The Stata Journal

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*), *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		**Printed & electronic**	
1-year subscription	$115	1-year subscription	$145
2-year subscription	$210	2-year subscription	$270
3-year subscription	$285	3-year subscription	$375
1-year student subscription	$ 85	1-year student subscription	$115
1-year institutional subscription	$345	1-year institutional subscription	$375
2-year institutional subscription	$625	2-year institutional subscription	$685
3-year institutional subscription	$875	3-year institutional subscription	$965
Electronic only		**Electronic only**	
1-year subscription	$ 85	1-year subscription	$ 85
2-year subscription	$155	2-year subscription	$155
3-year subscription	$215	3-year subscription	$215
1-year student subscription	$ 55	1-year student subscription	$ 55

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata, STATA, Stata Press, Mata, MATA, and NetCourse are registered trademarks of StataCorp LP.

The Stata Journal (2014)
14, Number 4, pp. 708–737

Plotting regression coefficients and other estimates

Ben Jann
University of Bern
Bern, Switzerland
jann@soz.unibe.ch

Abstract. Graphical display of regression results has become increasingly popular in presentations and in scientific literature because graphs are often much easier to read than tables. Such plots can be produced in Stata by the `marginsplot` command (see [R] **marginsplot**). However, while `marginsplot` is versatile and flexible, it has two major limitations: it can only process results left behind by `margins` (see [R] **margins**), and it can handle only one set of results at a time. In this article, I introduce a new command called `coefplot` that overcomes these limitations. It plots results from any estimation command and combines results from several models into one graph. The default behavior of `coefplot` is to plot markers for coefficients and horizontal spikes for confidence intervals. However, `coefplot` can also produce other types of graphs. I illustrate the capabilities of `coefplot` by using a series of examples.

Keywords: gr0059, coefplot, marginsplot, margins, regression plot, coefficients plot, ropeladder plot

1 Introduction

Tabulating regression coefficients has long been the preferred way of communicating results from statistical models. However, researchers now increasingly use graphs to present regression results, for several reasons. On the one hand, interpretation of regression tables can be challenging, especially if there are interaction effects, categorical variables, or nonlinear functional forms. Moreover, in nonlinear models, the original regression coefficients are often not the primary interest of researchers. For example, in logistic regression, the raw coefficients represent effects on log odds. However, most people would be more comfortable with effects expressed on the probability scale. Because probability effects are not constant in such a model, it can be helpful, for example, to plot effect functions. On the other hand, and more fundamentally, researchers have recognized that displaying results in the form of graphs can be much more effective than tabulation, especially in presentations and lectures, but also in written work. This is because the "reexpression of data in pictorial form capitalizes upon one of the most highly developed human information processing capabilities—the ability to recognize, classify, and remember visual patterns" (Lewandowsky and Spence 1989, 200).

Tables are well-suited as a look-up source for specific values, but it is difficult to interpret results presented as numbers in tables. Graphs generally do a much better job of "revealing patterns, trends, and relative quantities" (Jacoby 1997, 7) because graphs translate differences among numbers into spatial distances, thereby emphasizing the main features of the data and abstracting from irrelevant details. Pictorial representations of information also seem to be easier to remember (Lewandowsky and Spence 1989).[1]

Graphics are present in many scientific fields. Most prominently, graphs are used to depict univariate distributions (for example, histograms), bivariate distributions (for example, scatterplots), or changes over time (line diagrams). They are used to analyze data—for example, to get a quick overview of important features of the data or evaluate assumptions imposed by statistical models—or to present results (Healy and Moody 2014).

One type of presentation plot that has become popular recently, sometimes called a ropeladder plot, displays regression coefficients or other statistics of interest against a common scale, using markers for point estimates and spikes for confidence intervals (for examples, see Kastellec and Leoni [2007], Harrell [2001], Cleveland [1994, 217–220], Cleveland and McGill [1985], Dice and Leraas [1936], Gosset [Student, pseud.] [1927], and Chapin [1924]). It can be very effective to present statistical results in this way because evaluating the position of points along a common scale and judging the length of lines are two of the most powerful perceptional capabilities of humans (Cleveland and McGill 1985). Furthermore, ropeladder plots provide an immediate and accurate impression of the statistical precision of results, much preferred over p-values and significance stars in regression tables.

Unfortunately, creating such graphs in Stata is tedious, hindering their more widespread use (although, see Newson [2003]). The coefficients and variances have to be gathered from the `e()` returns, the confidence intervals have to be computed, and the results have to be appropriately stored as variables in the dataset. Then, a suitable variable for the category axis must be generated and coefficient labels must be defined. Finally, a complicated graph command has to be issued to plot the coefficients and confidence intervals.

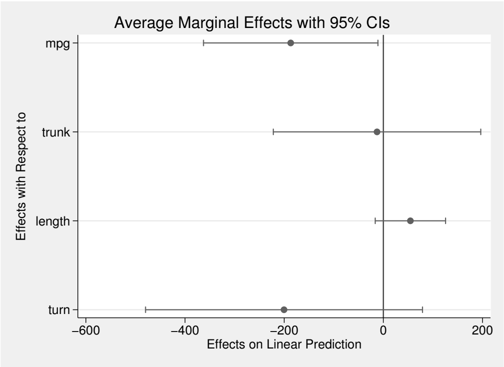
This task has been greatly simplified with the introduction of `marginsplot` (see [R] **marginsplot**) in Stata 12. It is now possible to plot coefficients and confidence intervals with just a few lines of code. For example, consider the following linear regression model (see [R] **regress**):

1. For a brief review of the literature on the merits of graphical displays over tabular representations, see Gelman, Pasarica, and Dodhia (2002). For results on graphical perceptions and general principles on designing effective graphics, see the works by Chambers et al. (1983), Lewandowsky and Spence (1989), and Cleveland (1993, 1994). As a rich source of inspiration, consider Tufte (2001) and Wainer (1997).

```
. sysuse auto
(1978 Automobile Data)

. regress price mpg trunk length turn

      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  4,    69) =    5.79
       Model |   159570047     4  39892511.8           Prob > F      =  0.0004
    Residual |   475495349    69  6891236.94           R-squared     =  0.2513
-------------+------------------------------           Adj R-squared =  0.2079
       Total |   635065396    73  8699525.97           Root MSE      =  2625.1

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         mpg |  -186.8417   88.17601    -2.12   0.038     -362.748   -10.93533
       trunk |  -12.72642   104.8785    -0.12   0.904    -221.9534    196.5005
      length |   54.55294   35.56248     1.53   0.130    -16.39227    125.4981
        turn |  -200.3248   140.0166    -1.43   0.157    -479.6502    79.00066
       _cons |   8009.893   6205.538     1.29   0.201    -4369.817     20389.6
------------------------------------------------------------------------------
```

To plot the regression coefficients (which, in this case, are equal to the average marginal effects), we could type

```
. margins, dydx(*)
  (output omitted)
. marginsplot, horizontal xline(0)
>    yscale(reverse) recast(scatter)
  (output omitted)
```



marginsplot is a versatile command that can do so much, especially when plotting predictive margins, the area of application that **marginsplot** was primarily designed for. However, **marginsplot** can only process results left behind by **margins** (see [R] **margins**), and it has some other limitations.

To overcome these limitations, I wrote a new command called **coefplot**. On the following pages, I illustrate the scope and usage of **coefplot** through a series of examples. For a systematic overview of the syntax and options, type **help coefplot** after you install the command in Stata.

2 Scope of coefplot

coefplot is a tool to graph results from estimation commands in Stata, comparable to commands such as **outreg** (Gallup 2012) or **estout** (Jann 2007) for tables. Some of coefplot's functionality overlaps with the possibilities offered by marginsplot, but coefplot goes much further.

- marginsplot can only process the results left behind by the margins command. coefplot, however, can be applied to the results of any estimation command in Stata that posts its results in e() (as most estimation commands do, including margins if specified with the post option) and can even be used to plot results that have been collected manually using the matrix commands (see section 7.6).
- marginsplot can only process the results from one call to margins. As with tables, however, it is often desirable to combine results from several model specifications or estimation techniques into one graph. With coefplot, multiple results can be freely combined and arranged in one graph, including the possibility to distribute results across subgraphs.
- marginsplot draws confidence intervals for only one confidence level. Given the criticism of a strict interpretation of significance tests and confidence intervals, it seems advisable to display multiple confidence intervals using varying levels. coefplot offers such functionality.
- Finally, good graphs need good labels. coefplot offers various options to label coefficients, equations, and subgraphs, include labels for groups of estimates, or to insert subheadings to structure the display.

The main purpose of coefplot is to plot point estimates of coefficients along with confidence intervals. By default, coefplot draws a ropeladder plot using markers for point estimates and spikes for confidence intervals and by arranging the estimates along a categorical axis providing labels for the different coefficients. Depending on context, however, it can also be sensible to draw different types of graphs. For example, one can use bars for point estimates and capped spikes for confidence intervals or display estimates as connected lines along a continuous axis, which is all supported by coefplot.

Figure 1 provides a "tour d'horizon" of coefplot, illustrating its scope.

Graph A displays a standard ropeladder plot containing regression coefficients from two subgroups for two different dependent variables. In sections 3 and 4, I discuss the basic usage of coefplot and explain how to create such a plot.

Graph B is a variant of a standard ropeladder plot, in which subgraphs are drawn by the coefficient instead of by models (see section 7.4). Graph B also illustrates the use of group labels (note the label "Subgroup results" below "Domestic" and "Foreign"), as discussed in section 5.

Graph C is an extreme example of using multiple confidence intervals. The graph contains 50 confidence intervals for each coefficient from levels 1% to 99% using varying line widths and color intensities. Of course, it is also possible to include just a few confidence intervals, say, the 99% and 95% confidence intervals, as discussed in section 6.

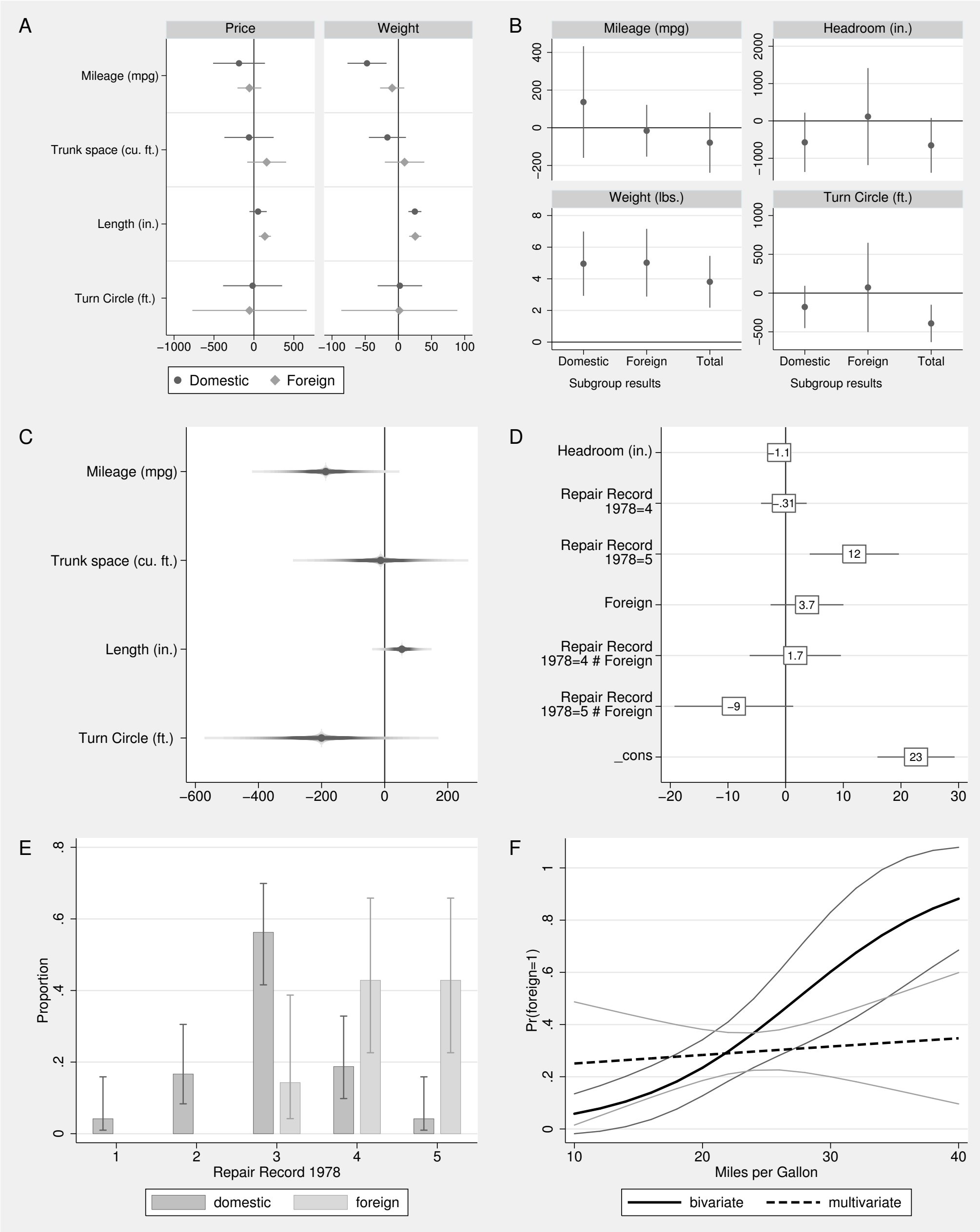


Figure 1. Examples of graphs produced by `coefplot`

Graph D is a plot in which the values of estimates are displayed as marker labels, as discussed in section 7.3. Furthermore, graph D also illustrates the automatic wrapping of long labels (see section 5).

Graph E is a bar plot with capped spikes for confidence intervals, a graph type that is appropriate if the estimates to be plotted are proportions (see section 7.2).

Finally, graph F is yet another graph type, suitable for plotting effect contours in which the categorical axis containing coefficient labels has been replaced with a continuous axis. Such graphs can be created by providing plot positions through the `at()` option, as discussed in section 7.5.

In the remainder of this article, I discuss how to use `coefplot` to produce rope-ladder plots (sections 3 and 4), introduce the various options for labeling the categorical axis (section 5), and illustrate the use of multiple confidence intervals (section 6). In section 7, I will cover more-advanced topics, such as using the `recast()` option, adding marker labels, arranging subgraphs by coefficients, using a continuous axis, and plotting results from matrices.

3 Plotting a single model

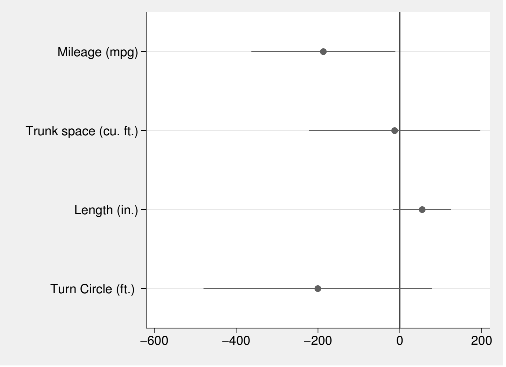
The syntax to produce a plot of the coefficients of one model is

`coefplot` [*name*] [, *options*]

where *name* is the name of a stored model (see [R] **estimates**), or `.` or empty string denoting the active model. For details about `coefplot` syntax, see `help coefplot`.

For example, to plot point estimates and 95% confidence intervals for the most recent model, type

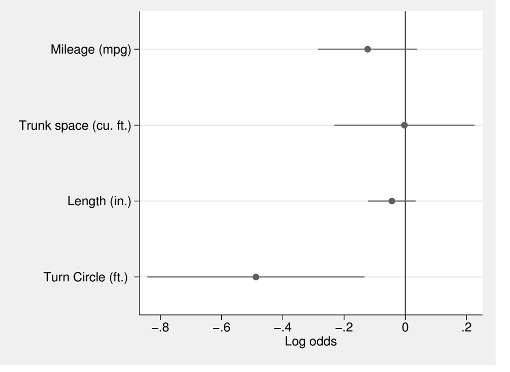
```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) xline(0)
```



Option `drop(_cons)` was added to remove the constant and `xline(0)` was used to draw a reference line at 0 so that we can better see which coefficients are significantly different from 0.

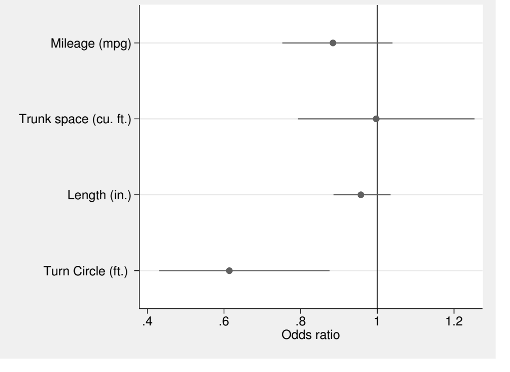
`coefplot` can graph results from almost any estimation command. For example, to plot coefficients from a logit model (see [R] **logit**), type

```
. sysuse auto, clear
(1978 Automobile Data)
. logit foreign mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) xline(0)
>    xtitle(Log odds)
```



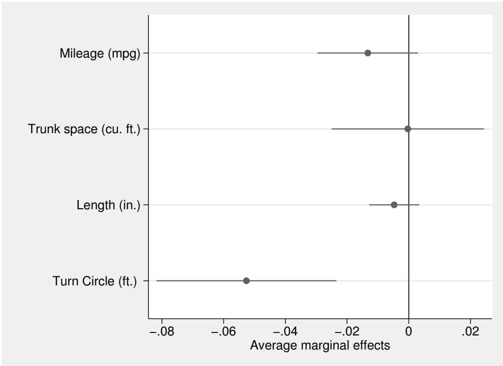
With logit models, one is often interested in odds ratios instead of the raw coefficients. To plot odds ratios instead of log odds, use the `eform` option, which causes `coefplot` to compute exponents of coefficients and confidence intervals (using endpoint transformation).

```
. coefplot, drop(_cons) xline(1) eform
>    xtitle(Odds ratio)
```



Furthermore, if you want to plot average marginal effects instead of log odds or odds ratios, you can apply `margins` (see [R] **margins**).

```
. sysuse auto, clear
(1978 Automobile Data)
. logit foreign mpg trunk length turn
  (output omitted)
. margins, dydx(*) post
  (output omitted)
. coefplot, xline(0)
>    xtitle(Average marginal effects)
```



It is essential to specify the `post` option with `margins` so that it posts its results in `e()`, which is where `coefplot` collects the results to display. If you do not specify the `post` option, then `margins` leaves `e()` unchanged and `coefplot` uses the raw coefficients from the logit model that still reside in `e()`.

4 Plotting multiple models

To include results from several commands in one graph, one can save the results from each command by using `estimates store` (see [R] **estimates**) and then provide the names of the stored estimation sets to `coefplot`. There are three alternatives for including multiple results in the graph. First, one can include models as different plots in the same graph. By "plot", I mean a set of markers and confidence spikes using the same plot style. Second, one can create separate subgraphs, with each subgraph containing one or more plots. Third, one can append multiple models into the same plot.

4.1 Models as plots

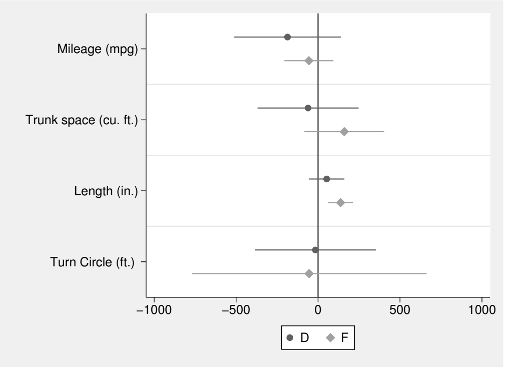
The syntax to include multiple models as separate plots is

`coefplot` [`(`]*name*[`,` *plotopts*`)`] [`(`*name*`,` *plotopts*`)` ...] [`,` *globalopts*]

where *name* is again the name of a stored model, or `.` or empty string denoting the active model. *plotopts* are options that apply to a single plot. They specify the information to be collected, affect the rendition of the plot, and provide a label for the plot in the legend. *globalopts* are options that apply to the overall graph, such as titles or axis labels, but may also contain any options allowed as plot options to provide defaults for the single plots. For details about `coefplot` syntax, see `help coefplot`.

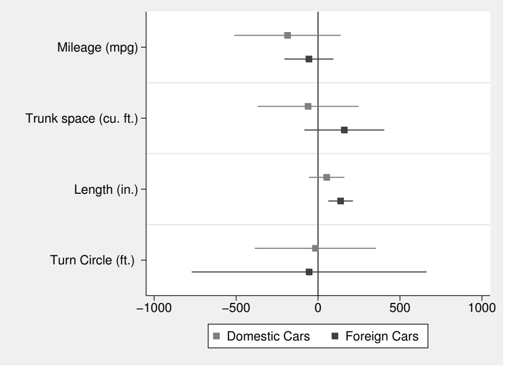
A basic example is as follows:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
>    if foreign==0
  (output omitted)
. estimates store D
. regress price mpg trunk length turn
>    if foreign==1
  (output omitted)
. estimates store F
. coefplot D F, drop(_cons) xline(0)
```



To specify separate options for the individual plots, enclose the models and their options in parentheses. For example, to add a label for each plot in the legend, to use alternative plot styles, and to change the marker symbol, type

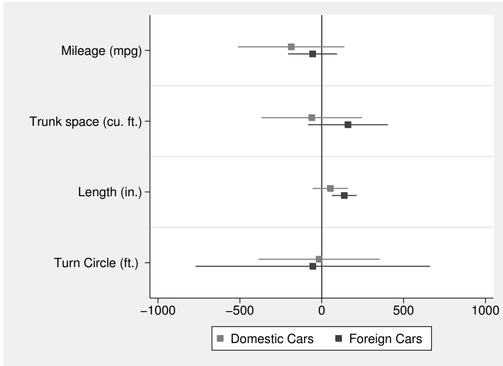
```
. coefplot (D, label(Domestic Cars)
>              pstyle(p3))
>          (F, label(Foreign Cars)
>              pstyle(p4)),
>          msymbol(S) drop(_cons) xline(0)
```



In the example, `msymbol()` was specified as a global option so that the same symbol is used in both plots. To use different symbols, include an individual `msymbol()` option for each plot.

`coefplot` offsets the plot positions of the coefficients so that the confidence spikes do not overlap. To deactivate the automatic offsets, one can specify the global option `nooffsets`. Alternatively, one can specify custom offsets by using the `offset()` option (if `offset()` is specified for at least one plot, automatic offsets are disabled). The spacing between coefficients is one unit, so usually offsets between -0.5 and 0.5 make sense. For example, to use smaller offsets than the default, type

```
. coefplot (D, label(Domestic Cars)
>              pstyle(p3) offset(0.05))
>          (F, label(Foreign Cars)
>              pstyle(p4) offset(-0.05)),
>          msymbol(S) drop(_cons) xline(0)
```



4.2 Subgraphs

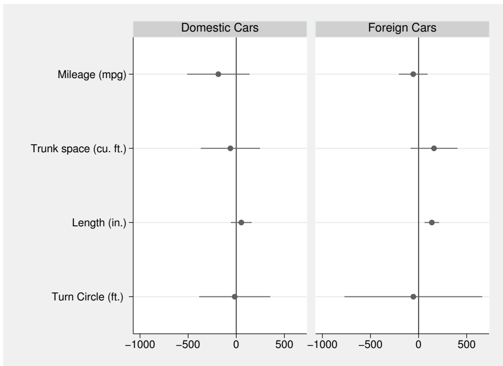
The syntax to create subgraphs is

`coefplot` *plotlist*[, *subgropts*] `||` [*plotlist*, *subgropts* `||` ...] [, *globalopts*]

where *plotlist* is a list of plots as in section 4.1, and *subgropts* are options that apply to a single subgraph. For details about `coefplot` syntax, see `help coefplot`.

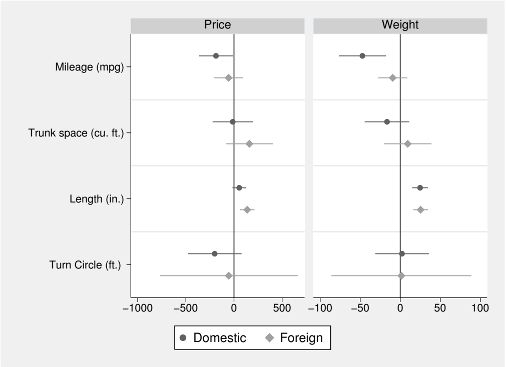
An example with one model per subgraph is

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
>    if foreign==0
  (output omitted)
. estimates store D
. regress price mpg trunk length turn
>    if foreign==1
  (output omitted)
. estimates store F
. coefplot D, bylabel(Domestic Cars)
>       || F, bylabel(Foreign Cars)
>       ||, drop(_cons) xline(0)
```



An example with multiple models per subgraph is

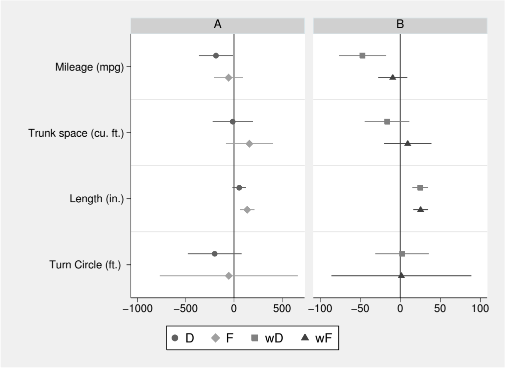
```
. regress weight mpg trunk length turn
>     if foreign==0
  (output omitted)
. estimates store wD
. regress weight mpg trunk length turn
>     if foreign==1
  (output omitted)
. estimates store wF
. coefplot
>     (D, label(Domestic))
>     (F, label(Foreign)), bylabel(Price)
>     || wD wF, bylabel(Weight)
>     ||, drop(_cons) xline(0)
>         byopts(xrescale)
```



Option `byopts(xrescale)` was specified so that each subgraph can have its own scale.

In the example above, plot labels for the legend were set within the first subgraph. They could also have been specified within the second subgraph, because plot styles are recycled with each new subgraph and plot options are collected across subgraphs. To prevent recycling of plot styles, add the `norecycle` option, as follows:

```
. coefplot D F, bylabel(A)
>     || wD wF, bylabel(B)
>     ||, drop(_cons) xline(0)
>       norecycle byopts(xrescale)
>       legend(rows(1))
```



4.3 Appending models

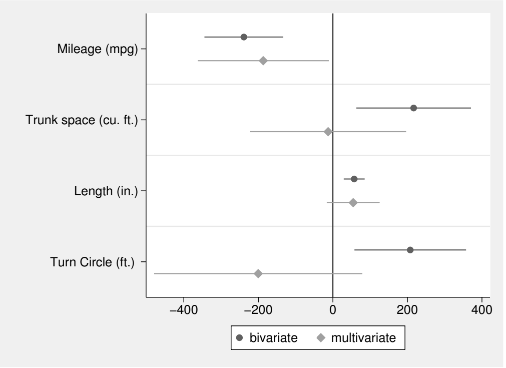
The syntax to append models within the same plot is

`coefplot` (*name*[, *modelopts*] \ [*name*, *modelopts* \ ...] [, *plotopts*]) [...]

where *name* is the name of a stored model, or `.` or empty string denoting the active model, and *modelopts* are options that apply to a single model. For details about `coefplot` syntax, see `help coefplot`.

For example, to draw a graph comparing bivariate and multivariate effects, type

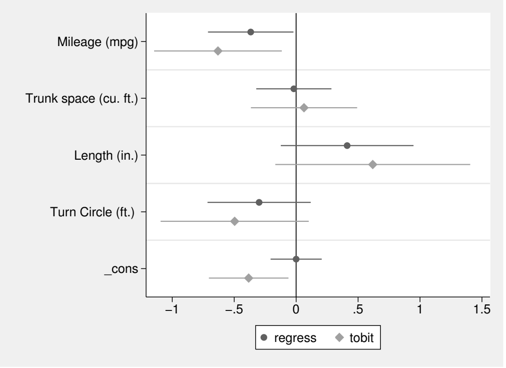
```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. estimates store multivariate
. foreach var in mpg trunk length turn {
  2.     quietly regress price `var´
  3.     estimates store `var´
  4. }
. coefplot (mpg \ trunk \ length \ turn,
>     label(bivariate)) (multivariate),
>     drop(_cons) xline(0)
```



4.4 How coefficients and equations are matched

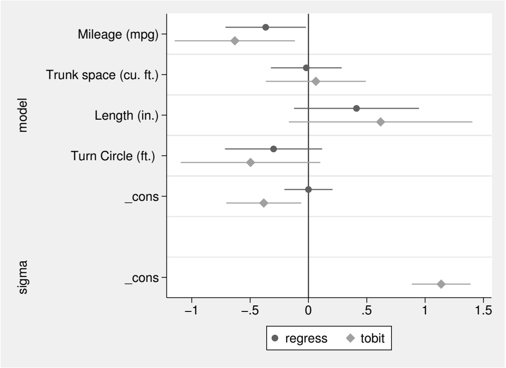
The default for `coefplot` is to use the first (nonzero) equation from each model and match coefficients across models by their names (ignoring equation names). For example, `regress` returns one (unnamed) equation containing the regression coefficients whereas `tobit` (see [R] **tobit**) returns two equations, equation "`model`" containing the regression coefficients and equation "`sigma`" containing the standard error of the regression. Hence, the default for `coefplot` is to match the regression coefficients from the two models and ignore equation "`sigma`" from the tobit model.

```
. sysuse auto, clear
(1978 Automobile Data)
. foreach v of var price mpg trunk length
>     turn {
  2.     quietly summarize `v´
  3.     quietly replace `v´ =
>            (`v´ - r(mean)) / r(sd)
  4. }
. regress price mpg trunk length turn
  (output omitted)
. estimate store regress
. tobit price mpg trunk length turn,
>     ll(-.5)
  (output omitted)
. estimate store tobit
. coefplot regress tobit, xline(0)
```



To include the second equation from the tobit model, you can add option `keep(*:)` (indicating that all equations are to be kept). However, as soon as more than one equation is collected per model, equation names start to matter and coefficients will be matched within equations. Therefore, you may want to assign the equation name "`model`" to the results from `regress` so that the coefficients from the two models are matched into the same equation.

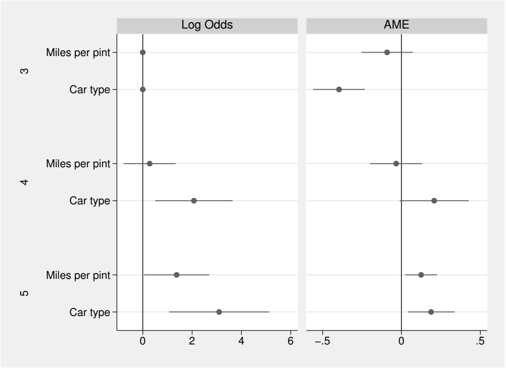
```
. coefplot (regress, asequation(model))
>         (tobit, keep(*:)),
>         xline(0)
```



Alternatively, you could also use `eqrename(_ = model)` to rename equation "`_`" to "`model`" or `eqrename(model = _)` to rename equation "`model`" to "`_`".

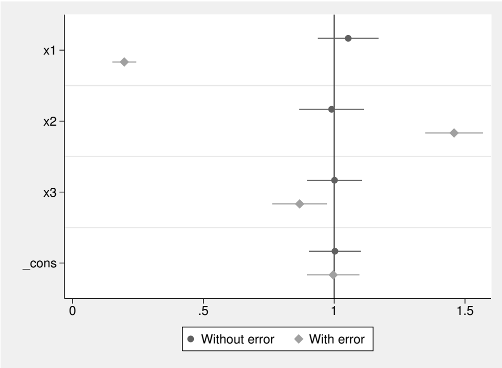
The option `asequation()` can also be applied when you want to assign equations to results from `margins`. In the following example, I show how to plot log odds of a multinomial logit (see [R] **mlogit**) along with average marginal effects:

```
. sysuse auto, clear
(1978 Automobile Data)
. gen mpp = mpg/8
. label variable mpp "Miles per pint"
. mlogit rep78 mpp foreign if rep>=3
  (output omitted)
. estimates store mlogit
. forvalues i = 3/5 {
  2.     quietly margins, dydx(*)
>            predict(outcome(`i´)) post
  3.     estimates store ame`i´
  4.     quietly estimates restore mlogit
  5. }
```




```
. coefplot mlogit, keep(*:) drop(_cons) omitted bylabel(Log Odds)
>       || (ame3, aseq(3) \ ame4, aseq(4) \ ame5, aseq(5)), bylabel(AME)
>       ||, xline(0) byopts(xrescale)
```

Finally, if you want to match coefficients that have different names in the input models, you can apply the `rename()` option. I use the following example to illustrate the effect of measurement error in regression models:



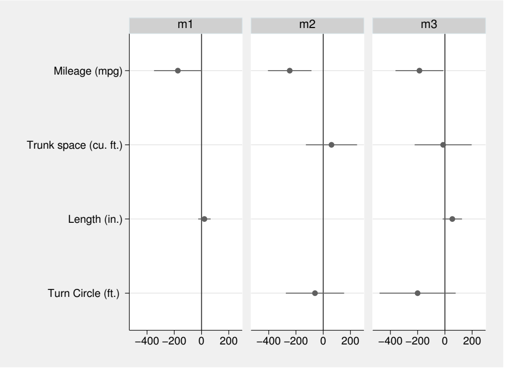

```
. drop _all
. matrix C = (  1, .5,  0 \ .5,  1, .3
>             \  0, .3,  1 )
. drawnorm x1 x2 x3, n(10000) corr(C)
(obs 10000)
. generate y = 1 + x1 + x2 + x3 +
>              5 * invnorm(uniform())
. regress y x1 x2 x3
  (output omitted)
. estimates store m1
. generate x1err = x1 +
>              2 * invnorm(uniform())
. regress y x1err x2 x3
  (output omitted)
. estimates store m2
. coefplot (m1, label(Without error)) (m2, label(With error)),
>     xline(1) rename(x1err = x1)
```

We can see how measurement error on `x1` distorts all slope coefficients in the model, even for variable `x3` that is uncorrelated with `x1` (due to the indirect correlation through `x2`).

4.5 How coefficients are ordered

In general, coefficients are plotted in the same order (from top to bottom) as they appear in the input models. However, coefficients appearing only in later models are placed after coefficients from earlier models (with the exception of `_cons`, which is always placed last). To arrange the coefficients in a different order, you can use the `order()` option, as in the following example:

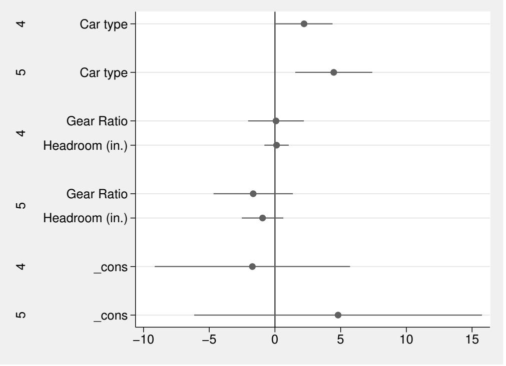
```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg length
  (output omitted)
. estimate store m1
. regress price mpg trunk turn
  (output omitted)
. estimate store m2
. regress price mpg trunk length turn
  (output omitted)
. estimate store m3
. coefplot m1 || m2 || m3, xline(0)
>     drop(_cons) byopts(row(1)) order(mpg trunk length)
```



Within `order()`, you can use the `*` (any string) and `?` (any nonzero character) wildcards. Furthermore, you can type `.` to insert gaps (but also see the section on headings and groups below).

In case of multiple equation models, the default is to order coefficients by equations. To reorder equations, to apply different orderings within equations, or to break equations apart, specify equation names within `order()`, as in the following example:

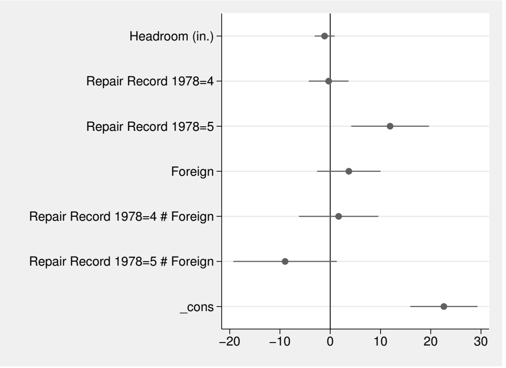
```
. sysuse auto, clear
(1978 Automobile Data)
. mlogit rep78 headroom gear_ratio foreign
>     if rep>=3
  (output omitted)
. coefplot, xline(0) keep(*:)
>     order(4:foreign 5:foreign
>           4:gear* head* 5:gear* head*
>           4: 5:)
```



5 Labeling the categorical axis

`coefplot` looks for variables that correspond to the collected coefficient names and then uses their variable labels for the categorical axis. For factor variables, `coefplot` additionally takes value labels into account (the rule is to print the value label, if a value label is defined, and otherwise print the variable label or name along with the level). The following is an example with categorical variables and interaction terms:

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0)
```

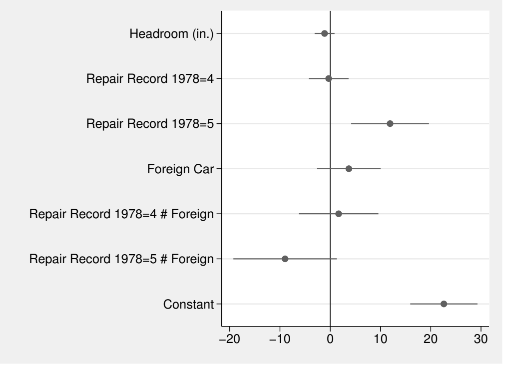


To use coefficient names instead of variable labels, specify the `nolabels` option.

5.1 Custom coefficient labels

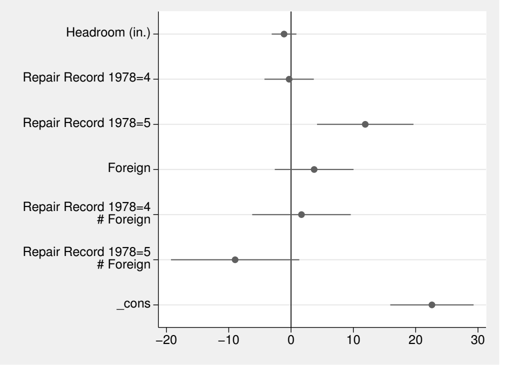
An easy way to provide labels for the coefficients is to define appropriate variable and value labels before applying `coefplot`; see [D] **label**. However, not all coefficients have corresponding variables (for example, `_cons`). To provide labels for such coefficients or to assign custom labels to coefficients without manipulating variable labels, use the `coeflabels()` option.

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0)
>     coeflabel(1.foreign = "Foreign Car"
>                    _cons = "Constant")
```



`coeflabels()` has a `wrap()` and a `truncate()` suboption to deal with long labels. These suboptions apply to all coefficient labels, whether they are automatically generated or provided within `coeflabels()`. For example, to limit the line to 20 characters and wrap long labels to multiple lines, type

```
. coefplot, xline(0) coeflabel(, wrap(20))
```

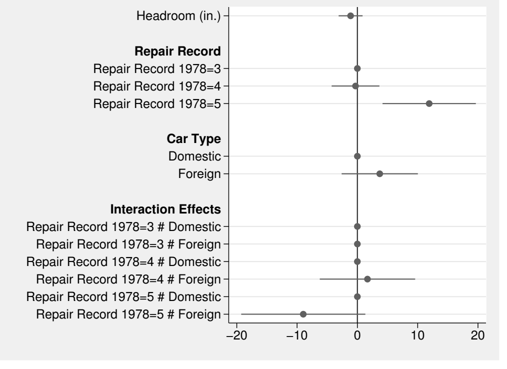


Multiline labels can also be created using compound double quotes, for example, `coeflabels(1.foreign = ‘""Line 1" "Line 2""’)`. Such labels will not be altered by `wrap()` or `truncate()`.

5.2 Headings and groups

Sometimes it is useful to add headings between coefficients to better arrange a graph. This can be achieved by using the `headings()` option.

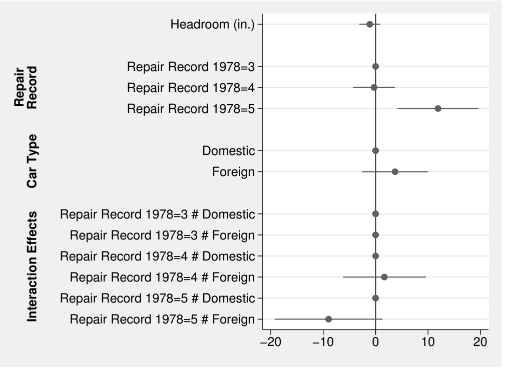
```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0) omitted baselevels
>     headings(
>     3.rep78 = "{bf:Repair Record}"
>     0.foreign = "{bf:Car Type}"
>     3.rep78#0.foreign =
>         "{bf:Interaction Effects}")
>     drop(_cons)
```



In this example, `omit` requests to plot omitted coefficients and `baselevels` requests to plot base-level coefficients. Omitted coefficients and base-level coefficients are always equal to 0, but it can sometimes be helpful to include them in a graph for clarity. The `{bf}` tag changes text to bold; see [G-4] ***text*** for details on text in graphs.

In addition to headings, you can also define groups of coefficients and add group labels using the `groups()` option as follows:

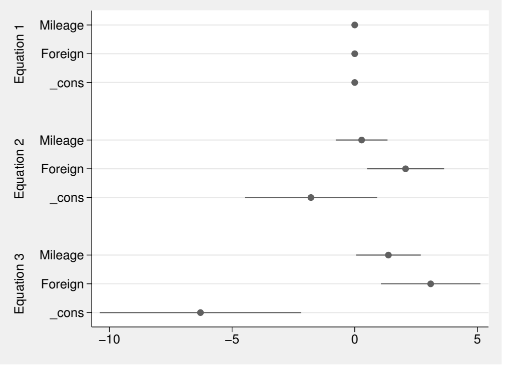
```
. coefplot, xline(0) omitted base
>     groups(?.rep78 =
>         `""{bf:Repair}" "{bf:Record}""´
>     ?.foreign = "{bf:Car Type}"
>     ?.rep78#?.foreign =
>         "{bf:Interaction Effects}")
>     drop(_cons)
```



5.3 Equation labels

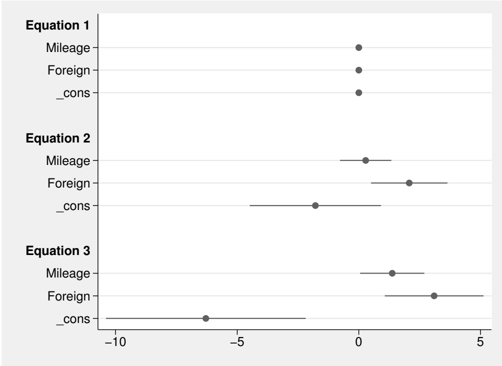
Equation labels provide yet another layer of labels. The default is to place the equation labels on the right-hand side, similar to group labels.

```
. sysuse auto, clear
(1978 Automobile Data)
. gen mpp = mpg/8
. mlogit rep78 mpp i.foreign if rep>=3
  (output omitted)
. coefplot, omitted keep(*:)
>     coeflabels(mpp = "Mileage")
>     eqlabels("Equation 1" "Equation 2"
>         "Equation 3")
```



However, you can also set the equation labels as headings between equations by using the `asheadings` suboption as follows:

```
. coefplot, omitted keep(*:)
>     coeflabels(mpp = "Mileage")
>     eqlabels("{bf:Equation 1}"
>         "{bf:Equation 2}"
>         "{bf:Equation 3}", asheadings)
```

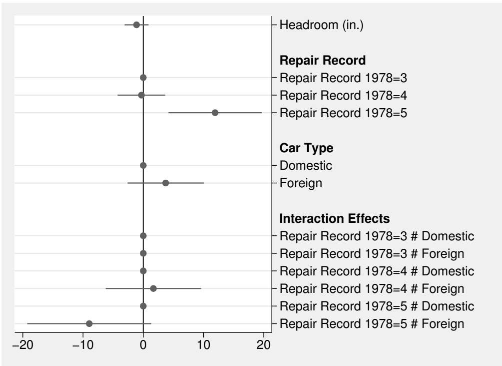


In this case, the `headings()` option is not allowed.

5.4 Labels on opposite side

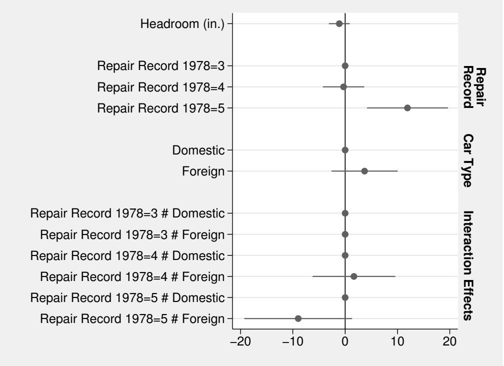
The default is to plot all labels on the left of the plot region. Use option `yscale(alt)` to move labels to the right (see [G-3] ***twoway_options***).

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0) omitted baselevels
>     headings(
>     3.rep78 = "{bf:Repair Record}"
>     0.foreign = "{bf:Car Type}"
>     3.rep78#0.foreign =
>         "{bf:Interaction Effects}")
>     drop(_cons) yscale(alt)
```



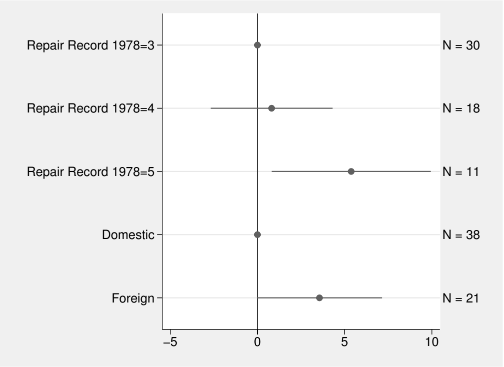
Group labels and equation labels are rendered as additional axes (axis 2 for group labels; axis 2 or 3 for equation labels, depending on whether groups were specified), so you have to use the `axis()` suboption to move these.

```
. coefplot, xline(0) omitted base
>     groups(?.rep78 =
>         `""{bf:Repair}" "{bf:Record}""'
>     ?.foreign = "{bf:Car Type}"
>     ?.rep78#?.foreign =
>         "{bf:Interaction Effects}",
>         angle(rvertical))
>     drop(_cons) yscale(alt axis(2))
```



Moving group labels to the right can also be useful if you want to add an extra set of coefficient labels without actually forming groups. The following is an example in which `groups()` is used to add information on the sample sizes of factor levels:

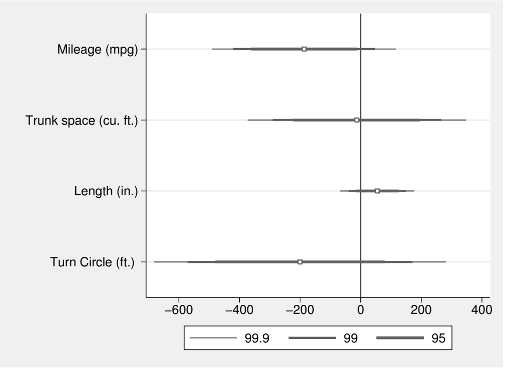
```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg i.rep i.foreign
  (output omitted)
. coefplot, xline(0) omitted baselevels
>     groups(3.rep78 = "N = 30"
>            4.rep78 = "N = 18"
>            5.rep78 = "N = 11"
>            0.foreign = "N = 38"
>            1.foreign = "N = 21",
>            nogap angle(horizontal))
>    drop(_cons) yscale(alt axis(2))
```



6 Confidence intervals

The default for `coefplot` is to draw spikes for 95% confidence intervals (or as set by `set level`; see [R] **level**). To specify a different level or to include multiple confidence intervals, use the `levels()` option. Here is an example with 99.9%, 99%, and 95% confidence intervals:

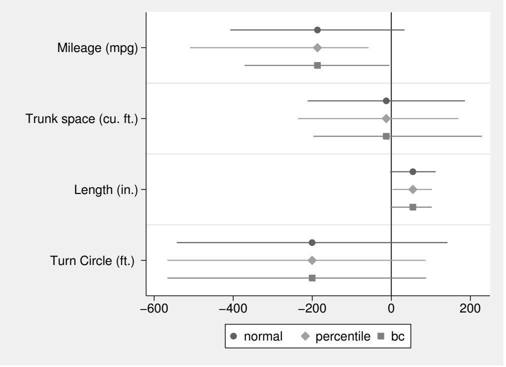
```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) xline(0)
>     msymbol(s) mfcolor(white)
>     levels(99.9 99 95)
>     legend(order(1 "99.9" 2 "99" 3 "95")
>         row(1))
```



Line widths are (logarithmically) increased across the confidence intervals. To use different line widths, specify, for example, `ciopts(lwidth(*1 *2 *4))`.

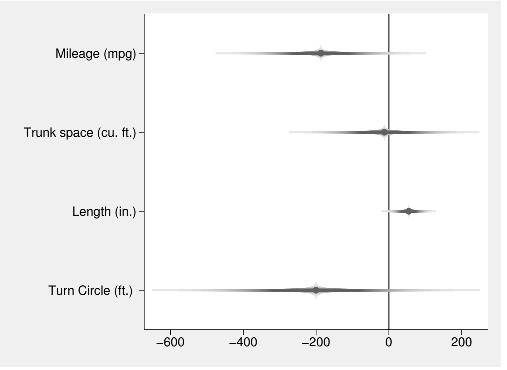
To compute confidence intervals, `coefplot` collects the variances of the coefficients from the diagonal of `e(V)` and then, depending on whether degrees of freedom is available in scalar `e(df_r)` (or, for estimates from [MI] **intro**, in matrix `e(df_mi)`), applies the standard formulas for confidence intervals on the basis of the t distribution or the normal distribution, respectively. If a model does not provide degrees of freedom but you want to compute confidence intervals by using the t distribution, you can provide the degrees of freedom through option `df()` (see the online help). If variances are stored in a matrix other than `e(V)`, use the `v()` option to provide the appropriate matrix name, or use option `se()` to provide custom standard errors (in which case variances from `e(V)` will be ignored). Likewise, if your estimation command provides precomputed confidence intervals, use the `ci()` option to include them in the plot. For example, to plot the normal-approximation, percentile, and bias-corrected confidence intervals that are provided in `e(ci_normal)`, `e(ci_percentile)`, and `e(ci_bc)` by the bootstrap method, you could type

```
. regress price mpg trunk length turn,
>     vce(bootstrap)
  (output omitted)
. coefplot
>     (, ci(ci_normal) label(normal))
>     (, ci(ci_percentile)
>         label(percentile))
>     (, ci(ci_bc) label(bc)),
>     drop(_cons) xline(0) legend(row(1))
```



In addition to `levels()` and `ci()`, you can also use option `cismooth` to add smoothed confidence intervals.[2] By default, `cismooth` generates confidence intervals for 50 equally spaced levels (1, 3, ..., 99) with graduated color intensities and varying line widths, as illustrated in the following example:

```
. coefplot, drop(_cons) xline(0)
>     cismooth grid(none)
```



The smoothed confidence intervals are produced independently from `levels()` and `ci()` and are not affected by `ciopts()`. Their appearance, however, can be set by several suboptions (see the online help). If `cismooth` is specified together with `levels()` or `ci()`, then the smoothed confidence intervals are placed behind the confidence intervals from `levels()` or `ci()`.

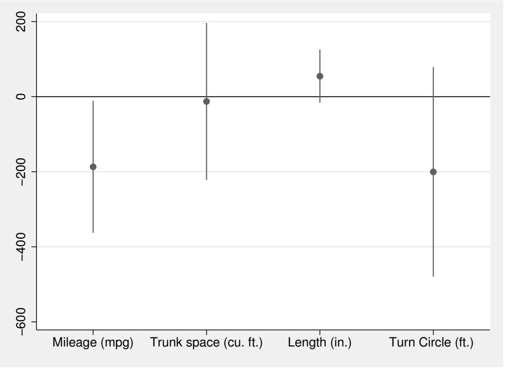
7 Alternate plot types and advanced examples

7.1 Vertical mode

By default, `coefplot` produces a horizontal graph with labels on the y axis and values on the x axis. To flip axes, specify the `vertical` option.

2. The `cismooth` option has been inspired by code by David B. Sparks to produce smoothed confidence interval plots in R (see http://dsparks.wordpress.com/2011/02/21/choropleth-tutorial-and-regression-coefficient-plots/).

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) vertical yline(0)
```



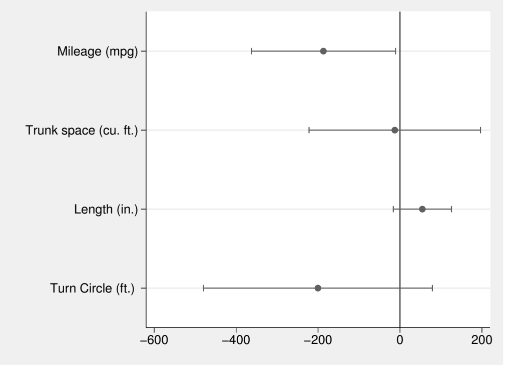
When changing from horizontal to vertical mode, options referring to specific axes must be adjusted. This is why `yline(0)` was used in the example instead of `xline(0)` to draw the 0 line.

7.2 Using the recast() option

To change the plot types used for markers and confidence intervals, you can use the `recast()` option. Available plot types for markers are standard twoway plots such as `scatter` (the default), `line`, `dot`, or `bar`. For confidence intervals, use range plots such as `rspike` (the default), `rline`, `rcap`, or `rbar`.

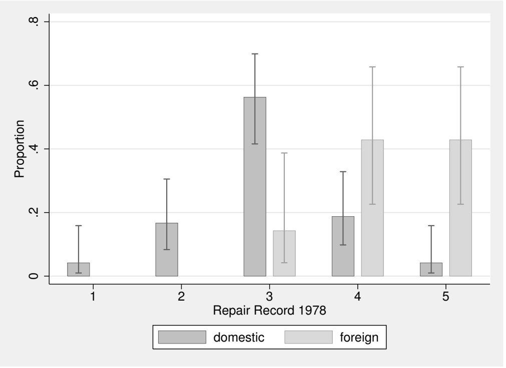
For example, to display confidence intervals using capped spikes, you could type

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) xline(0)
>     ciopts(recast(rcap))
```



Furthermore, a bar chart of proportions with capped confidence spikes can be produced as follows:

```
. sysuse auto, clear
(1978 Automobile Data)
. proportion rep if foreign==0
  (output omitted)
. estimates store domestic
. proportion rep if foreign==1
  (output omitted)
. estimates store foreign
. coefplot domestic foreign,
>     vertical recast(bar)
>     barwidth(0.25) fcolor(*.5)
>     ciopts(recast(rcap)) citop
>     citype(logit)
>     xtitle(Repair Record 1978) ytitle(Proportion)
```

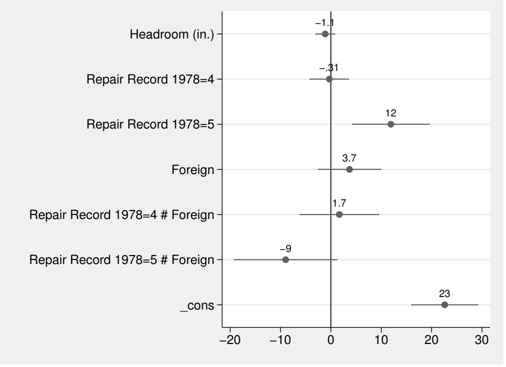


In this example, the `citop` option was used to prevent the lower limits of the confidence intervals from being hidden behind the bars. Furthermore, the `citype(logit)` option was specified to compute confidence intervals using the logit transformation, as is appropriate for proportions (see [R] **proportion**).

7.3 Adding marker labels

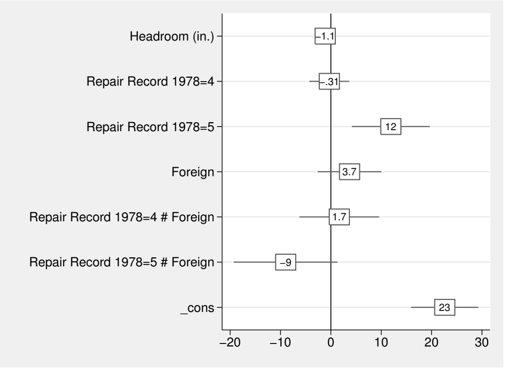
To add the values of the coefficients as marker labels, use the `mlabel` option, possibly together with `format()` to set the display format.

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0) mlabel format(%9.2g)
>     mlabposition(12) mlabgap(*2)
```



Stata graphs do not support background colors for marker labels, which makes labels unreadable if you place them on top of the markers using `mlabposition(0)`. However, the following is a workaround:

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. mata: st_matrix("e(box)",
>     (st_matrix("e(b)") :- 2 \
>      st_matrix("e(b)") :+ 2))
. coefplot, xline(0) mlabel format(%9.2g)
>     mlabposition(0) msymbol(i)
>     ci(95 box) ciopts(recast(. rbar)
>         barwidth(. 0.35) fcolor(. white)
>         lwidth(. medium))
```



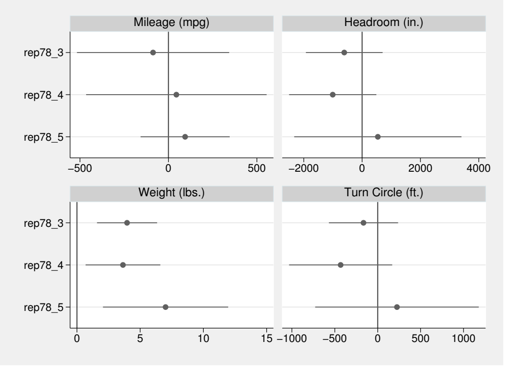
The trick is to add a second "confidence interval" that is a bar of fixed width. The dot in each suboption within `ciopts()` specifies the "default" style (see [G-4] ***stylelists***).

7.4 Arranging subgraphs by coefficients

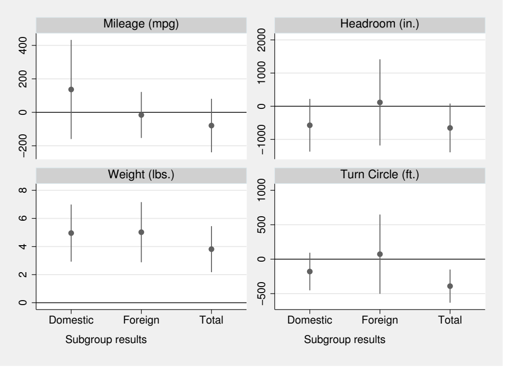
It is often sensible to arrange coefficients in separate subgraphs with individual scales, because the size of coefficients may vary considerably. For example, when comparing results by subgroups or estimation techniques, the focus is usually more on differences across models and less on differences within models, so it appears natural to use individual subgraphs for the different coefficients (see Gelman, Pasarica, and Dodhia [2002]).

Creating subgraphs by coefficients requires lengthy commands, because a separate piece of subgraph syntax must be put together for each coefficient. To circumvent the extra typing, you can use the `bycoefs` option. Technically, `bycoefs` flips coefficients and subgraphs, that is, the coefficients are treated as subgraphs and what was specified as subgraphs is treated as coefficients. This seems difficult to understand, but it should become clear in the following example:

```
. sysuse auto, clear
(1978 Automobile Data)
. forv i = 3/5 {
  2.     quietly regress price mpg
>            headroom weight turn
>            if rep78==`i´
  3.     estimate store rep78_`i´
  4. }
. coefplot rep78_3 || rep78_4 || rep78_5,
>     drop(_cons) xline(0)
>     bycoefs byopts(xrescale)
```



If `bycoefs` is specified, options such as `headings()` and `groups()` apply to the elements on the categorical axis (instead of coefficients). To address the elements, use integer numbers `1`, `2`, `3`, etc., as in the following example:



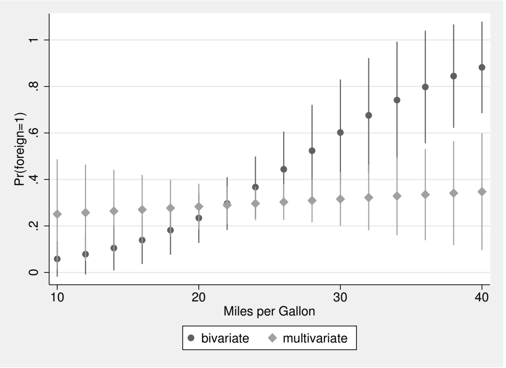

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg headroom weight turn
  (output omitted)
. estimates store Total
. regress price mpg headroom weight turn
>     if foreign==0
  (output omitted)
. estimates store Domestic
. regress price mpg headroom weight turn
>     if foreign==1
  (output omitted)
. estimates store Foreign
. coefplot Domestic || Foreign || Total, drop(_cons) yline(0) vertical
>     bycoefs byopts(yrescale)
>     group(1 2 = "Subgroup results", nogap) ylabel(0, add)
```

Option `ylabel(0, add)` was added to ensure that the 0 baseline is included in each subgraph.

7.5 Using a continuous axis

Coefficients provided to `coefplot` may represent estimates along a continuous dimension. Examples are predictive margins or marginal effects computed over values of a continuous variable. In such a case, one can use the `at()` option to provide the plot positions to `coefplot`. Here is an example where predictive margins of `foreign` are computed by level of `mpg`, once from a bivariate model and once from a multivariate model:

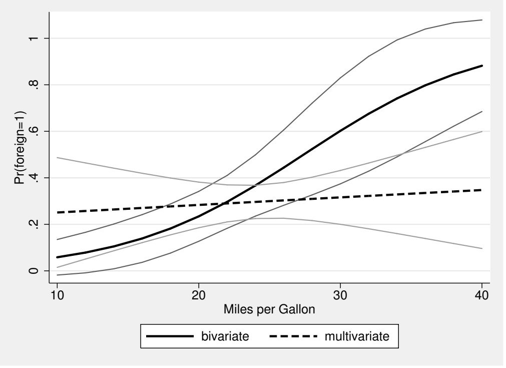
```
. sysuse auto, clear
(1978 Automobile Data)
. logit foreign mpg
  (output omitted)
. margins, at(mpg=(10(2)40)) post
  (output omitted)
. estimates store bivariate
. logit foreign mpg turn price
  (output omitted)
. margins, at(mpg=(10(2)40)) post
  (output omitted)
. estimates store multivariate
. coefplot bivariate multivariate, at
>     ytitle(Pr(foreign=1)) xtitle(Miles per Gallon)
```



at() causes coefplot to use a continuous axis with default labeling for the plotted estimates instead of compiling a categorical axis. It also causes coefplot to switch to vertical mode, because this is more common for such plots. Because no categorical axis is constructed if at() is specified, options such as order(), coeflabels(), headings(), and groups() are not allowed. Furthermore, continuous and categorical mode cannot be mixed. That is, at() must be specified for all models or for none. In the example above, at was used without argument. This is suitable for results provided by margins, because coefplot contains special code to retrieve the plot positions in this case. See the online help for alternative applications of at().

coefplot does not change the plot type for markers and confidence intervals, and hence still draws dots and spikes. Use option recast() to change this, for example, as follows:

```
. coefplot (bivariate) (multivariate), at
>     ytitle(Pr(foreign=1))
>     xtitle(Miles per Gallon)
>     recast(line) lwidth(*2)
>     ciopts(recast(rline))
```



7.6 Plotting results from matrices

Finally, to plot results from a matrix (see [P] **matrix**) instead of the `e()` returns, use syntax

`coefplot` [`(`]`matrix(`*mspec*`)`[`,` *modelopts*] [...][`)`] [...]

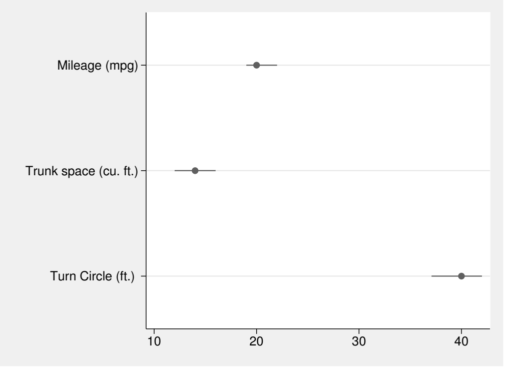
where *mspec* is

name	get point estimates from first row of matrix *name*
name`[`#`,.]`	get point estimates from row # of matrix *name*
name`[.,`#`]`	get point estimates from column # of matrix *name*

For details about `coefplot` syntax, see `help coefplot`.

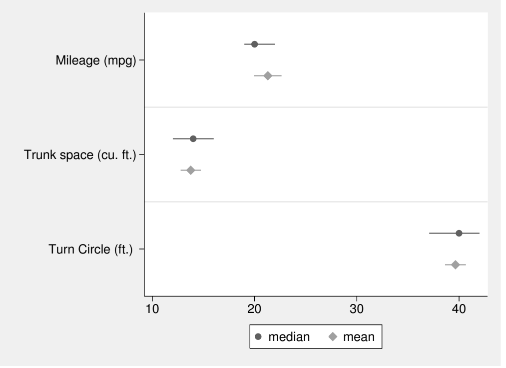
In this case, names given in options such as `at()` or `ci()` will also be interpreted as matrix names. For example, to plot medians and their confidence intervals as computed by `centile` (see [R] **centile**), you could type

```
. sysuse auto, clear
(1978 Automobile Data)
. matrix res = J(3,3,.)
. matrix coln res = median ll95 ul95
. matrix rown res = mpg trunk turn
. local i 0
. foreach v of var mpg trunk turn {
  2.     local ++ i
  3.     quietly centile `v´
  4.     matrix res[`i´,1] = r(c_1),
>         r(lb_1), r(ub_1)
  5. }
. matrix list res
res[3,3]
          median       ll95       ul95
  mpg         20         19         22
trunk         14         12         16
 turn         40  37.078729         42
. coefplot matrix(res[.,1]), ci((res[.,2] res[.,3]))
```



A single `coefplot` command can contain both regular syntax and `matrix()` syntax. For example, to add means to the graph above, you could proceed as follows:

```
. mean mpg trunk turn
  (output omitted)
. estimates store mean
. coefplot (matrix(res[,1]), label(median)
>         ci((res[,2] res[,3])))
>     (mean)
```



8 Acknowledgments

I thank Nick Cox, Marc Höglinger, and an anonymous reviewer for their helpful suggestions.

9 References

Chambers, J. M., W. S. Cleveland, B. Kleiner, and P. A. Tukey. 1983. *Graphical Methods for Data Analysis*. Belmont, CA: Wadsworth.

Chapin, F. S. 1924. The statistical definition of a societal variable. *American Journal of Sociology* 30: 154–171.

Cleveland, W. S. 1993. *Visualizing Data*. Summit, NJ: Hobart.

———. 1994. *The Elements of Graphing Data*. Rev. ed. Summit, NJ: Hobart.

Cleveland, W. S., and R. McGill. 1985. Graphical perception and graphical methods for analyzing scientific data. *Science* 229: 828–833.

Dice, L. R., and H. J. Leraas. 1936. A graphic method for comparing several sets of measurements. *Contributions from the Laboratory of Vertebrate Genetics* 3: 1–3.

Gallup, J. L. 2012. A new system for formatting estimation tables. *Stata Journal* 12: 3–28.

Gelman, A., C. Pasarica, and R. Dodhia. 2002. Let's practice what we preach: Turning tables into graphs. *American Statistician* 56: 121–130.

Gosset [Student, pseud.], W. S. 1927. Errors of routine analysis. *Biometrika* 19: 151–164.

Harrell, F. E., Jr. 2001. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis.* New York: Springer.

Healy, K., and J. Moody. 2014. Data visualization in sociology. *Annual Review of Sociology* 40: 105–128.

Jacoby, W. G. 1997. *Statistical Graphics for Univariate and Bivariate Data.* Thousand Oaks, CA: Sage.

Jann, B. 2007. Making regression tables simplified. *Stata Journal* 7: 227–244.

Kastellec, J. P., and E. L. Leoni. 2007. Using graphs instead of tables in political science. *Perspectives on Politics* 5: 755–771.

Lewandowsky, S., and I. Spence. 1989. The perception of statistical graphs. *Sociological Methods and Research* 18: 200–242.

Newson, R. B. 2003. Confidence intervals and *p*-values for delivery to the end user. *Stata Journal* 3: 245–269.

Tufte, E. R. 2001. *The Visual Display of Quantitative Information.* 2nd ed. Cheshire, CT: Graphics Press.

Wainer, H. 1997. *Visual Revelations: Graphical Tales of Fate and Deception from Napoleon Bonaparte to Ross Perot.* New York: Copernicus.

About the author

Ben Jann is professor of sociology at the University of Bern, Switzerland. His research interests include social-science methodology, statistics, social stratification, and labor market sociology. Recent publications include articles in *Sociological Methodology*, *Sociological Methods and Research*, *Public Opinion Quarterly*, and the *American Sociological Review*.