J. S. Pitblado. 2003. From the help desk: Local polynomial regression and Stata plugins. *Stata Journal* 3: 412–419.

Hansen, B. E. 2014. “Nonparametric sieve regression: Least squares, averaging least squares, and cross-validation”. In *The Oxford Handbook of Applied Nonparametric and Semiparametric Econometrics and Statistics*, edited by J. S. Racine, L. Su, and A. Ullah, 215–248. New York: Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199857944.013.008.

Henderson, D. J., Q. Li, C. F. Parmeter, and S. Yao. 2015. Gradient-based smoothing parameter selection for nonparametric regression estimation. *Journal of Econometrics* 184: 233–241. https://doi.org/10.1016/j.jeconom.2014.09.007.

Hurvich, C. M., 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 ser., 60: 271–293. https://doi.org/10.1111/1467-9868.00125.

Li, Q., X. Lu, and A. Ullah. 2003. Multivariate local polynomial regression for estimating average derivatives. *Journal of Nonparametric Statistics* 15: 607–624. https://doi.org/10.1080/10485250310001605450.

Li, Q., and J. S. Racine. 2004. Cross-validated local linear nonparametric regression. *Statistica Sinica* 14: 485–512.

———. 2007. *Nonparametric Econometrics: Theory and Practice.* Princeton, NJ: Princeton University Press.

Nadaraya, E. A. 1965. On nonparametric estimates of density functions and regression curves. *Theory of Probability and Its Applications* 10: 186–190. https://doi.org/10.1137/1110024.

Pinzon, E. 2017. Nonparametric regression: Like parametric regression, but not. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2017/06/27/nonparametric-regression-like-parametric-regression-but-not/.

Powell, J. L., J. H. Stock, and T. M. Stoker. 1989. Semiparametric estimation of index coefficients. *Econometrica* 57: 1403–1430. https://doi.org/10.2307/1913713.

Powell, J. L., and T. M. Stoker. 1996. Optimal bandwidth choice for density-weighted averages. *Journal of Econometrics* 75: 291–316. https://doi.org/10.1016/0304-4076(95)01761-5.

Rios-Avila, F. 2020. Smooth varying-coefficient models in Stata. *Stata Journal* 20: 647–679.

Shao, J. 2003. *Mathematical Statistics.* 2nd ed. New York: Springer. https://doi.org/10.1007/b97553.

Sheather, S. J., and M. C. Jones. 1991. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society,* B ser., 53: 683–690. https://doi.org/10.1111/j.2517-6161.1991.tb01857.x.

Stinchcombe, M. B., and D. M. Drukker. 2013. “Regression efficacy and the curse of dimensionality”. In *Recent Advances and Future Directions in Causality, Prediction, and Specification Analysis: Essays in Honor of Halbert L. White Jr*, edited by X. Chen and N. R. Swanson, 527–549. New York: Springer. https://doi.org/10.1007/978-1-4614-1653-1_20.

Verardi, V., and N. Debarsy. 2012. Robinson’s square root of N consistent semiparametric regression estimator in Stata. *Stata Journal* 12: 726–735.

Watson, G. S. 1964. Smooth regression analysis. *Sankhyā,* A ser., 26: 359–372.

Also see

[R] **npregress kernel postestimation** — Postestimation tools for npregress kernel

[R] **npregress intro** — Introduction to nonparametric regression

[R] **kdensity** — Univariate kernel density estimation

[R] **lpoly** — Kernel-weighted local polynomial smoothing

[U] 20 Estimation and postestimation commands