The Stata Journal (2003)
3, Number 4, pp. 412–419

From the help desk: Local polynomial regression and Stata plugins

Roberto G. Gutierrez
Jean Marie Linhart
Jeffrey S. Pitblado

StataCorp

Abstract. Local polynomial regression is a generalization of local mean smoothing as described by Nadaraya (1964) and Watson (1964). Instead of fitting a local mean, one instead fits a local pth-order polynomial. Calculations for local polynomial regression are naturally more complex than those for local means, but local polynomial smooths have better statistical properties. The computational complexity may, however, be alleviated by using a Stata plugin. In this article, we describe the `locpoly` command for performing local polynomial regression. The calculations involved are implemented in both ado-code and with a plugin, allowing the user to assess the speed improvement obtained from using the plugin. Source code for the plugin is also provided as part of the package for this program.

Keywords: st0053, local polynomial, local linear, smoothing, kernel, plugin

1 Introduction

The last twenty years or so have seen a significant outgrowth in the literature on the subject of scatterplot smoothing, otherwise known as univariate nonparametric regression. Of most appeal is the idea of not making any assumptions about the functional form for the expected value of a response given a regressor but instead allowing the data to "speak for itself". Various methods and estimators fall into the category of nonparametric regression, including local mean smoothing, as described independently by Nadaraya (1964) and Watson (1964); the Gasser–Müller (1979) estimator; locally weighted scatterplot smoothing (LOWESS), as described by Cleveland (1979); wavelets (e.g., Donoho 1995); and splines (Eubank 1988), to name a few. Much of the vast literature focuses on automating the amount of smoothing to be performed and dealing with the bias/variance trade-off inherent to this type of estimation. For example, in the case of Nadaraya–Watson, the amount of smoothing is controlled by choosing a *bandwidth*.

Smoothing via local polynomials is by no means a new idea but instead one that has been rediscovered in recent years in articles such as Fan (1992). A natural extension of the local mean smoothing of Nadaraya–Watson, local polynomial regression, involves fitting the response to a polynomial form of the regressor via locally weighted least squares. Compared with the Nadaraya–Watson estimator (local polynomial of degree zero), local polynomials of higher order have better bias properties and, in general, do not require bias adjustment at the boundary of the regression space. For a definitive reference on local polynomial smoothing, see Fan and Gijbels (1996).

The apparent cost of these improved properties is that local polynomial smooths are computationally more complex. For example, the Nadaraya–Watson estimator requires at each point in the smoothing grid the calculation of a locally weighted mean, whereas local polynomial smoothing would require a weighted regression at each grid point. This cost, however, can be alleviated by using approximation methods such as linear binning (Hall and Wand 1996) or by using updating methods that retain information from previous points in the smoothing grid (e.g., Fan and Marron 1994). For purposes of simplicity of code, no such devices are considered in this paper. Instead, we will work with the idea of running a full regression at each grid point and gain speed by performing these regressions using a Stata plugin.

Plugins are a relatively new feature of Stata, made available in July 2003. A Stata plugin is a piece of compiled code (written in C or C++) that a user attaches to the Stata executable and then executes either interactively or from within a program. Because they consist of precompiled code, plugins generally run faster than equivalent code written in the ado language, where each command must be interpreted each time it is executed.

In the context of local polynomial regression, we implement a Stata plugin to perform all the required linear regressions, thus speeding up execution considerably. In this paper, we do not discuss the actual creation of the plugin, although we do make our source code available for your examination; for more information on creating Stata plugins, see StataCorp (2003). Calculations are implemented in both ado-code and with a plugin, allowing for comparison in execution times between the two. A dialog-box generating program for `locpoly` is also provided, along with instructions on how you can permanently add an entry for `locpoly` to the Stata **User** menu.

Section 2 describes the method of local polynomial regression. Section 3 provides documentation for the `locpoly` command. Section 4 briefly discusses some issues of using Stata plugins and compares execution times for `locpoly` when the calculations are performed entirely in ado-code with those for which the plugin is used.

2 Local polynomial regression

Consider a set of scatterplot data $\{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ from the model

$$Y_i = m(X_i) + \epsilon_i \tag{1}$$

for some unknown mean function $m()$ and symmetric errors ϵ_i. Without making any assumption about the functional form of $m()$, we wish to estimate $m(x_0) = E(Y|X = x_0)$.

For some x in the neighborhood of x_0, a Taylor expansion of $m(x)$ gives

$$m(x) \approx m(x_0) + m^{(1)}(x_0)(x - x_0) + \frac{m^{(2)}(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{m^{(p)}(x_0)}{p!}(x - x_0)^p$$

That is, we can approximate $m(x)$ locally by a pth order polynomial in $x - x_0$. Substituting this approximation into (1), we see that for X_is local to x_0, $m(x_0)$ can be

estimated as the constant term (intercept) of a regression of Y_i on the polynomial terms $(X_i - x_0), (X_i - x_0)^2, \ldots, (X_i - x_0)^p$.

To preserve the locality, we introduce a kernel function $K()$, which is a probability density function that is symmetric about zero and a bandwidth h to control the degree of locality. Defining $\beta_j = m^{(j)}(x_0)/j!$ for $j = 0, \ldots, p$, we can then estimate $\beta_0 = m(x_0)$ by minimizing in β_j the weighted least squares expression

$$\sum_{i=1}^{n} \left\{ Y_i - \sum_{j=0}^{p} \beta_j (X_i - x_0)^j \right\}^2 K_h(X_i - x_0)$$

for $K_h(a) = h^{-1}K(a/h)$.

The above is equivalent to a weighted linear regression with weights equal to $K_h(X_i - x_0)$. Thus, a local polynomial smooth can be obtained by specifying a smoothing grid consisting of a series of x_0s and then, for each x_0 in the grid, performing the above weighted regression (with polynomial terms $(X_i - x_0)^j$ as regressors) and picking off the estimated intercept term $\widehat{\beta}_0 = \widehat{m}(x_0)$.

It is up to the user to specify the degree p, kernel function $K()$, and bandwidth h. When $p = 0$, the above reduces to local mean smoothing, otherwise known as the Nadaraya–Watson estimator. Note that the above scheme also allows for estimation of the first p derivatives of $m()$ with $\widehat{m}^{(j)}(x_0) = j!\widehat{\beta}_j$, $j = 1, \ldots, p$, although no facility for retrieving these estimates is provided in our implementation.

3 Stata implementation

3.1 Syntax

`locpoly` *yvar xvar* [`if` *exp*] [`in` *range*] [`,` `degree(#)` `width(#)` `n(#)` `at(`*var*$_x$`)` `generate(`[*newvar*$_x$] *newvar*$_y$`)` [`epanechnikov` | `biweight` | `cosine` | `gaussian` | `parzen` | `rectangle` | `triangle`] `adoonly` `nograph` `noscatter` `plot(`*plot*`)` *line_options* *twoway_options*]

3.2 Options

`degree(#)` specifies the degree of the polynomial to be used in the smoothing. Zero is the default, meaning local mean smoothing.

`width(#)` specifies the halfwidth of the kernel, the width of the smoothing window around each point. If `width()` is not specified, the default width is used; see [R] **kdensity**. Note that this default is appropriate for kernel-density estimation and not for local polynomial smoothing. It is best to use the default as a starting point and adjust the bandwidth according to your needs.

`n(#)` specifies the number of points at which the smooth is to be evaluated. The default is $\min(N, 50)$, where N is the number of observations in your data.

`at(`*var*$_x$`)` specifies a variable that contains the values at which the smooth should be evaluated. `at()` allows you to more easily obtain smooths for different variables or different subsamples of a variable and then overlay the estimates for comparison. By default, the smoothing is done on an equally spaced grid, but you can use `at()` to perform the smoothing at the observed xs, for example.

`generate(`$\left[\right.$*newvar*$_x\left.\right]$ *newvar*$_y$`)` creates new variables storing the results of the estimation. *newvar*$_y$ will contain the estimated smooth. *newvar*$_x$ will contain the smoothing grid. If `at()` is not specified, then both *newvar*$_x$ and *newvar*$_y$ must be specified. Otherwise, only *newvar*$_y$ is to be specified.

`epanechnikov`, `biweight`, `cosine`, `guassian`, `parzen`, `rectangle`, and `triangle` specify the kernel, with `epanechnikov` being default. For definitions of these kernels, see [R] **kdensity**.

`adoonly` suppresses the use of the Stata plugin and instead performs the necessary regressions entirely using ado-code. That is, the file containing this program includes a subroutine written in the ado language. This subroutine has also been implemented as a plugin. Both produce the same results, but the plugin is faster and hence run by default. By specifying `adoonly`, you run only in ado-code. Thus, `adoonly` is useful should the plugin not be available on your platform (in our case it will always be), or if you wish to perform speed comparisons.

`nograph` suppresses drawing the graph of the estimated smooth. This option is often used in conjunction with `generate()`.

`noscatter` suppresses superimposing a scatterplot of the observed data over the smooth. This option is useful when the number of resulting points would be so large as to clutter the graph.

`plot(`*plot*`)` provides a way to add other plots to the generated graph. See [G] ***plot_option***.

line_options affect the rendition of the plotted line(s); see [G] **graph twoway line**.

twoway_options are any of the options documented in [G] ***twoway_options***, excluding `by()`. These include options for titling the graph (see [G] ***title_options***) and options for saving the graph to disk (see [G] ***saving_options***).

3.3 Example

Local polynomial regression is described in section 2. For an example, consider the motorcycle data as examined (among other places) in Fan and Gijbels (1996). The data consist of 133 observations and measure the acceleration (`accel` measured in g) of the head of a test object during impact over time (`time` measured in milliseconds). For these data, we use `locpoly` to fit a local cubic polynomial with Gaussian kernel and bandwidth equal to 2.

```
. use motorcycle, clear
(Motorcycle data from Fan & Gijbels (1996))
. describe
Contains data from motorcycle.dta
  obs:           133                          Motorcycle data from Fan &
                                                Gijbels (1996)
 vars:             2                          5 Nov 2003 16:18
 size:         1,596 (99.9% of memory free)
```

variable name	storage type	display format	value label	variable label
time	float	%9.0g		time (msec)
accel	float	%9.0g		acceleration (g)

```
Sorted by:
. locpoly accel time, degree(3) gaussian width(2)
```

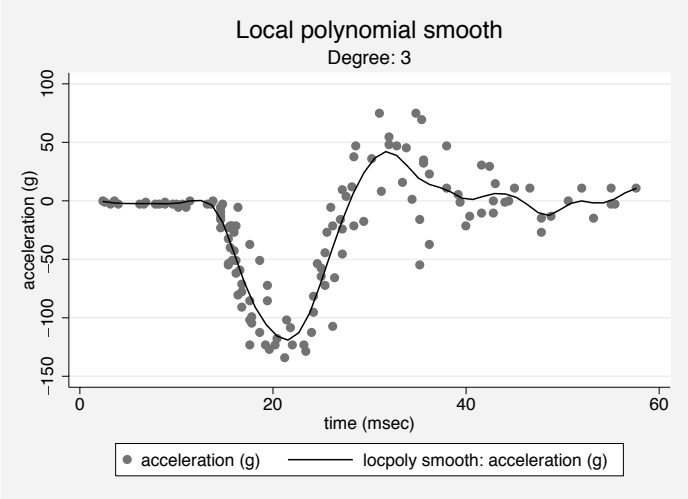

Figure 1: Local cubic smooth of the motorcycle data.

3.4 Saved Results

`locpoly` saves in `r()`:

Scalars

`r(degree)`	smoothing polynomial degree	`r(ngrid)`	number of successful regressions
`r(width)`	bandwidth		

Macros

`r(kernel)`	name of kernel

3.5 Dialog

The `locpoly` package includes a dialog-box program for this command, contained in the file `locpoly.dlg`, which is downloaded with the program. To launch this dialog interactively, type `db locpoly` from within Stata.

Alternately, GUI users can add `locpoly` permanently to their **User** menu by including the following in `profile.do`:

```
if _caller() > 7 {
        if "`c(console)'"=="" {
                window menu append item "stUserGraphics" /*
                    */ "Local Polynomial Smoothing (&locpoly)" "db locpoly"
        }
}
```

The first line ensures that you are running Stata 8 or later, and the second ensures that you are running a GUI version of Stata and not console Stata. The `window` command then adds an entry for local polynomial smoothing within the **Graphics** submenu of the **User** menu, and selecting this entry launches the `locpoly` dialog.

For more information on customizing your **User** menu, see [P] **window menu**.

4 locpoly uses a Stata plugin

Plugins are useful for speeding up numerical calculations and similar manipulations, but Stata ado-code is much better at tasks such as parsing syntax, dealing with options, saving results, and as is the case of `locpoly`, creating graphs. As such, most programs that utilize plugins will be written almost entirely in ado-code, with only the most computationally intense portions relegated to plugins.

In the case of `locpoly`, the difficult computations are the regressions required for each point in the smoothing grid. Since the number of points in the grid can be as large as the number of observations in the data, looping over these points can be slow. Also, from within Stata, the best way to fit a weighted linear regression model for which the weights are nonintegers is to use `regress` with `iweight`s. Since `regress` is a built-in command, it is very fast. However, since we are only interested in the estimated intercept, running a full-blown regression can be bit wasteful. Therefore, there is considerable speed to be gained by writing a plugin that does the looping implicitly and that cleverly performs the matrix manipulations necessary to estimate only the intercept term in a weighted regression.

Within `locpoly.ado` exists the subroutine `Lpwork`, which loops over the grid and performs a weighted regression at each point. `Lpwork` is written entirely in ado-code, but an equivalent plugin routine has also been provided in precompiled form in the file `locpoly.plugin`, which is downloaded as part of this package. Since plugins are platform specific, the version of `locpoly.plugin` that you download depends on your computer platform (Windows, Macintosh, IBM–AIX, etc.), but this is handled automatically within the Stata package file (`.pkg`) for this package; see [R] **net** for the details

on making platform-specific files available for download. Note that since we work at StataCorp, it was easy enough for us to compile the plugin code on all platforms, so `locpoly.plugin` is available to anyone who can run Stata.

For those interested in compiling the plugin themselves following the instructions given in StataCorp (2003), or for those interested in examining the source code, the file `locpoly.c` is also available as part of this package for download via `net get`; see [R] **net**.

When we initially wrote `locpoly`, we wrote it entirely in ado-code; i.e., we implemented the calculations via the `Lpwork` subroutine. When we implemented the equivalent plugin routine, we could have just thrown `Lpwork` away in favor of `locpoly.plugin`, but we realized that keeping both around would allow us to compare the execution times for both implementations and to reestablish at a future time that both implementations produced the same results. In order to easily switch between the two, we added the `adoonly` option to `locpoly`. By default, `locpoly` calls the plugin to perform the calculations. When you specify `adoonly`, however, `locpoly` instead uses `Lpwork`, which, again, is written entirely as ado-code.

Using the motorcycle data, we can use the `adoonly` option to compare execution times, with and without the plugin. We perform the same smooth that we did before, this time setting the size of smoothing grid equal to the number of observations in our data. We also add the `nograph` option, so as to not confound our comparison with the time required to draw the graph.

```
. use motorcycle, clear
(Motorcycle data from Fan & Gijbels (1996))
. set rmsg on
r; t=0.00 17:00:31
. locpoly accel time, degree(3) gaussian width(2) n(133) nograph
r; t=0.01 17:00:31
. locpoly accel time, degree(3) gaussian width(2) n(133) nograph adoonly
r; t=0.12 17:00:31
```

Using the plugin in this case resulted in code that ran about 12 times faster. We ran the above on a 2.4GHz PC running Linux. Timings will vary depending on the platform, size of the dataset, number of smoothing points, degree of the polynomial, etc., but in general, the speed gain obtained from using `locpoly` with the plugin is substantial.

5 References

Cleveland, W. S. 1979. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association* 74: 829–836.

Donoho, D. L. 1995. Nonlinear solution of linear inverse problems by wavelet–vaguelette decomposition. *Applied and Computational Harmonic Analysis* 2: 101–126.

Eubank, R. L. 1988. *Spline Smoothing and Nonparametric Regression*. New York: Marcel Dekker.

Fan, J. 1992. Design–adaptive nonparametric regression. *Journal of the American Statistical Association* 87: 998–1004.

Fan, J. and I. Gijbels. 1996. *Local Polynomial Modelling and Its Applications*. London: Chapman & Hall.

Fan, J. and J. S. Marron. 1994. Fast implementations of nonparametric curve estimation. *Journal of Computational and Graphical Statistics* 3: 35–56.

Gasser, T. and H.-G. Müller. 1979. Kernel estimation of regression functions. In *Smoothing Techniques for Curve Estimation*, Lecture Notes in Mathematics, vol. 757, 23–68. New York: Springer.

Hall, P. and M. P. Wand. 1996. On the accuracy of binned kernel density estimates. *Journal of Multivariate Analysis* 56: 165–184.

Nadaraya, E. A. 1964. On estimating regression. *Theory of Probability and Its Application* 9: 141–142.

StataCorp. 2003. *Creating and using Stata plugins*. *http://www.stata.com/support/plugins*

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

About the Authors

Roberto G. Gutierrez is Director of Statistics at StataCorp.

Jean Marie Linhart is Senior Mathematician at StataCorp.

Jeffrey S. Pitblado is Senior Statistician at StataCorp.