STATA TECHNICAL BULLETIN

September 1999

STB-51

A publication to promote communication among Stata users

Editor

H. Joseph Newton
Department of Statistics
Texas A & M University
College Station, Texas 77843
409-845-3142
409-845-3144 FAX
stb@stata.com EMAIL

Associate Editors

Nicholas J. Cox, University of Durham
Francis X. Diebold, University of Pennsylvania
Joanne M. Garrett, University of North Carolina
Marcello Pagano, Harvard School of Public Health
J. Patrick Royston, Imperial College School of Medicine

Subscriptions are available from Stata Corporation, email stata@stata.com, telephone 979-696-4600 or 800-STATAPC, fax 979-696-4601. Current subscription prices are posted at www.stata.com/bookstore/stb.html.

Previous Issues are available individually from StataCorp. See www.stata.com/bookstore/stbj.html for details.

Submissions to the STB, including submissions to the supporting files (programs, datasets, and help files), are on a nonexclusive, free-use basis. In particular, the author grants to StataCorp the nonexclusive right to copyright and distribute the material in accordance with the Copyright Statement below. The author also grants to StataCorp the right to freely use the ideas, including communication of the ideas to other parties, even if the material is never published in the STB. Submissions should be addressed to the Editor. Submission guidelines can be obtained from either the editor or StataCorp.

The *Stata Technical Bulletin* (ISSN 1097-8879) is published six times per year by Stata Corporation. Stata is a registered trademark of Stata Corporation.

Contents of this issue

dm50.1	Update to defv

John R. Gleason, Syracuse University, loesljrg@accucom.net

The command `defv` in Gleason (1997) has been updated to take advantage of Stata Version 6.0. In particular, this means that `defv` can now properly deal with definitions that contain embedded double-quote (‘"’) characters. Thus definitions resembling

```
. defv byte not_fem = (sex=="M") | (sex=="?")
```

are now acceptable. In addition, the syntax has been slightly expanded so that

```
. defv ?
```

will produce a brief reminder of proper usage by issuing the command ‘`which defv`’.

Reference

Gleason J. R. 1997. dm50: Defining variables and recording their definitions. *Stata Technical Bulletin* 40: 9–10. Reprinted in *Stata Technical Bulletin Reprints*, vol. 7, pp. 48–49.

dm56.1	Update to labedit

John R. Gleason, Syracuse University, loesljrg@accucom.net

The labels editor `labedit` for Windows and Macintosh (Gleason 1998) has been updated to take full advantage of Stata 6.0. This means that labels with embedded double-quote (‘"’) characters are now permitted, labels can be up to 80 characters long, value labels can be assigned to negative integers, and so on. Also a small bug introduced by other changes in Stata 6.0 has been fixed.

In addition, `labedit` will now load much faster when the varlist is long. `labedit` now indicates whether a variable has notes and provides a button to display them. Numerous other minor changes have been made in the layout of the dialog box.

Reference

Gleason J. R. 1998. dm56: A labels editor for Windows and Macintosh. *Stata Technical Bulletin* 43: 3–6. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 5–10.

dm61.1	Update to varxplor

John R. Gleason, Syracuse University, loesljrg@accucom.net

The visual tool `varxplor` for Windows and Macintosh (Gleason 1998) has been updated to take full advantage of Stata 6.0. This means that commands entered from `varxplor`’s simulated command line may now contain double-quote (‘"’) characters, as may the commands assigned to the five launch buttons. In addition, commands issued from the simulated command line are now recorded in a command history buffer, and there are two buttons that traverse the buffer in a manner similar to that of the *PrevLine* and *NextLine* keys for Stata’s (real) command line.

Reference

Gleason J. R. 1998. dm61: A tool for exploring Stata datasets (Windows and Macintosh only). *Stata Technical Bulletin* 45: 2–5. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 22–27.

dm66.2	Update of cut to Stata 6

David Clayton, MRC Biostatistical Research Unit, Cambridge, david.clayton@mrc-bsu.cam.ac.uk
Michael Hills (retired), mhills@regress.demon.co.uk

Versions of `cut` (Clayton and Hills 1999) for both Stata 5 and 6 are included in this STB. The Stata 6 version takes advantage of the new `numlist` concept. They are called by

```
. egen newvar=cutv5(var), ...
. egen newvar=cutv6(var), ...
```

respectively. To convert either to

```
. egen newvar=cut(var), ...
```

rename _gcutv*x*.ado to _gcut.ado and cutv*x*.hlp to cut.hlp and replace the program define line in _gcut.ado to

```
program define _gcut
```

An error in the way noninteger break points were treated has also been fixed.

Reference

Clayton, D. and M. Hills. 1999. dm66.1: Stata 6 version of recoding variables using grouped values. *Stata Technical Bulletin* 50: 3.

dm71 Calculating the product of observations

Philip Ryan, University of Adelaide, Australia, pryan@medicine.adelaide.edu.au

This insert describes a new option in egen which creates a new variable whose values are the product of observations of an expression.

Syntax

egen *newvar* = prod(*expression*) [if *exp*] [in *range*] [, by(*grouping varlist*) pmiss(ignore | missing | 1)]

Description

prod provides a multiplicative function for egen analogous to the additive sum function. The product of observations of *expression* meeting optional in and if conditions is returned in *newvar*. The *expression* would most commonly be an existing variable in the dataset. The underlying program file is _gprod.ado, the help file is prod.hlp.

Options

by(*grouping varlist*) specifies the variable(s) defining groups within which the products are calculated.

pmiss(ignore | missing | 1) specifies what is to be the product of a group at least one of whose observations are missing.

- pmiss(ignore), the default, ignores missing values and returns the product of all nonmissing values in the group. If all values are missing, then missing is returned.
- pmiss(missing) returns missing for the product of observations in a group if any observation in the group is missing.
- pmiss(1) returns 1 for the product of observations in a group if all observations in the group are missing. Otherwise it returns the product of all nonmissing values. This choice would rarely be made.

Note that a missing value is returned for observations excluded by if or in qualifiers. (See, for example, observations 16 on px0 below.)

Example

```
. use testprod
Contains data from testprod.dta
  obs:            22
 vars:             3                          6 Aug 1999 12:45
 size:           154 (100.0% of memory free)
------------------------------------------------------------------------------
   1. score     byte   %9.0g                  score
   2. gvar1     byte   %9.0g                  first group variable, 5 groups
   3. gvar2     byte   %9.0g                  second group variable nested in
                                                 gvar1, 2 groups
------------------------------------------------------------------------------
Sorted by:
. egen px = prod(score)
. egen px0 = prod(score) if score != 0
. egen px1i = prod(score), by(gvar1)
. egen px1m = prod(score), by(gvar1) pmiss(missing)
. egen px12i = prod(score), by(gvar1 gvar2) pmiss(ignore)
. egen px12m = prod(score), by(gvar1 gvar2) pmiss(missing)
```

```
. format px0 %12.0g   /*without this, get px0 in scientific notation*/
. listblck _all, rep(3)  /*requires -listblck.ado- (Weesie, STB-50)*/
        score      gvar1      gvar2         px          px0       px1i
 1.         3          1          1          0   7153920000       1200
 2.         4          1          1          0   7153920000       1200
 3.         5          1          1          0   7153920000       1200
 4.         1          1          2          0   7153920000       1200
 5.         2          1          2          0   7153920000       1200
 6.        10          1          2          0   7153920000       1200
 7.         4          2          1          0   7153920000       -360
 8.         5          2          1          0   7153920000       -360
 9.         6          2          1          0   7153920000       -360
10.        -3          2          2          0   7153920000       -360
11.        -1          3          1          0   7153920000        120
12.        -3          3          1          0   7153920000        120
13.         4          3          1          0   7153920000        120
14.         2          3          2          0   7153920000        120
15.         5          3          2          0   7153920000        120
16.         0          4          1          0            .          0
17.       -23          4          2          0   7153920000          0
18.         2          5          1          0   7153920000          6
19.         .          5          1          0   7153920000          6
20.         3          5          1          0   7153920000          6
21.         .          5          2          0   7153920000          6
22.         .          5          2          0   7153920000          6

        score      gvar1      gvar2       px1m      px12i      px12m
 1.         3          1          1       1200         60         60
 2.         4          1          1       1200         60         60
 3.         5          1          1       1200         60         60
 4.         1          1          2       1200         20         20
 5.         2          1          2       1200         20         20
 6.        10          1          2       1200         20         20
 7.         4          2          1       -360        120        120
 8.         5          2          1       -360        120        120
 9.         6          2          1       -360        120        120
10.        -3          2          2       -360         -3         -3
11.        -1          3          1        120         12         12
12.        -3          3          1        120         12         12
13.         4          3          1        120         12         12
14.         2          3          2        120         10         10
15.         5          3          2        120         10         10
16.         0          4          1          0          0          0
17.       -23          4          2          0        -23        -23
18.         2          5          1          .          6          .
19.         .          5          1          .          6          .
20.         3          5          1          .          6          .
21.         .          5          2          .          .          .
22.         .          5          2          .          .          .
```

Acknowledgments

I am grateful to Nick Cox, Tom Steichen, Clyde Schecter and, especially, Michael Blasnik for helpful comments and coding suggestions. Contributors to a thread on Statalist concerning handling of missing values in Stata functions are also thanked. Jeroen Weesie provided the handy utility `listblck` in STB-50.

Appendix

In a thread on Statalist Clyde Schecter (`clyde_schechter@smtplink.mssm.edu`) wrote (July 6, 1999) that

"From a mathematical standpoint, a product operator should obey the law: Prod(x in $A \cup B$) = Prod(x in A) Prod(x in B) if A and B are disjoint sets. From this it automatically follows that if A is the empty set, Prod(x in A) = 1."

However, this relationship depends upon B having no zero elements. If this assumption fails, Prod(x in A) will be indeterminate. `pmiss(1)` is provided for those who wish to make this assumption, but caution is advised.

dm72	Alternative ranking procedures

Nicholas J. Cox, University of Durham, UK, n.j.cox@durham.ac.uk
Richard Goldstein, richgold@ix.netcom.com

Syntax

```
egen [type] newvar = rankf(exp) [if exp] [in range] [, by(varlist)]

egen [type] newvar = rankt(exp) [if exp] [in range] [, by(varlist)]

egen [type] newvar = ranku(exp) [if exp] [in range] [, by(varlist)]

lbleqrnk varname [, suffix(suffix)]
```

Description

The `egen` functions `rankf( )`, `rankt( )` and `ranku( )` produce what are here called the "field," 'track," and "unique" ranks of *exp*, as alternatives to the ranks produced by the official `egen` function `rank( )`.

On-line help for these functions is available through `help altrank`.

`lbleqrnk` assigns value labels to tied ranks. The default label is of the form "*rank*=" whenever *rank* occurs twice or more.

Options

`by(`*varlist*`)` specifies that ranks are to be calculated separately for the distinct groups defined by *varlist*.

`suffix(`*suffix*`)` specifies a suffix for value labels indicating ties, for example, `suffix(" (tied)")`.

The existing rank function

Stata's `egen, rank( )` ranks what is fed to it according to two conventions commonly used in statistics:

a. The lowest value has rank 1.

b. Tied values are assigned the average of the ranks that would have been allocated had the values been minutely different. In other words, the sum of the ranks is preserved.

Hence a dataset of `0, 1, 2.71828, 3.14159` yields ranks `1, 2, 3, 4` and one of `1, 1, 1, 3.14159` yields ranks `2, 2, 2, 4`.

This insert explains how to calculate ranks in Stata according to other procedures that are sometimes used.

High values have low ranks

The opposite convention to (a) above is clearly

a′. The highest value has rank 1.

This seems more natural, for example, in dealing with the modeling of extreme values, such as floods and earthquakes, where scientific and statistical interest is concentrated on the highest flood discharges or earthquake magnitudes. The biggest flood in each set of data thus has rank 1, and not a rank that is equal to the sample size, which often depends arbitrarily on how data were produced or are available.

[R] **egen** explains that the syntax of `rank( )` is, at its simplest,

```
. egen newvar = rank(exp)
```

The key point here is that `rank( )` works on some expression *exp*; its argument is not restricted to a single variable name. Hence to get ranks according to convention (a′), an easy way is to insert a minus sign:

```
. egen newvar = rank(- varname)
```

Alternatively, make use of the fact that the rank of an observation in the reversed order is equal to the number of values plus one minus the rank of the observation in the usual order. (As an aside, note that it can be useful to be able to say things like

```
. egen newvar = rank(x - y)
```

without explicitly generating a new variable containing `x - y`.)

Field and track ranks

For some purposes, we do not want to adjust for ties using convention (b) above. In sports, in education, and in informal quantitative analysis, and in fact in most spheres outside formal statistical analysis, two contestants or candidates or cities that tie for second are described as "2nd equal," not as "rank 2.5." In general, the rank is just one plus the number of values "better." When lower values are better, we have what we may call the "track" convention: in track events, the winner had the lowest time. When higher values are better, we have similarly the "field" convention: in field events, the winner had the greatest jump or throw.

Given *myvar*, at its simplest, the code is for track ranks

```
. sort myvar
. gen rankt = _n
. qui replace rankt = rankt[_n-1] if myvar==myvar[_n-1]
```

and for field ranks

```
. gsort -myvar
. gen rankf = _n
. qui replace rankf = rankf[_n-1] if myvar==myvar[_n-1]
```

but to cope with common complications (`if`, `in`, `by( )`, and missing values) two ados defining `egen` functions accompany this insert. Strictly, only one of these is needed given that you can negate a variable, but the redundancy is not harmful.

If you wanted ranks of either field or track form, you might want to see them appearing in reports with some suitable suffix: for example, as `1=` or `1 (tied)`. The best way to achieve this is through value labels: `lbleqrnk` is supplied for this purpose.

Completed ranks

Tukey (1977, ch. 18) used what he called "completed ranks" in the analysis of some very right-skewed distributions. These are the number of values with at least that value, or the largest rank assigned to that value. The completed ranks of `1, 1, 1, 3.14159` are 4, 4, 4, 1. Such ranks can be computed from field ranks by

```
. sort myvar
. egen rankf = rankf(myvar)
. qui by myvar: replace rankf = rankf + _N - 1
```

An analog of completed ranks evidently could be defined in terms of "at most" rather than "at least," and it could be computed from track ranks similarly.

Unique ranks

Sometimes it is useful to flout the so-far tacit rule that equal values should be assigned the same rank. In our experience this arises only in graphing ranked data when equal values would otherwise be shown by coincident data points. Assigning each value a unique rank, so that `1, 1, 1, 3.1459` are assigned ranks `1, 2, 3, 4`, means that on a graph ties are evident by a line of data points. The idea of unique ranks is built into the official `quantile` command, although the variable in question is, strictly, fraction of the data. Mountain plots (Monti 1995; Goldstein 1996) provide a second and directly opposite example.

In Stata, the code is even simpler than in previous cases:

```
. sort myvar    OR    gsort -myvar
. gen ranku = _n
```

but once again to cope with common complications (`if`, `in`, `by( )`, and missing values) an ado defining an `egen` function accompanies this insert. This function `ranku( )` is a slight generalization of the `rank2( )` function included by Goldstein (1996) with the Stata code for mountain plots.

Examples

```
. use auto
(1978 Automobile Data)
. egen mpgrank = rankt(mpg)
. lbleqrnk mpgrank
```

```
. list make mpg mpgrank in 1/10
     make                      mpg    mpgrank
  1. Linc. Continental          12         1=
  2. Linc. Mark V               12         1=
  3. Linc. Versailles           14         3=
  4. Merc. XR-7                 14         3=
  5. Cad. Deville               14         3=
  6. Peugeot 604                14         3=
  7. Cad. Eldorado              14         3=
  8. Merc. Cougar               14         3=
  9. Buick Electra              15         9=
 10. Merc. Marquis              15         9=
. egen hdrrank = rankf(hdroom)
. lbleqrnk hdrrank
. list make hdroom hdrrank in 1/15
     make                 hdroom    hdrrank
  1. Plym. Volare            5.0          1
  2. Dodge St. Regis         4.5         2=
  3. Olds Cutlass            4.5         2=
  4. Buick Century           4.5         2=
  5. Olds Omega              4.5         2=
  6. Plym. Horizon           4.0         6=
  7. Chev. Impala            4.0         6=
  8. Olds Delta 88           4.0         6=
  9. Pont. Catalina          4.0         6=
 10. Buick Electra           4.0         6=
 11. Dodge Diplomat          4.0         6=
 12. Buick LeSabre           4.0         6=
 13. Dodge Magnum            4.0         6=
 14. Olds 98                 4.0         6=
 15. Cad. Deville            4.0         6=
```

References

Goldstein, R. 1996. sg58: Mountain plots. *Stata Technical Bulletin* 33: 9–10. Reprinted in *The Stata Technical Bulletin Reprints*, vol 6, pp. 143–145.

Monti, K. L. 1995. Folded empirical distribution function curves—mountain plots. *American Statistician* 49: 342–345.

Tukey, J. W. 1977. *Exploratory data analysis.* Reading, MA: Addison–Wesley.

gr39	3D surface plots

Adrian Mander, Cambridge, UK, adrian.mander@mrc-bsu.cam.ac.uk

This insert describes the command `surface` which draws a 3D wireform surface plot of three variables containing the x, y, and z coordinates for a set of points. The program is fairly rigid and does not allow axis labeling and allows only the default ticmarks, which are the maximum and minimum of each of the three variables.

The `surface` command can be used to add to the suite of 3D programs in van Melle (1998). His functions `hidlin` and `altitude` require a function $z = f(x, y)$ in order to generate a surface plot. In the same suite of programs the function `makfun` can summarize 3-dimensional data which can then be passed to `hidlin` and `altitude`. The function `surface` is an attempt to plot the exact data and relies upon no fixed functions or summaries of the data. Representation of 3-dimensional data is complicated and `surface` is a first step to improving Stata's capabilities.

Syntax

`surface` *var1 var2 var3* [`, saving(`*filename*`) round(#) orient(`*str*`) nowire`]

where *var1*, *var2*, and *var3* are the variables containing the x, y, and z coordinates of the points to be plotted.

Options

`saving(`*filename*`)` saves the resulting graph in *filename*. If the file already exists, it will be deleted and the new graph saved.

`round(#)` controls the number of lines drawn in the wireframe diagram when the data have numerous possible values for the x and y variables.

`nowire` substitutes the wireframe for points with lines that join the point and the minimum of the z variable.

`orient(`*str*`)` interchanges axes labels in order to simulate a rotation. This option takes the letters `xyz` or a combination of them. Whichever letter comes first is the x-axis, second is y-axis and third is the z-axis. Thus `orient(zxy)` means that *var1* is now the y coordinates, *var2* is the z coordinates and `var3` is the x coordinates. This is different from changing the variables around since the wireframe is still drawn across the original x and y values. This is a crude attempt to implement rotation to obtain a clearer picture.

Examples

The first set of data illustrates the basic features of `surface`. By default the three axes are labeled `x-axis`, `y-axis`, and `z-axis`. This is controlled by the order in which the three variables are in the command line. Regardless of which orientation is used the axes will be correctly labeled. The range of values on each of the axes is the minimum to the maximum of the respective variables.

```
. surface x y z
```

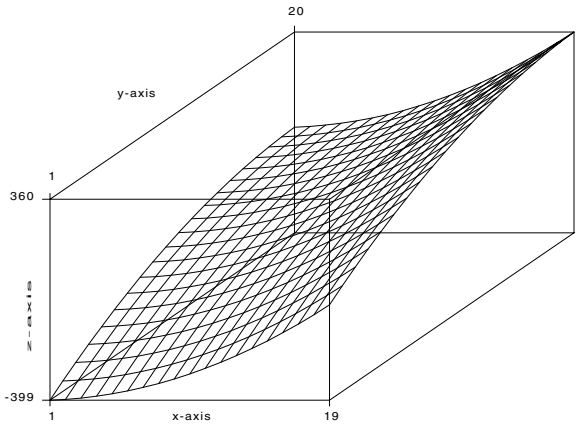

Figure 1: The default behavior of `surface`.

There is one alternative to the wireframe diagram and that is each point is plotted individually. The point is represented as a circle and the line represents the displacement from the minimum of the z variable. This command may be of most use when plotting an irregular grid of points, where the wireframe diagram becomes cluttered. Using the same data as in Figure 1, we have

```
. surface x y z, nowire
```

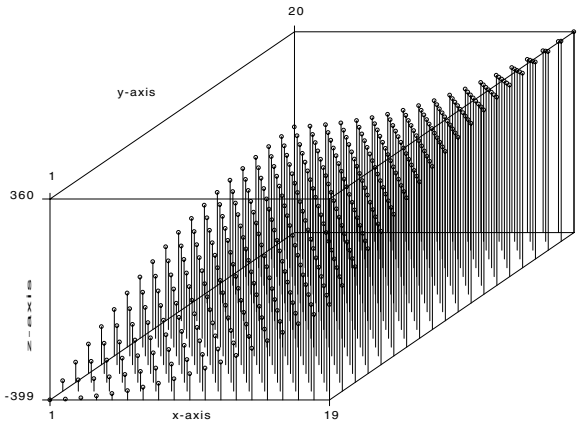

Figure 2: An alternative view of the data in Figure 1.

Rotation

The next plot illustrates the `orient` option, which is an attempt to introduce some sort of rotation ability into the `surface` command. This simple approach allows the user to interchange the axes of the diagram, hence allowing the user to view the plot from a different angle.

```
. surface x y z, orient(zxy)
```

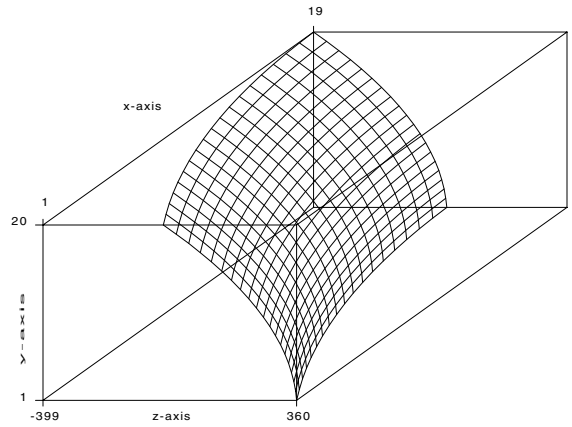

Figure 3: A reorientation of axes.

Nonregular grids and missing data

In order to look at the case where x and y do not form a perfect grid, here is an example surface.

```
. surface x y z
```

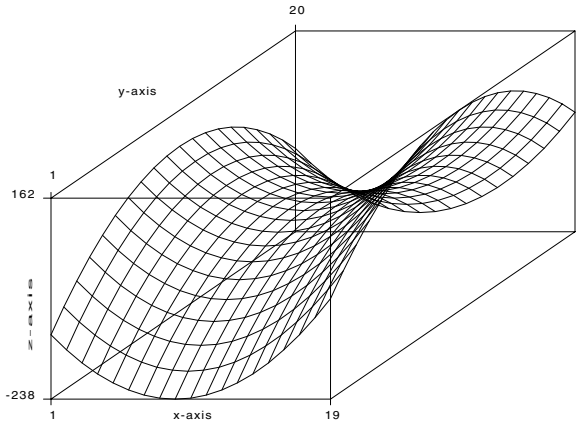

Figure 4: A simple smooth surface.

If we add noise to the x and y values we will not have a regular grid with which to draw the wireframe. The surface command will round the x and y values to reduce the number of different values and then attempt to draw the frame. This is an automatic procedure to attempt to obtain a diagram resembling the underlying surface. The procedure is not guaranteed to produce an adequate grid, for example

```
. surface x y z
WARNING the x-variable contains too many unique values attempting to round
WARNING the y-variable contains too many unique values attempting to round
```

and the resulting graph is empty.

If we use the `round` option, we can obtain a reasonable plot:

(*Graph on next page*)

```
. surface x y z, round(20)
```

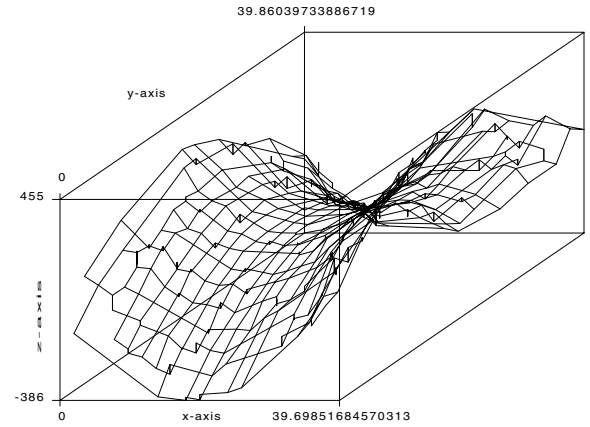

Figure 5: A wireframe plot from irregular x and y.

Missing values are omitted when drawing the wireframe. As can be seen from the diagram below, the lines may not reach the edges of the grid due to missing values. Also, the x and y direction lines may not intersect at a few x and y coordinates where the missing values should have been.

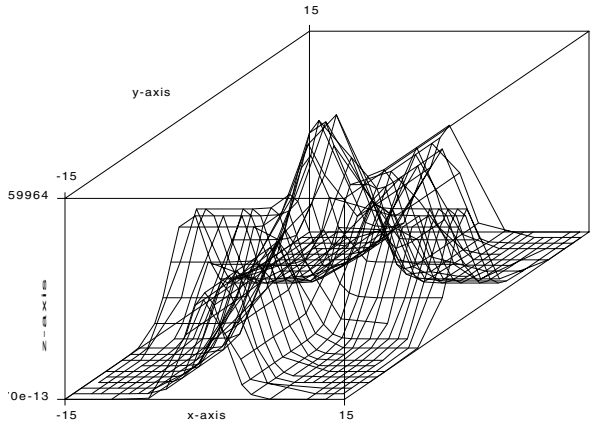

Figure 6: A plot with missing values.

References

van Melle, G. D. 1998. gr30: A set of 3D-programs. *Stata Technical Bulletin* 45: 7–13. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 41–50.

gr40	A simple contour plot

Adrian Mander, Cambridge, UK, adrian.mander@mrc-bsu.cam.ac.uk

This insert describes the command `contour` that plots a 2D contour plot of several variables. The variables must be called *x1*, *x2*, *x3*, and so on. Missing data values are ignored. This specific structure of variable names allows the program to see that data as a matrix. Explicit use of matrices was not implemented as the size of matrices may be too small.

The diagram uses straight lines to join variable entries of the same value. As the program is limited in nature, there is no adjustment of lines between cells of differing values. Instead, the line is equidistant to the center of each cell. No smoothing is possible of the lines as the algorithm is quite simple and constructs the contours by drawing horizontal lines and then vertical lines and does not draw around a shape.

Syntax

`contour` *x1 x2 ... xk* [`if` *exp*] [`, saving(`*filename*`) ticks(#) ltitle(`*string*`) btitle(`*string*`) title(`*string*`)

`contour(#,#, ... ,#) pen(#,#, ... ,#) text legend box(#,#,#,#)`]

draws a contour plot of the variables *x1 x2 ... xk*. If the axis labels are cluttered use the `ticks` option. For clarity of individual contours, use different colors by using the `pen` option.

Options

`saving(`*filename*`)` will save the resulting graph in the specified file name. If the file already exists it is deleted and the new graph will be saved.

`b|l title` are titles for the x, y, and top portions of the graph. The default is the variable name in brackets followed by `follow up` or `timein` depending on the axes.

`ticks()` specifies when a tick is drawn. The integer represents the gap between successive ticks. `ticks(5)` means that ticks are drawn at 5, 10, and so on. The ticks represent a band corresponding to either the row or column.

`legend` specifies that colored numbers in the bottom right corner of the plot represent the corresponding values of the contour lines. Obviously, if all the lines are the same color then the legend numbers will all be the same color.

`text` is an option that allows the values of each cell to be displayed. For small matrices of values this is quite clear, but for larger dimensions the text will obscure the lines and hence reduce clarity.

`contour()` allows the user to specify at what values the contours should be drawn. The default is that the median and quartiles of the data are used giving three lines. There is no restriction on the number of lines that can be drawn. Increasing the number will slow the program down and give a very "busy" diagram.

`pen()` allows the user to specify separate colors for each line specified in the `contour()` option, for example, `con(1,2) pen(3,4)`, specifies that the line at 1 will be drawn in color pen 3 and the contour at 2 will be drawn by the color of pen 4. These pen values correspond to Stata's `gph pen` command.

`box()` specifies bounding box parameters, for example, `box(2500,20000,2500,20000)`. The first number is top y coordinate; second is bottom y coordinate; third is left x coordinate; and fourth is right x coordinate.

Examples

The first example is the simplest; merely going with the built-in options. So the three contours representing the quartiles are plotted.

```
. contour x*
```

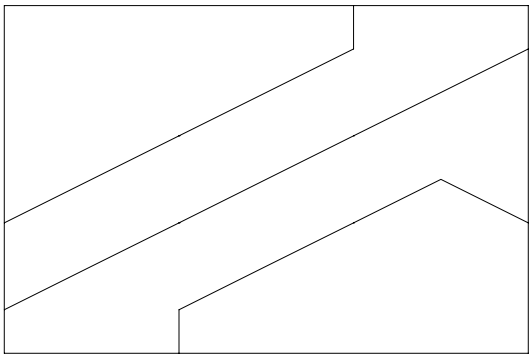

Figure 1. The default behavior of `contour`.

The next example is on a slightly larger dataset and has most of the options activated. The legend is displayed in the bottom right corner. The actual numbers in the variables are plotted on the diagram clearly showing how the contours are fitted. The assumption of plotting the line in between lines is fairly bad for small datasets but will make no overall difference in larger plots.

(*Continued on next page*)

```
. contour x*, legend title(Contour Plot) ltitle(Y) btitle(X) ticks(1) text con(1,2,3,4) pen(3,4,5,6)
```

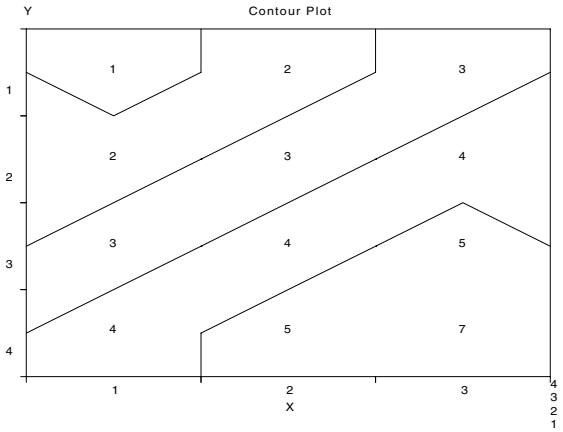

Figure 2. Using the `contour` options.

The last plot is for a larger dataset. The numbers are generated from a cubic function but have had some random noise added, This makes a more detailed diagram.

```
. contour x*, legend title(Contour Plot) ltitle(Y) btitle(X) ticks(2)
```

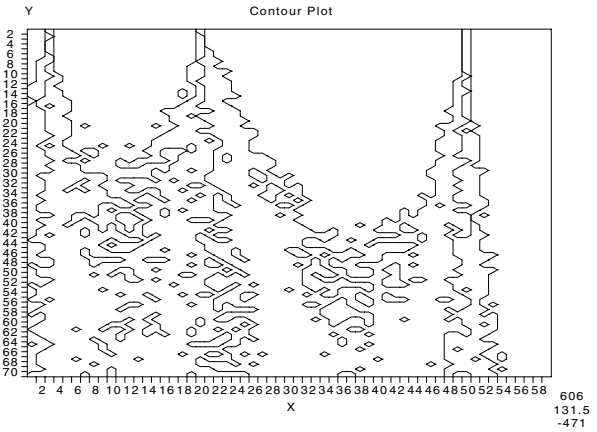

Figure 3. A more complex example.

gr41	Distribution function plots

Nicholas J. Cox, University of Durham, UK, n.j.cox@durham.ac.uk

Syntax

`distplot` *varname* [*weight*] [`if` *exp*] [`in` *range*] [`,` `surv` *graph_options* `by(`*byvar*`)` `freq` `generate(`*newvar*`)`

`mono` `missing`]

`distplot` *varlist* [*weight*] [`if` *exp*] [`in` *range*] [`,` `surv` *graph_options* `freq` `mono`]

`fweights` and `aweights` are allowed.

Description

`distplot` produces a plot of the (empirical) cumulative distribution function(s) for the variables in *varlist*. This shows the proportion (or if desired the frequency) of values less than or equal to each value.

With the `surv` option, `distplot` produces a plot of the (empirical) survival (a.k.a. survivor, reliability, complementary or reverse distribution) function for each *varname*. This shows the proportion (or if desired the frequency) of values greater than each value, that is, the complement of the cumulative distribution function.

Options

`surv` specifies calculation and graphing of the survival function rather than the distribution function.

graph_options are options allowed with `graph, twoway`.

`by(`*byvar*`)` specifies that calculations are to be carried out separately for each class defined by *byvar*. Any graph will, however, show the functions for all classes. For a graph with separate panels for each class, use the `generate( )` option and then `graph` *newvar varname*, `by(`*byvar*`)`. `by( )` is only allowed with a single *varname*.

`freq` specifies calculation of frequency rather than probability.

`generate(`*newvar*`)` specifies a new variable in which the function will be stored. `generate( )` is only allowed with a single *varname*.

`mono` specifies a monochrome treatment, with a single `pen` color, `connect` style and point `symbol`.

`missing`, used only with `by( )`, permits the use of nonmissing values of *varname* corresponding to missing values for the variable named by `by( )`. The default is to ignore such values.

Note that with `by( )` each function is treated graphically as if it were a separate variable, so long as the number of groups is not greater than the limit in Stata on the number of y variables on a scatter plot (20 in Stata 6.0).

With more groups, all functions must be treated graphically as a single variable, by using the `mono` option, which enforces a monochrome treatment. The only `connect` line style appropriate is then `c(L)`, and only one `pen` and one point `symbol` may be used.

If `ylog` is specified, zero values of the survival function are automatically suppressed.

Remarks

For notation let $\Pr(\textit{event})$ denote the probability of *event* and $\#(\textit{event})$ denote the number of occurrences of *event*.

A plot of the cumulative distribution function of a variable X, namely a plot of $F(x) = \Pr(X \leq x)$ versus x, is often a very useful display. The same information is conveyed by the survival (a.k.a. survivor, reliability, complementary or reverse distribution) function $1 - F(x) = \Pr(X > x)$. In many biomedical or engineering applications, the survival function appears closer to the practical problem. The name 'survival function' is suitably evocative if data are indeed times to patient death, component failure, or something similar, and just a name people use otherwise.

In practice, for n values $x_1, \ldots, x_n$, $F(x)$ is estimated by $\#(X < x_i)/n$, which varies from almost 0 to 1. The estimate of $1 - F(x)$ thus varies from almost 1 to 0.

Such plots have several key advantages (see also D'Agostino, 1986):

- the plot indicates the general level, spread and shape of the distribution, and any peculiarities such as outliers or gaps,
- the idea works quite well not only over a range of sample sizes but also for several different groups or variables plotted on the same graph,
- there is no need to choose arbitrary bin widths or origins, as there is with histograms.

According to Hald (1990, 108), the first graph of (the complement of) a distribution function appears in a 1669 letter from Christiaan Huygens (1629–1695) to his brother Lodewijk (1631–1699). He plotted a survival function from data from the life table of John Graunt (1620–1674). Huygens made numerous contributions to mathematics, astronomy and physics, studying, among many other matters, games of chance, the collision of elastic bodies, the rings of Saturn, the pendulum clock and the wave theory of light.

Comparison with cumul

The existing Stata command `cumul` (see [R] **cumul**) is available to calculate the cumulative distribution function for a single variable, after which the function may be plotted using `graph`. The user with several variables to be compared or with an interest in survival functions needs to repeat the `cumul` command or to take the further step of calculating survival functions from cumulative distribution functions. It would seem convenient, therefore, to bundle these calculation and graphing steps into a single program.

With the auto data read in, instead of

```
. cumul mpg, gen(cumpg)
. gra cumpg mpg
```

at its simplest, you can say

```
. distplot mpg
```

and, if you want to keep the cumulative,

```
. distplot mpg, g(cumpg)
```

`cumul` supports separate calculations for each class of some classifying variable, so you could say

```
. cumul mpg, gen(cumpg) by(foreign)
```

but for a useful graph you would need to separate out the different classes. In Stata 6.0, this is easy with `separate` (see [R] **separate**):

```
. separate cumpg, by(foreign)
. gra `r(varlist)´ mpg
```

but it is even easier with

```
. distplot mpg, by(foreign)
```

Comparison with quantile

Note that Stata does have the `quantile` command (see [R] **diagplots**) which produces, for sample data ordered such that $x_{(1)} \leq \ldots \leq x_{(n)}$ a plot of $x_{(i)}$ versus the so-called plotting position $(i - 0.5)/n$, which provides, in essence, an estimate of the proportion of data less than or equal to $x_{(i)}$. This conveys the same information as the cumulative distribution function plot, but with the axes reversed. However, `quantile` is limited to single variables, does not support `by( )` and is constrained in other ways. An accompanying insert describes `quantil2`, which is a generalization of `quantile`.

Comparison with sts graph

Stata already has a graph command for survival functions, `sts graph`. If your data really are survival times and you have any of the complications that are the stuff of survival analysis, such as censoring or subjects entering at different times, then you should use `sts graph`. See [R] **st sts graph**. However, if you have data that are not survival times and you want to plot a survival function, then you could pretend to Stata that your data are survival data, and you would get a graph quickly.

With the auto data, type

```
. stset mpg
. sts
```

and you get a graph of the survival function that is quite similar to that produced by `distplot` with various option choices. By default, however, the `t1title` refers to Kaplan–Meier survival estimates and the `b2title` refers to analysis time. More generally, the graphs produced by `sts graph` seem geared to what biostatisticians (and perhaps engineering statisticians) typically do, with some design choices welded into the program. In particular, there seems to be a consensus that showing survival functions as step functions (in Stata terms, using the connect style `c(J)`) is the only proper way. In other circumstances, or for other tastes, another program may be of use. In territory familiar to me, for example, "what is the probability of a higher rainfall?" is a natural question, but very few talk about Kaplan–Meier estimates, and so the default `t1title` is just confusing. And it is easy to get thousands of values, so the `connect( )` choices `c(l)`, `c(s)`, and `c(J)` have essentially the same effect in practice.

Note also that Stata will not accept variables with zero or negative values as survival time data. In contrast, `distplot` will happily plot survival functions for such variables. Examples would be residuals, which will be both negative and positive.

Note on ylog option in distplot

Often survival functions are inspected for tendencies to either exponential or power-law behavior. In either case, examining the survival function on a logarithmic scale is clearly a natural step. A small but notable detail is that whenever the `ylog` option is invoked, `distplot` automatically suppresses from the graph (but not from any generated variable) any zero values. This is not true of `sts graph`.

This suppression can be justified on two grounds:

- On any survival function, zero occurs only for the maximum value of the data variable.
- Zeros arise because we are calculating $\Pr(X > x)$ or $\#(X > x)$. This is a conventional choice for the definition, which meshes nicely with the convention $\Pr(X \leq x)$ or $\#(X \leq x)$ for the cumulative. However, some statisticians use

$\Pr(X \geq x)$ or $\#(X \geq x)$, with which zeros would not be observed. (For theoretical results with continuous variables, it makes essentially no difference what you use, and if this difference made a practical difference to data analysis, then your data are inadequate.)

On the other hand, in `distplot` the `xlog` option is not implemented in the same way. If zero is a reasonable value for your variable, then presumably it could occur more than once, and it would be a distortion to omit those values from your graph. Hence with zero or indeed negative values, `xlog` would fail, as with `graph`. Naturally, you can always explicitly exclude nonpositive values using `if` *myvar* `> 0` (or `if` *myvar*).

Comparison with cdf

`distplot` is complementary to `cdf` as published by Clayton and Hills (1999). Unlike `distplot`, `cdf` supports `iweights` and `pweights` and the plotting of normal (Gaussian) distribution functions as reference curves. On the other hand, unlike `cdf`, `distplot` supports plotting of functions for several variables, plotting of survival functions, and plotting of frequency curves as well as probability curves.

Examples

The command

```
. distplot mpg , surv yla(0(0.1)1) xla(10(5)40) gap(3)
```

gives

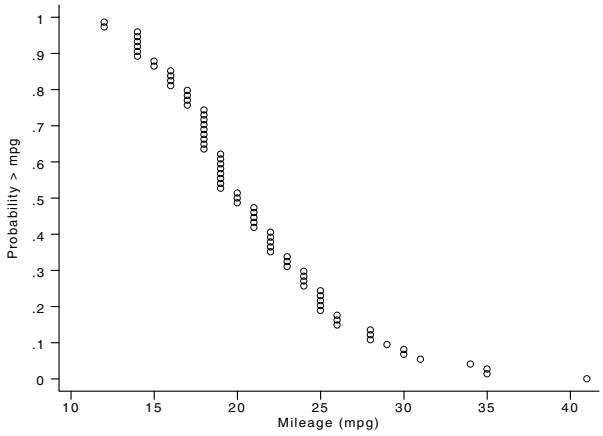

Figure 1. The survival function for miles per gallon.

while

```
. distplot mpg, by(foreign) yla(0(0.1)1) xla(10(5)40) c(ll) gap(3)
```

gives

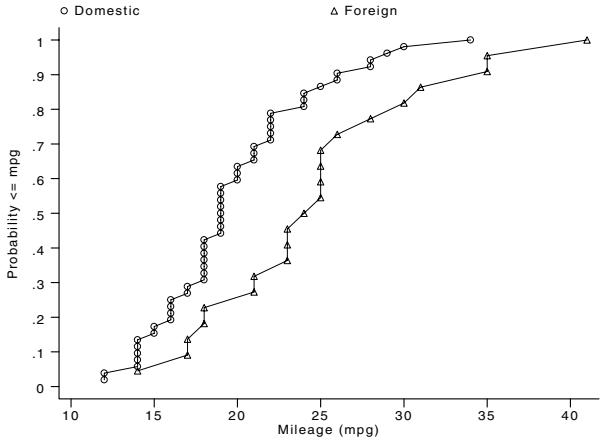

Figure 2. Distribution function for miles per gallon by domestic and foreign.

Acknowledgments

Elizabeth Allred made helpful comments during program development.

References

Clayton, D. and M. Hills. 1999. Cumulative distribution function plots. *Stata Technical Bulletin* 49: 10–12.

D'Agostino, R. B. 1986. Graphical analysis. In *Goodness-of-fit techniques*, ed. R. B. D'Agostino and M. A. Stephens. 7–62. New York: Marcel Dekker.

Hald, A. 1990. *A History of Probability and Statistics and Their Applications Before 1750.* New York: John Wiley & Sons.

gr42	Quantile plots, generalized

Nicholas J. Cox, University of Durham, UK, n.j.cox@durham.ac.uk

Syntax

`quantil2` *varname* [`if` *exp*] [`in` *range*] [`, reverse` *graph_options* `a(#) by(`*byvar*`) missing`]

`quantil2` *varlist* [`if` *exp*] [`in` *range*] [`, reverse` *graph_options* `a(#)`]

Description

`quantil2` produces a plot of the ordered values of *varlist* against the so-called plotting positions, which are essentially quantiles of a uniform distribution on $[0, 1]$ for the same number of values.

For n values of a variable x ordered so that $x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n-1)} \leq x_{(n)}$ and a given constant a, the plotting positions are $(i - a)/(n - 2a + 1)$ for $i = 1, \cdots, n$.

For more than one variable in *varlist*, only observations with all values of *varlist* present are shown.

Options

`reverse` reverses the sort order, so that values decrease from top left. Ordered values are plotted against $1-$ plotting position.

graph_options are options allowed with `graph, twoway`.

`a(#)` specifies a in the formula for plotting position. The default is $a = 0.5$, giving $(i - 0.5)/n$. Other choices include $a = 0$, giving $i/(n + 1)$, and $a = 1/3$, giving $(i - 1/3)/(n + 1/3)$.

`by(`*byvar*`)` specifies that calculations are to be carried out separately for each class defined by a single variable *byvar*. Any graph will, however, show the functions for all classes. `by( )` is only allowed with a single *varname*.

`missing`, used only with `by( )`, permits the use of nonmissing values of *varname* corresponding to missing values for the variable named by `by( )`. The default is to ignore such values.

Note that with `by( )` each function is treated graphically as if it were a separate variable, so long as the number of groups is not greater than the limit in Stata on the number of y variables on a scatter plot (20 in Stata 6.0).

Remarks

Plotting quantiles (vertical axis) against plotting position (horizontal axis) in quantile plots can be seen as a variation on the longer-established plotting of distribution functions, in which cumulative probability (vertical axis) is plotted against the magnitude of a variable (horizontal axis). An accompanying insert describes `distplot`, a program for plotting distribution functions and their complements. The term quantile plot appears in Chambers et al. (1983) and Cleveland (1993, 1994). Modern use of quantile plots and their relatives stems largely from the path-breaking paper of Wilk and Gnanadesikan (1968). Examples of antecedents from the nineteenth century can be found in Quetelet (1827) and Galton (1875); see Stigler (1986, 167, 270).

The official Stata command `quantile` produces quantile plots for single variables. See [R] **diagplots**.

`quantil2` generalizes the `quantile` command in five ways:

1. One or more variables may be plotted.
2. The sort order may be reversed, so that values decrease from top left.
3. A single variable may be classified by another single variable, specified by the `by( )` option.

4. There is support for graphical choices other than the set `s(o) c(.) xsca(0,1) xlab(0,.25,.5,.75,1)` wired into `quantile`.

5. The plotting position is in general $(i-a)/(n-2a+1)$, in contrast to the more specific $(i-0.5)/n$ wired into `quantile`. This is a minor point graphically, but may be useful to some users.

The plotting position is, in essence, an estimate of the proportion of data less than or equal to $x_{(i)}$. Popular choices for a are 0.5, suggested by Hazen, and 0, suggested by Weibull and Gumbel, and wired into the official Stata commands `pnorm`, `qnorm`, `pchi` and `qchi`.

For many years there has been debate about the relative merits of these plotting position formulae: see Barnett (1975), Cunnane (1978), Harter (1984) and Hyndman and Fan (1996). It is agreed that the ideal plotting positions depend on the distribution being fitted, and also on the precise purpose of plotting, whether model validation or parameter estimation. Cunnane (1978) focuses on probability plotting as estimation of quantiles, ideally with no bias and minimum variance. This implies, for example, that the Weibull or Gumbel formula with $a = 0$ is correct for the uniform distribution alone, while $a = 0.375$ should be used for the Gaussian or normal distribution, and $a = 0.44$ for the exponential and Gumbel (extreme value I) distributions. The latter values of a are closer to the Hazen proposal of 0.5 than to the Weibull or Gumbel proposal of 0. Many authors, including Chambers et al. (1983) and Meeker and Escobar (1998), use $a = 0.5$ as a general rule.

From a slightly different perspective, it is worth noting that using $a = 1/3$ yields the median of the sampling distribution of the estimated quantile, to a good approximation for any continuous distribution (Hoaglin 1983).

`quantil2` uses the `egen` function `pp( )` to calculate plotting positions, previously published by Cox (1999).

One feature of `quantile` cannot be emulated with `quantil2`. `quantile` draws a straight line connecting bottom left and top right data points. `quantil2` does not draw such lines. There are two reasons for this choice. First, they would complicate the graph. Second, the rationale for drawing a straight line in `quantile` is presumably to aid comparison with a uniform distribution with the same range. That kind of comparison appears uncommon in data analysis, and in any case with multiple datasets there is likely to be far more interest in comparing datasets directly with each other.

Examples

The command

```
. quantil2 mpg
```

gives

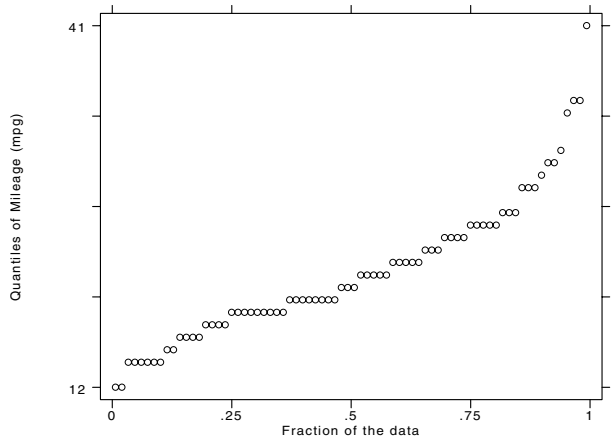

Figure 1. Quantile plot of miles per gallon.

while

```
. quantil2 mpg, by(foreign)
```

gives

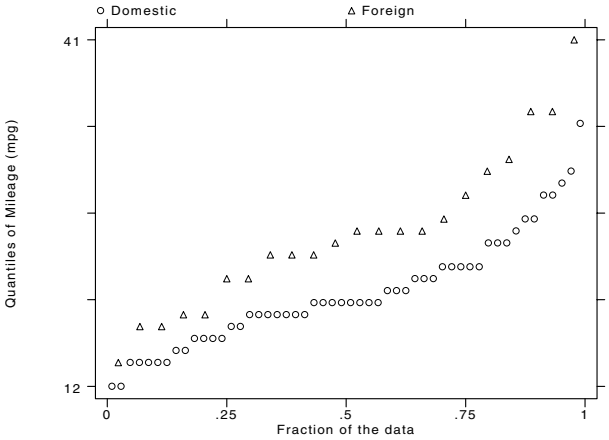

Figure 2. Miles per gallon for domestic and foreign cars.

and

```
. quantil2 mpg trunk hdroom, rescale
```

gives

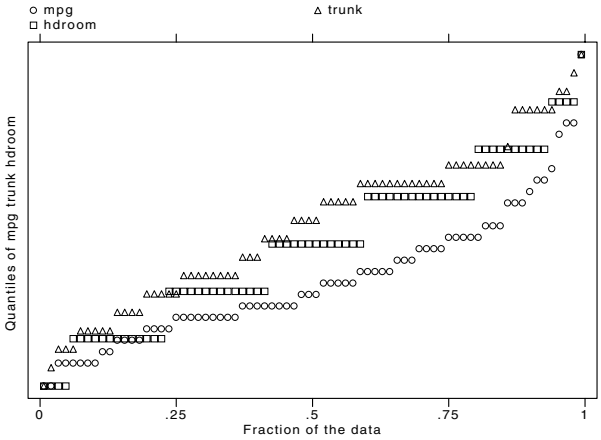

Figure 3. Quantile plots of three variables.

References

Barnett, V. 1975. Probability plotting methods and order statistics. *Applied Statistics* 24: 95–108.

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

Cleveland, W. S. 1993. *Visualizing Data.* Summit, New Jersey: Hobart Press.

——. 1994. *The Elements of Graphing Data.* Summit, New Jersey: Hobart Press.

Cox, N. J. 1999. Extensions to `generate`, extended. *Stata Technical Bulletin* 50: 9–17.

Cunnane, C. 1978. Unbiased plotting positions–a review. *Journal of Hydrology* 37: 205–22.

Galton, F. 1875. Statistics by intercomparison, with remarks on the law of frequency of error. *Philosophical Magazine* 4th series, 49: 33–46.

Harter, H. L. 1984. Another look at plotting positions. *Communications in Statistics, Theory and Methods* 13: 1613–33.

Hoaglin, D. C. 1983. Letter values: a set of selected order statistics. In *Understanding Robust and Exploratory Data Analysis.* ed. D. C. Hoaglin, F. Mosteller, and J. W. Tukey. 33–57. New York: John Wiley & Sons.

Hyndman, R. J. and Y. Fan. 1996. Sample quantiles in statistical packages. *American Statistician* 50: 361–365.

Meeker, W. Q. and L. A. Escobar. 1998. *Statistical Methods for Reliability Data.* New York: John Wiley & Sons.

Quetelet, A. 1827. Recherches sur la population, les naissances, les décès, les prisons, les dépôts de mendicité, etc., dans le royaume des Pays-Bas. *Nouveaux mémoires de l'Académie royale des sciences et belles-lettres de Bruxelles* 4: 117-192.

Stigler, S. M. 1986. *The History of Statistics: the Measurement of Uncertainty Before 1900.* Cambridge, MA: Harvard University Press.

Wilk, M. B. and R. Gnanadesikan. 1968. Probability plotting methods for the analysis of data. *Biometrika* 55: 1–17.

os15	Command-name registration at www.stata.com

William Gould, Stata Corporation, wgould@stata.com
Alan Riley, Stata Corporation, ariley@stata.com

Very active Stata programmers, and especially those who exchange ado-files with others, worry about choosing unique names for their commands. Below is presented a new Stata command to help alleviate that concern. The new command, `cmdname`, communicates with a registry we have installed at www.stata.com. Using `cmdname`, you can check whether a command name is available, and you can register the name to yourself if it is available.

Below is presented a manual entry for `cmdname` written in a style as if all decisions concerning this registry have been made and are final. This is, however, work in progress and nothing is final. If `cmdname` turns out to be useful, we intend to make it an official part of Stata. Until that time, we invite comments.

The last section of this insert deals with programming commands that communicate in sophisticated ways with web servers.

Syntax

`cmdname` *newcmd* [`, register name(`*string*`) email(`*string*`)`]

Note: You may set the global macros `S_NAME` and `S_EMAIL` to avoid specifying the `name()` and `email()` options.

Description

`cmdname` without the `register` option contacts http://www.stata.com to find out whether you could register the new command name *newcmd*.

`cmdname` with the `register` option contacts http://www.stata.com and registers *newcmd* in your name if that is possible.

Options

`register` specifies that you wish to register *newcmd* in your name. If `register` is not specified, a report is returned as to whether you could register *newcmd*.

`name()` and `email()` must be specified if `register` is specified. `name()` specifies your name and `email()` your email address. For example, you might type

```
. cmdname mynewcmd, register name(Bob Smith) email(bsmith@uni.edu)
```

If you set the global macros `S_NAME` and `S_EMAIL` to contain your name and email address, you need not specify options `name()` and `email()`:

```
. global S_NAME "Bob Smith"
. global S_EMAIL "bsmith@uni.edu"
```

Many users set `S_NAME` and `S_EMAIL` in their profile.do file.

Remarks

Very active Stata programmers, and especially those who exchange ado-files with others, worry about choosing unique names for their commands. `cmdname` helps to alleviate that concern.

A command-name registry maintained at www.stata.com records the names programmers have chosen for new commands. You can check whether a name is available and you can register names you have chosen.

The registry represents a loose agreement among Stata programmers to try to name new commands uniquely.

The concern cmdname alleviates

Say you write a new command called `reghpb` and somebody else on the other side of the world also writes a new command of that name. Mostly when that occurs it does not matter because the two new commands never meet on the same computer. If both programmers make their commands available to others, however, someone could want both. If both commands have the same name, that will not be possible.

To avoid the problem, when you write `reghpb`, you could find out whether that name is available by typing

```
. cmdname reghpb
```

If it were available, you could make it yours by typing

```
. cmdname reghpb, register
```

Now if someone else comes up with the bright idea of writing a program named `reghpb`, he or she should type '`cmdname reghpb`' and so discover that you have already registered that name.

Registration rules enforced by cmdname

1. You may not register a name already registered.
2. You may not register a name that appears in the English-language dictionary.
3. You may not register a name that is less than 4 characters long.

When you type '`cmdname` *name*`, register`', `cmdname` checks that the name is not already registered, that the name is not an English word, and that the name is long enough.

In addition, there is a fourth rule that is only loosely enforced by `cmdname`:

4. You may not register a name that is a word of statistical jargon that is in common use.

Be careful because, in many cases `cmdname` will allow you to register such names. Periodically, newly registered names are reviewed by human beings. If you have registered a name that is deemed to be common statistical jargon, your registration will be revoked and you will be sent nice email alerting you to this. (That does not mean you have to change your program; it only means that the name is not registered to you.)

Reason for the registration rules

The thinking is that really nice names should be reserved for really nice commands that are used by a large fraction of the Stata community.

That thinking is then coupled with the argument that any command that fits the above definition is or will become part of Stata as it is distributed by StataCorp.

Programmers other than StataCorp programmers can write really nice commands, but such commands get really nice names only after being published in the STB or otherwise distributed under not-so-nice names and then being adopted for inclusion into Stata, at which point they will be renamed. At that point, such programmers will also get a really nice reference in the manual.

Concerning names shorter than 4 characters, there is nothing wrong with giving your programs such names and the author of `cmdname` admits that he, himself, has numerous such programs, all unregistered. StataCorp would like to keep short names reserved for private use and, on rare occasion, for use for really common commands (think of `d` for `describe`). That is to say, programs with short names should not be traded.

Administration of the command registry

The command registry is administered by StataCorp. Questions should be directed to registry@stata.com.

Registration is not a guarantee

Nobody forces other programmers to use `cmdname`, so even if you use `cmdname`, somebody else could still use "your" name.

Also understand that, even if you register your command's name, that registration might later be revoked. Such action is not undertaken lightly, but it does happen. We apologize but there is no appeal. On the bright side, at least you will receive email alerting you of this. And remember, there is nothing to stop you from continuing to call your program by an unregistered or revoked name. If the name is revoked, you have merely been reminded that someday a command of the same name might appear in Stata.

This is all a very loose arrangement. Nevertheless, if you do register your program name, you can be reasonably assured that you will be able to keep the name. For instance, StataCorp will not publish an insert in the STB with the same name as your program. StataCorp checks the registry and, if an STB insert comes in with the same name, they inform the author that the program's name must be changed.

Examples

cmdname without the register option merely checks whether a name is available. For instance, you are thinking about writing a program named super:

```
. cmdname super
(contacting http://www.stata.com)
super is a word in the dictionary and not available for registration
```

You are thinking about writing a program named kapwgt:

```
. cmdname kapwgt
(contacting http://www.stata.com)
kapwgt already registered to Stata Corp., registry@stata.com
```

kapwgt is already a command of Stata.

You are thinking about writing a program named outreg:

```
. cmdname outreg
(contacting http://www.stata.com)
outreg already registered to STB, stb@stata.com
```

A command named outreg has already been published in the STB. (In setting up the registry, we went back and registered all previously published commands. In the future, STB published commands will likely be registered to the author, not the STB itself.)

You are thinking about writing a program named vi:

```
. cmdname vi
you may not register command names shorter than 4 characters
r(198);
```

You are thinking about writing a program named regdad:

```
. cmdname regdad
(contacting http://www.stata.com)
regdad is available for registration
```

You could register that name. (The authors request, however, that you do not register regdad because if you do, he will have to rewrite this section of the on-line help file.)

Examples of registration

You register names by specifying the register option. You do not need to check that the name is available first:

```
. cmdname kapwgt, register
(contacting http://www.stata.com)
kapwgt already registered to Stata Corp., registry@stata.com
r(110);
```

If kapwgt had not already been registered, you would have seen

```
. cmdname kapwgt, register
kapwgt now registered to you (Your Name, you@email.adr)
```

and, from then on, if anyone else tried to register kapwgt, they would be told that "kapwgt already registered to *Your Name*, *you@email.adr*".

Now actually what would happen if you tried to register kapwgt would be

```
. cmdname kapwgt, register
must specify options -name()- and -email()- or you must set the
global macros S_NAME and S_EMAIL to contain your name and email
address.
r(198);
```

Either you would have to type

```
. cmdname kapwgt, register name(Your Name) email(you@email.adr)
```

or you could type

```
. global S_NAME "Your Name"
. global S_EMAIL "you@email.adr"
. cmdname kapwgt, register
```

Many users put the two definitions of S_NAME and S_EMAIL in their profile.do file.

An example you can try

We do not want you registering real names unless you intend to use them, but we know you want to try registering a name, so we have created a convention for phony names you can register.

cmdname allows phony names of the form "1x...", names beginning with the character one followed by lowercase x. Names starting with the characters 1x are obviously not valid command names, but cmdname will let you register them anyway. These names really will be registered, but when we review the registry, we will remove names starting with 1x.

Try this:

```
. cmdname 1xthis, register
(contacting http://www.stata.com)
1xthis now registered to you (Your Name, you@email.adr)
```

Probably you will not be successful in registering 1xthis because someone else has already registered it. Try another name starting with "1x".

If you make a mistake when registering your command name and want to unregister it, please email registry@stata.com and let us know what name you want to unregister.

Programming considerations

We have nothing more to say about using cmdname. The rest of this insert is intended for readers interested in programming other commands that communicate with web servers, for which we will use cmdname as an example.

There are two components to cmdname:

1. cmdname.ado, the part that resides on the local computer;
2. http://www.stata.com/commands/register.cgi, the CGI script installed on our server with which cmdname communicates.

register.cgi does most of the work. It is register.cgi that maintains the registry and is able to answer questions such as "is such-and-such already registered?" and perform tasks such as "register such-and-such if it is not already registered".

cmdname.ado simply formulates the requests, passes them along to the CGI script, and then presents whatever output the CGI script sends back.

Communication between cmdname.ado and register.cgi is handled by one Stata command—copy. Inside cmdname.ado is

```
tempfile file
copy `"http://www.stata.com/commands/register.cgi?version=1&
       subcmd=`subcmd´&
       word=`sword´&
       name=`sname´&
       email=`smail´"´   `"`file´"´, text
```

except that the line in the code is all run together on one physical line. We have introduced spaces and alignment here simply to improve readability.

Understand, Stata has a copy command that can copy files. If you have the file mytext.txt and type

```
. copy mytext.txt mytext2.txt
```

you will now have a copy of mytext.txt in the file mytext2.txt. copy can work over the Internet. If you type

```
. copy http://www.stata.com/index.html shome.txt
```

you will have a copy on your computer of Stata's home page in the file shome.txt. Unless you are using a Unix system, however, it would be a little better if you were to type

```
. copy http://www.stata.com/index.html shome.txt, text
```

because Stata Corporation has a Unix server and it will send the file to you with Unix-style line-end characters. copy's text option tells copy to expect a text file and to change the line-end characters to what is appropriate for your computer.

Whatever the server is willing to deliver, Stata can copy. So we installed a CGI script on our server and coded

```
tempfile file
```

```
copy `"http://www.stata.com/commands/register.cgi?version=1&
       subcmd=`subcmd´&
       word=`sword´&
       name=`sname´&
       email=`smail´"´   `"`file´"´, text
```

The result of this is to send to `register.cgi` the ampersand-separated information and to retrieve into a temporary file any output `register.cgi` produced. Stata sends the ampersand-separated information as part of the HTTP request it makes to the server, and the server passes it to `register.cgi` as if it were posted from a form using the GET method.

We wrote `register.cgi` to expect five arguments:

	`version=1`	This must be the first argument
	`subcmd=query`	Check whether *word* is already registered
or	`subcmd=register`	or register *word* if not already registered
	`word=`*word*	word to be checked or registered
	`name=`*text*	name of person making request
	`email=`*text*	email address of person making request

and we wrote `register.cgi` to create standardized output: the first line of the output would contain a numeric "result code" and the second and subsequent lines would contain additional detail, information, or messages. For instance, on a `query` request, `register.cgi` might return on line 1

0	word available for registration
1	word already registered, `line[2]` = text message saying same thing as numeric code `line[3]` = who previously registered word `line[4]` = email address of person who previously registered
2	word is in English-language dictionary
5	word is invalid (such as being null, too long, etc.)
96	no action taken; incompatible version
97	no action taken; `register.cgi` temporarily unavailable

Thus, the next step in `cmdname.ado` was to read back what `register.cgi` had to say, making the entire process

```
tempfile file
copy `"http://www.stata.com/commands/register.cgi?version=1&
       subcmd=`subcmd´&
       word=`sword´&
       name=`sname´&
       email=`smail´"´   `"`file´"´, text
infix str line 1-80 using `"`file´"´
```

There is, however, a complication that we gloss over that is of great importance when copying output from `CGI` scripts. The actual code inside Stata reads

```
tempfile file
set checksum off                /* <- turn off checksum      */
capture {
        copy `"http://www.stata.com/commands/register.cgi?version=1&
               subcmd=`subcmd´&
               word=`sword´&
               name=`sname´&
               email=`smail´"´   `"`file´"´, text
}
local rc = _rc
set checksum on                 /* <- turn checksum back on  */
if `rc´ { exit `rc´ }
infix str line 1-80 using `"`file´"´
```

When Stata copies a file over the Internet, it also looks for the same filename with the extension `.sum`. It does not bother Stata if the file is not found but, if it is, Stata interprets that as a checksum file and uses that to verify that the copy Stata received was not corrupted.

In the case of CGI scripts, Stata's requesting the corresponding `.sum` file can corrupt the input to the script. Stata does not know it is calling a CGI script, so it searches from the end of the "filename" in the `copy` command for an extension to replace with `.sum`. In `cmdname`, this would result in the last part of the email address being replaced with `.sum` So it is necessary to turn off this feature before copying the file and then to turn it back on again afterwards.

In any case, the details of the design of `cmdname` do not really matter. What is important to understand is that Stata can copy files over the web and that those "files" can include the output from CGI scripts installed on the web server. Using this, one can create Stata commands that send instructions to web servers. Reading back the output, one can even create commands that respond appropriately to how the server responds.

sbe30	Improved confidence intervals for odds ratios

John R. Gleason, Syracuse University, loesljrg@accucom.net

Commands in Stata's `epitab` suite (see [R] **epitab**) calculate quantities of interest to epidemiologists, including confidence intervals for odds ratios. This insert presents `oddsrci` and `oddsrcii`, two new commands for confidence intervals about odds ratios. These commands differ from the current `epitab` offerings in several respects. First, they treat odds ratios as an ordinary part of the analysis of 2×2 tables, free of the specialized jargon of epidemiology. Second, `oddsrci` provides a convenient, impromptu means of defining the 2×2 table underlying an odds ratio; `oddsrcii` is its immediate form. Most importantly, the `epitab` commands implement only the Cornfield and Woolf methods of forming the confidence limits. `oddsrci` and `oddsrcii` provide, in addition, the Gart method and the endpoint-adjusted, independence-smoothed confidence interval recently advocated by Agresti (1999). The latter is arguably the method of choice for forming odds ratio confidence intervals.

Overview

We begin with a brief review of the basic ideas underlying the various methods of setting confidence intervals for odds ratios. (We also acknowledge a large debt to Alan Agresti; this review borrows heavily from Agresti, 1999.) Epidemiologists and others familiar with the details of odds ratios may wish to skip this section.

Consider a 2×2 contingency table with observed cell counts $\{n_{ij}\}$ and expected counts $\{\nu_{ij}\}$, $i, j = 1, 2$. The odds ratio is $\theta = (\nu_{11}\nu_{22})/(\nu_{12}\nu_{21})$ and the obvious (and often, maximum likelihood) estimator is $\widehat{\theta} = (n_{11}n_{22})/(n_{12}n_{21})$. The obvious estimator has an obvious problem; what to do about the possibility that $n_{ij} = 0$ in some cell. The problem is compounded in the usual (normal theory) approach to forming a confidence interval for θ. Under multinomial, independent binomial, or Poisson sampling models, $\log(\widehat{\theta})$ is asymptotically normal with standard error estimated by $(\sum n_{ij}^{-1})^{1/2}$. (All summations are over all four cells of the 2×2 table.)

The obvious problem has an obvious solution; add some constant to each cell count n_{ij} before computing $\log \widehat{\theta}$ or its estimated standard error. But what constant would be best? Anticipating future developments, write the (now modified) estimator and its estimated standard error as

$$\log \widehat{\theta} = \log\left(\frac{(n_{11}+c_{11})(n_{22}+c_{22})}{(n_{12}+c_{12})(n_{21}+c_{21})}\right) \quad \text{and} \quad \widehat{\sigma}_{\log\hat{\theta}} = \sqrt{\sum (n_{ij}+c_{ij})^{-1}}$$

where the $\{c_{ij}\}$ are nonnegative constants. An approximate $100(1-\alpha)\%$ confidence interval for $\log\theta$ is then

$$\log \widehat{\theta} \pm z_{\alpha/2}\widehat{\sigma}_{\log\hat{\theta}}$$

where $z_{\alpha/2} = \Phi^{-1}(1-\alpha/2)$ and Φ^{-1} is the $N(0,1)$ quantile function, evaluated in Stata with `invnorm()`. Exponentiating the interval endpoints gives an approximate confidence interval for the odds ratio, θ.

As noted above, setting all $c_{ij} = 0$ makes $\widehat{\theta}$ the unconditional maximum likelihood estimator in most settings; the resulting confidence interval is known as the Woolf (1955) interval, and is computed by the `woolf` option of Stata's `cci` command. Perhaps the most common choice is $c_{ij} = 0.5$, which can be traced to Haldane (1955), Anscombe (1956), and Gart (1966). The `gart` option of `oddsrci` and `oddsrcii` implements this approach.

Agresti (1999) argues for a different choice. Noting that adding $c_{ij} = c$ to each cell count is tantamount to smoothing the observed frequencies toward a model of equiprobability, Agresti points out that a better choice would be to smooth toward a model of independence. In particular, consider the choice $c_{ij} = 2(n_{i+}n_{+j}/n^2)$, where n_{i+} and n_{+j} are row and column total counts, and $n = \sum n_{ij}$. In addition, Agresti argues (and not without reason) that a confidence interval for θ should have a lower limit of 0 if and only if $n_{11}n_{22} = 0$, and an upper limit of $+\infty$ if and only if $n_{12}n_{21} = 0$.

Denote the interval just described as the *endpoint-adjusted, independence-smoothed* confidence interval for the log odds ratio, θ. Agresti then goes on to demonstrate numerically that the performance of this interval is far better than that of other available intervals. In fact, its coverage probability in the independent binomials case is accurate even for samples as small as $n_1 = n_2 = 10$. This interval is implemented by the `agresti` option of the `oddsrci` and `oddsrcii` commands.

Finally, we note that `epitab` commands provide yet another confidence interval, one due to Cornfield (1956). `oddsrci` and `oddsrcii` also offer this option, although the evidence in Agresti (1999) would indicate that it is a poor choice, especially with small samples.

New commands for odds ratios

The command `oddsrci` has syntax

`oddsrci` [*weight*] [`if` *exp*] [`in` *range*] `, c1(`*cond1*`) c2(`*cond2*`)` [`level(`#`)` `tabopt(`*tab_options*`)`

`none all agresti cornf gart woolf`]

`fweights` are allowed.

Options

`c1(`*cond1*`)` provides a Boolean (true–false) expression that defines the rows of a 2×2 table. (`c1` is not optional.)

`c2(`*cond2*`)` provides another Boolean (true–false) expression that defines the columns of a 2×2 table. (`c2` is not optional.)

`level(`#`)` specifies the desired confidence level specified either as a proportion or as a percentage. The default level is the current setting of the system macro `S_level`.

`tabopt(`*tab_options*`)` is used to pass options to the `tabulate` command; only the `nofreq` option is unavailable.

`all, none` choose among the following confidence intervals in the obvious way.

- `agresti` produces the "endpoint-adjusted, independence smoothed" confidence interval recommended by Agresti (1999). (This is the default.)
- `cornf` yields the Cornfield confidence interval, as computed by `cci`.
- `gart` computes the Anscombe–Gart–Haldane confidence interval.
- `woolf` requests the Woolf confidence interval (same as in `cci`).

The arguments of the required options `c1()` and `c2()` are any two Boolean (true–false) conditions whose truth values define the 2×2 table of interest. *cond1* and *cond2* can be arbitrarily complex expressions and they may contain embedded double-quote (`"`) characters; the only requirement is that they evaluate to true or false. `oddsrci` creates two temporary variables with commands resembling '`gen byte` *Var1* `= (`*cond1*`)`' and '`gen byte` *Var2* `= (`*cond2*`)`', and then uses the `tabulate` command to form the required table. The option `tabopt()` can be used to supply options to `tabulate`; the `nofreq` option is unavailable.

The remaining options choose confidence interval methods for the odds ratio. `all` selects each of the four available methods, `none` chooses none of them (useful to see just the output of `tabulate`). `agresti` is the default method.

`oddsrcii` is an immediate command with the following syntax:

`oddsrcii` *a b c d* [`, notable level(`*level*`) tabopt(`*tab_options*`) none all agresti cornf gart woolf`]

The arguments *a*, *b*, *c*, and *d* are the four cell counts of the 2×2 table. The option `notable` suppresses all output except for the odds ratio confidence interval(s). The remaining options are identical to those of `oddsrci`; indeed, `oddsrci` calls `oddsrcii` to display confidence intervals.

Each of the commands also has an alternative syntax that displays a quick reminder of usage:

```
oddsrci ?

oddsrcii ?
```

In each case, this form just uses the `which` command; for example:

```
. oddsrci ?
c:\ado\oddsrci.ado
*!  CIs for the odds ratio defined by two Boolean conditions
*!  Syntax:  . oddsrci [fweight] [if] [in], C1(1st Boolean condition)
*!                C2(2nd Boolean condition)     [ LEVel(0.xx)
*!                TABopt(tab options)   NONE ALL AGresti COrnf GArt WOolf ]
*!  Version 1.0.1 <JRG; 01Aug1999>
```

Example

To illustrate, consider the dataset `cancer.dta` supplied with Stata 6.0:

```
. use cancer, clear
(Patient Survival in Drug Trial)
```

```
. describe
Contains data from cancer.dta
  obs:            48                          Patient Survival in Drug Trial
  vars:            4                          16 Nov 1998 11:49
  size:          576 (99.5% of memory free)
-------------------------------------------------------------------------------
  1. studytim  int    %8.0g                  Months to death or end of exp.
  2. died      int    %8.0g                  1 if patient died
  3. drug      int    %8.0g                  Drug type (1=placebo)
  4. age       int    %8.0g                  Patient´s age at start of exp.
-------------------------------------------------------------------------------
Sorted by:
```

Suppose we are interested in a possible relationship between whether a patient was older than 60 and whether that patient survived for more than a year, considering only patients who received one of the two active drugs:

```
. oddsrci if drug > 1, c1(studytim > 12) c2(age > 60) all
  Select cases: if drug > 1
  Cond. 1: studytim > 12
  Cond. 2: age > 60
           |        Cond. 2
  Cond. 1 |     False       True |     Total
----------+----------------------+----------
    False |         5          3 |         8
     True |        17          3 |        20
----------+----------------------+----------
    Total |        22          6 |        28
  Cornfield:  Odds Ratio = .29412,  [95% CI]:  .04907  1.7131
    Agresti:  Odds Ratio = .31836,  [95% CI]:  .04844  2.0926
       Gart:  Odds Ratio = .31429,  [95% CI]:  .05383  1.8348
      Woolf:  Odds Ratio = .29412,  [95% CI]:  .04463  1.9382
```

None of the confidence intervals finds evidence of such a relationship. Given knowledge of the cell frequencies, one might instead do something along these lines:

```
. oddsrcii 5 3 17 3, tab(V lr) lev(.975)
           |          col
       row |         1          2 |     Total
-----------+----------------------+----------
         1 |         5          3 |         8
         2 |        17          3 |        20
   --------+----------------------+----------
     Total |        22          6 |        28
  likelihood-ratio chi2(1) =   1.6031   Pr = 0.205
               Cramer´s V =  -0.2477
  Agresti:  Odds Ratio = .31836,  [97.5% CI]:  .03696  2.7422
```

Saved Results

`oddsrcii` saves in `r()`, whether or not called from `oddsrci`. Results saved by the (embedded) call to `tabulate` will also be preserved. Thus, following the last example above:

```
. return list
  scalars:
       r(level)    =  97.5
       r(or_Agres) =  .3183624801271863
       r(lb_Agres) =  .036960670590051
       r(ub_Agres) =  2.742230244599933
       r(CramersV) =  -.2477168471534312
       r(p_lr)     =  .2054650645448687
       r(chi2_lr)  =  1.603095668541514
       r(c)        =  2
       r(r)        =  2
       r(N)        =  28
```

In general, the estimated odds ratio ($\widehat{\theta}$), lower and upper limits are saved for each confidence interval requested, in this case just for the Agresti interval.

References

Agresti, A. 1999. On logit confidence intervals for the odds ratio with small samples. *Biometrics* 55: 597–602.

Anscombe, F. J. 1956. On estimating binomial response relations. *Biometrika* 43: 461–464.

Cornfield, J. 1956. a statistical problem arising from retrospective studies. In *Proceedings of the Third Berkeley Symposium* vol 4: ed. J. Neyman, 135–148. Berkeley, CA: University of California Press.

Gart, J. J. 1966. Alternative analyses of contingency tables. *Journal of the Royal Statistical Society, Series B* 28: 164–179.

Haldane, J. B. S. 1955. The estimation and significance of the logarithm of a ratio of frequencies. *Annals of Human Genetics* 20: 309–311.

Woolf, B. 1955. On estimating the relationship between blood group and disease. *Annals of Human Genetics* 19: 251–253.

sg67.1 Update to univar

John R. Gleason, Syracuse University, loesljrg@accucom.net

The command `univar` of Gleason (1997) has been updated. Aside from minor improvements in the code, three new options have been introduced since its appearance. First, the `vlabel` option will display a variable's label beside its univariate summary. The other two options, `onescal` and `onehdr` modify the output when the `byvar()` option is given.

Previously, every boxplot was scaled to span columns 11–79 of the *Results* screen, even if the `byvar()` option was used to specify subgroups of observations. With the `byvar()` and `onescal` options, all boxplots for a given variable are drawn in a common scale. That is, the overall minimum and maximum appear at columns 11 and 79, and boxplots for individual subgroups contract as necessary to fit that scale.

Also formerly, a display of column headings was printed for each new subgroup of observations identified with the `byvar()` option. This style works well when there are few groups and a long varlist; it can appear cluttered when there are many groups and only a few variables being summarized. The `onehdr` option combined with `byvar()` prints the column headings just once, and shows summaries for subgroups of observations as distinct panels. That is, `byvar()` prints multiple tables by default; adding the `onehdr` option gives one table with sub-tables.

Example

To demonstrate the new options, consider the dataset `cancer.dta` supplied with Stata 6.0:

```
. describe
  obs:            48                          Patient Survival in Drug Trial
 vars:             4                          16 Nov 1998 11:49
 size:           576 (99.5% of memory free)
-------------------------------------------------------------------------------
   1. studytim  int    %8.0g                  Months to death or end of exp.
   2. died      int    %8.0g                  1 if patient died
   3. drug      int    %8.0g                  Drug type (1=placebo)
   4. age       int    %8.0g                  Patient´s age at start of exp.
-------------------------------------------------------------------------------
Sorted by:
```

First, a summary of `studytim` by `drug` groups, in the original style:

```
. univar studytim, by(drug) box
-> drug=1
                                        ----------:::::::::|:::::::::----------
Variable       n     Mean     S.D.      Min      .25      Mdn      .75      Max
-------------------------------------------------------------------------------
studytim  ---------:::::::::::::|:::::::::::----------------------------------
studytim      20     9.00     6.45     1.00     4.00     8.00    12.00    23.00
-------------------------------------------------------------------------------

-> drug=2
                                        ----------:::::::::|:::::::::----------
Variable       n     Mean     S.D.      Min      .25      Mdn      .75      Max
-------------------------------------------------------------------------------
studytim  --------:::::::::::::|:::::::::::::::::-------------------------------
studytim      14    14.93     7.60     6.00     9.00    14.00    20.00    32.00
-------------------------------------------------------------------------------

-> drug=3
                                        ----------:::::::::|:::::::::----------
Variable       n     Mean     S.D.      Min      .25      Mdn      .75      Max
-------------------------------------------------------------------------------
studytim  ------------------------:::::::::::::::|:::::::::::::::------------
studytim      14    25.36     9.58     6.00    19.00    26.50    33.00    39.00
-------------------------------------------------------------------------------
```

And the same summaries with the `vlabel`, `onescal`, and `onehdr` options in force:

```
. univar studytim, by(drug) vlab box onescal onehdr
                                          ----------:---------:---------:----------
Variable       n     Mean     S.D.      Min      .25      Mdn      .75      Max
-------------------------------------------------------------------------------

-> drug=1
studytim  Months to death or end of exp.
studytim  -----:--------:--------------------
studytim      20     9.00     6.45     1.00     4.00     8.00    12.00    23.00
-> drug=2
studytim  Months to death or end of exp.
studytim           -----:---------:-----------:----------------------
studytim      14    14.93     7.60     6.00     9.00    14.00    20.00    32.00
-> drug=3
studytim  Months to death or end of exp.
studytim           ----------------------:--------------:-----------:-----------
studytim      14    25.36     9.58     6.00    19.00    26.50    33.00    39.00
-------------------------------------------------------------------------------
```

A final modification is that typing '`univar ?`' now issues the command `which univar` thereby displaying a brief reminder of usage.

Reference

Gleason J. R. 1997. sg67: Univariate summaries with boxplots. *Stata Technical Bulletin* 36: 23–25. Reprinted in *Stata Technical Bulletin Reprints*, vol. 6, pp. 179–183.

sg115 | Bootstrap standard errors for indices of inequality

Dean Jolliffe, Center for Economic Research and Graduate Education, Czech Republic, dean.jolliffe@cerge.cuni.cz
Bohdan Krushelnytskyy, Center for Economic Research and Graduate Education, Czech Republic, bohdan.krushelnytskyy@cerge.cuni.cz

This insert provides the program `ineqerr` for estimating three indices of inequality; the Gini, Theil, and variance of logs, and bootstrap estimates of their sampling variances. The program offers three variations of the bootstrap variance estimates. The first is the standard bootstrap which assumes that the sample was selected using a simple random design. The second is a bootstrap estimate which assumes that the sample was selected in two-stages, both stages being simple random draws. The third bootstrap estimate replicates a fairly standard sample design for household survey data in which primary sampling units (PSUs) are selected with probability proportional to population (PPP) in the first stage and then in the second stage the ultimate sampling units (USUs) are selected in a simple random draw.

While there are several other programs which provide measures of inequality indices (for example, Jenkins, 1999), there are no Stata ado files which provide estimates of standard errors for the Gini, Theil, and variance of logs. It is only with estimates of the sampling variance that one can answer the important policy questions of whether inequality has changed over time or differs over regions. Mills and Zandvakili (1997) provide an interesting example of the importance of testing for the significance of differences in estimated inequality indices. Using data from the Panel Study of Income Dynamics, they show the somewhat troublesome result that the Gini and Theil indices are greater for post-tax income than for pre-tax income. Upon constructing bootstrap standard errors for the indices, they determine that none of the observed differences are statistically significant. In their paper they also show that the bootstrap estimated standard errors for the Gini are similar to the asymptotic estimates.

An advantage of the two-stage bootstrap estimates available in `ineqerr` is that if the sample was collected using a two-stage process, then the estimated standard errors will be robust to this design effect. Kish (1995) and Cochran (1996) show the importance of correcting mean values for design effects. Scott and Holt (1982) show that the magnitude of the bias for the estimated variance-covariance matrix for ordinary least squares estimates can be quite large when it is erroneously assumed that the data were collected using a simple random sample if in fact a two-stage design had been used. While there is no literature we are aware of which directly discusses the assumption of design effects in estimating standard errors for inequality indices, our empirical work suggests that correcting inequality estimates for design effects is also important. (See for example, Datt, Jolliffe and Sharma, 1998.)

Syntax

`ineqerr` *varlist* [*weight*] [`if` *exp*] [`in` *range*] [, `reps(`*#*`)` `psu(`*varname*`)` `psuwt(`*varname* | *expression*`)`]

`fweights` and `aweights` are allowed.

Options

`reps(#)` specifies the number of bootstrap replications to be performed. The default value is 100.

`psu(`*varname*`)` specifies the variable identifying the primary sampling unit. If no variable is specified, then the bootstrap replication is a single-stage, simple random draw on the sample.

`psuwt(`*varname* | *expression*`)` specifies the weight to be used for the first-stage selection process. If, for example, the population of the primary sampling unit is specified, then the bootstrap replicates a random draw with probability proportional to population. Both `psuwt` and the weight options accept either a variable name or an expression, where for example an expression might be the product of two variables. The `psuwt` option can only be used if the primary sampling unit is specified. If no weighting variable is specified for the first stage but the `psu` is specified, the bootstrap replication is two stages of a simple random draw on the sample.

Examples

To illustrate the use of `ineqerr`, we use data from the 1997 Egypt Integrated Household Survey (EIHS). The variable `pcexp_r` is a household-level measure of per capita consumption, which is adjusted to control for spatial price variation. `wt96ind` is a weighting variable which is the product of household size and strata weights. When we issue the `ineqerr` command, the following results:

```
. ineqerr pcexp_r [w=wt96ind]
pcexp_r ---------------------------------------- Real Per Capita Expenditure
(obs=2449)
Bootstrap statistics
Variable |   Reps   Observed       Bias   Std. Err.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
    Gini |    100   .3464858   .0006895   .0071348    .3323289  .3606428  (N)
         |                                            .3345012   .361028  (P)
         |                                            .3327775  .3595716 (BC)
---------+--------------------------------------------------------------------
   Theil |    100   .2232024   .0011754   .0134943    .1964268  .2499781  (N)
         |                                            .2036364  .2518677  (P)
         |                                            .1975445  .2516475 (BC)
---------+--------------------------------------------------------------------
 Varlogs |    100   .3603729   .0019088   .0128569     .334862  .3858838  (N)
         |                                            .3408104  .3913696  (P)
         |                                            .3341884  .3833428 (BC)
-------------------------------------------------------------------------------
                           N = normal, P = percentile, BC = bias corrected
```

In this case, no sample design information is passed to `ineqerr` and the program calls Stata's `bsample` utility to resample the data. In order to maintain the same sample size in each bootstrap resample, `ineqerr` ignores observations where `pcexp_r` or `wt96ind` is missing. Since the number of replications is not specified, the default value of 100 is used. The results from `bsample` are then passed to the `bstat` command to generate the standard Stata bootstrap output. For more information about the normal, percentile, and bias-corrected percentile confidence intervals, see [R] **bstrap** in the Stata manuals. For an introduction to the bootstrap principle, see Efron and Tibshirani (1993). In order to reproduce results from `ineqerr` it is necessary to set the random number seed first; see [R] **generate**.

The reported standard errors above will be correct if the sample comes from a simple random draw. This is not the case with the EIHS data, which was collected using a stratified, two-stage design. `ineqerr` can generate bootstrap estimates of the standard errors which are robust to the two-stage design by passing the information about the primary sampling unit to `ineqerr`. So, for example, we correct the standard errors above for this aspect of the sample design by issuing the following command. (We now also specify the number of replications to be 50.)

```
. ineqerr pcexp_r [w=wt96ind], reps(50) psu(psu)
pcexp_r ---------------------------------------- Real Per Capita Expenditure
(obs=2449)
Bootstrap statistics
Variable |   Reps   Observed       Bias   Std. Err.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
    Gini |     50   .3464858   .0005401   .0123159    .3217361  .3712356  (N)
         |                                            .3233636  .3683307  (P)
         |                                            .3233636  .3732372 (BC)
---------+--------------------------------------------------------------------
   Theil |     50   .2232024   .0017971   .0211976    .1806042  .2658007  (N)
         |                                            .1872635  .2664682  (P)
         |                                            .1872635  .2664682 (BC)
```

```
---------+-------------------------------------------------------------------
 Varlogs |     50   .3603729   .0000892   .0238775    .3123893  .4083565  (N)
         |                                             .32027  .4053129  (P)
         |                                             .32027  .4121322 (BC)
-------------------------------------------------------------------------------
                              N = normal, P = percentile, BC = bias corrected
```

Note that the point estimates for the inequality indices are unchanged but the estimated standard errors have all increased. If we consider the case of the Gini coefficient, the standard error increases by 73 percent when we correct for the two-stage nature of the sample design. It is worth noting that this program does not correct for stratification, and the reported standard errors are likely to be somewhat too large as the typical effect of stratification is to slightly improve the precision of the sample estimates.

As a final example, we consider a case which is not completely appropriate for the EIHS data, but may be of use when there is more complete information on the sample design. An important intuition behind the bootstrap is that the resampling of the data should replicate the way in which the data were originally collected. A fairly standard design for many household surveys is to select PSUs with probability proportional to population, and then select the USUs with a simple random draw. `ineqerr` with the `psuwt` (PSU weight) option can replicate this design if the user specifies the population estimates that were used to select the PSUs. In the case of the EIHS data, this information is not available. The EIHS data do provide PSU population estimates from the rural questionnaire, and to illustrate this feature, we treat this information as a proxy for the weights used in selecting the PSUs. The procedure used to do this is described in the section on methods and formulas. The syntax used to implement this feature follows.

```
. ineqerr pcexp_r [w=wt96ind] if rural==1, reps(50) psu(psu) psuwt(psupop)
pcexp_r ------------------------------------------ Real Per Capita Expenditure
(obs=1326)
Bootstrap statistics
Variable |   Reps   Observed       Bias   Std. Err.   [95% Conf. Interval]
---------+-------------------------------------------------------------------
    Gini |     50    .316077  -.0081707   .0136645    .2886171  .3435368  (N)
         |                                           .2858742  .3353139  (P)
         |                                           .3005531  .3383057 (BC)
---------+-------------------------------------------------------------------
   Theil |     50   .1836162  -.0053397   .0272215    .1289126  .2383198  (N)
         |                                           .1397549   .238112  (P)
         |                                           .1469196   .252798 (BC)
---------+-------------------------------------------------------------------
 Varlogs |     50   .3101612  -.0142084    .021793    .2663666  .3539558  (N)
         |                                           .2552118  .3348132  (P)
         |                                           .2766019  .3368765 (BC)
-------------------------------------------------------------------------------
                              N = normal, P = percentile, BC = bias corrected
```

Methods and Formulas

The Gini is perhaps one of the most widely used indicators of inequality, and can be written as

$$G = 1 + \frac{1}{H} - \frac{2}{H^2\mu}\sum_{h=1}^{H}\rho_h M_h$$

where H is the sample size, the ρ's are the ranks of the observations ranging from 1 to H with the richest observation having the rank of one ($\rho_1 = 1$), μ is the average value of M, and M is the measure of welfare which is sorted in descending order so that M_1 is the richest individual and M_H is the poorest individual.

When weights are introduced, we follow Deaton (1997) who shows that (nonnegative) analytical weights can be treated just as frequency weights. For the purposes of exposition, we consider the case where the weights are household size so that we are adjusting our measure to reflect inequality of individuals and not households. In this case we convert the ranks to reflect household size. We set $\rho_1 = 1$ and then $\rho_2 = 1 + w_1$, where w_1 is the size of the first household or the weight assigned to the first household. More generally:

$$\rho_{h+1} = \rho_h + w_h$$

and the average rank of all the individuals in household h can be written as

$$\bar{\rho}_h = \rho_h + 0.5(w_h - 1)$$

The weighted Gini coefficient is then

$$G = 1 + \frac{1}{N} - \frac{2}{N^2\mu_w}\sum_{h=1}^{H} w_h\bar{\rho}_h M_h$$

where N is the weighted sample size (or the number of individuals in the sample when the weight is household size), M is sorted as above, and μ_w is the weighted average value of M.

The Gini index satisfies the Pigou–Dalton principle of transfers, that is, a transfer from a richer person to a poorer person decreases the index. The magnitude of the decline, though, is determined by the difference in income rank between the two individuals and not the difference in incomes. Another characteristic of the Gini is that it is not generally decomposable.

The Theil index of income inequality is defined as follows:

$$T = (1/H)\sum_{h}(M_h/\bar{M})\log(M_h/\bar{M})$$

where H is again sample size and $\bar{M}$ is mean income. Foster (1983) shows that the Theil index satisfies several properties, including decomposability, principle of transfers, symmetry, and income scale independence. The Theil index also has the somewhat more attractive characteristic (relative to the Gini) that the magnitude of the decline in the index resulting from a transfer from a richer person to a poorer person is determined by the difference in the log of incomes.

The variance of logs index is defined as

$$V = \frac{1}{H}\sum_{h}\left[\ln M_h - \overline{\ln M}\right]^2$$

where terms are defined as above, except the mean is now of the log of income. It is perhaps worth noting that the variance of logs index does not satisfy the transfer principle, when the transfer is between two particularly rich observations.

The three bootstraps are implemented as follows. For the simple random sample (srs) we simply use Stata's `bsample` utility to bootstrap the three inequality indices. The srs, two-stage bootstrap proceeds as follows. In the first stage it counts the number of unique PSUs, say k, and then using Stata's `uniform` function, randomly selects with replacement k (not necessarily unique) PSUs. At this point it counts the number of times each PSU has been selected and this is stored for later use. To implement the second stage, the program first counts the number of USUs, say m, in each selected PSU, and then randomly selects m USUs from each selected PSU. If a PSU is selected more than once, say α times, then in the second stage the program randomly selects αm USUs from the selected PSU.

The third variant of the bootstrap is the procedure in which PSUs are first selected with probability proportional to population (or whatever weight is specified in the `psuwt` option) and then the second stage is the same as above. The first-stage selection is then a weighted, random draw. This selection is implemented by creating a new variable which is the running sum of the PSU population or weight. For the last listed PSU this variable takes the value of total population (or sum of weights) of the USUs, say N. Again assuming there are k PSUs, the first stage begins by randomly selecting k numbers using the `uniform` function ranging from 1 to N. Each of these numbers is then associated with the PSU it represents and this is then the population-weighted, randomly selected PSU. To illustrate this, consider the following table with 4 PSUs.

PSU	Population	Cumulative Population	Random Number $[1, N]$	Selected PSU
1	100	100	633	4
2	200	300	305	2
3	300	600	585	2
4	100	700	22	1

Assume the first randomly selected number is 633. The fourth PSU contains USUs ranging from 601 to 700, and so USU number 633 comes from this PSU. In the next case, suppose the randomly selected number is 305. Again note that the 305th population USU resides in the second PSU, and so this PSU is selected. Following this methodology, the resulting selected PSUs are randomly selected with probability proportional to population. (For more details on this see, for example, Cochran 1996, 250–251.)

References

Cochran, W. G. 1996. *Sampling Techniques*, 3d. ed. New York: John Wiley & Sons.

Datt, G., D. Jolliffe, and M. Sharma. 1998 A profile of poverty in Egypt–1997. Food Consumption and Nutrition Division Discussion Paper No. 49. Washington DC: International Food Policy Research Institute.

Deaton, A. 1997. Welfare, poverty, and distribution. In *The Analysis of Household Surveys*, Baltimore: Johns Hopkins University Press.

Efron, B. and R. Tibshirani. 1993. *An Introduction to the Bootstrap*. New York: Chapman & Hall.

Foster, J. 1983. An axiomatic characterization of the Theil measure of income inequality. *Journal of Economic Theory* 31: 105–121.

International Food Policy Research Institute. 1998. Egypt Integrated Household Survey (EIHS) 1997: Data and Documentation, IFPRI, Washington, DC.

Jenkins, S. 1999. sg104: Analysis of income distributions. *Stata Technical Bulletin* 48: 4–18. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 243–260.

Kish, L. 1995. *Survey Sampling* New York: Wiley Classics Library Edition.

Mills, J. and S. Zandvakili. 1997. Statistical inference via bootstrapping for measures of inequality. *Journal of Applied Econometrics* 12: 133–150.

Scott, A. J. and D. Holt. 1982. The effect of two-stage sampling on ordinary least squares methods. *Journal of American Statistical Association* 77: 848–854.

sg116 | Hotdeck imputation

Adrian Mander, MRC Biostatistics Unit, Cambridge, adrian.mander@mrc-bsu.cam.ac.uk
David Clayton, MRC Biostatistics Unit, Cambridge, david.clayton@mrc-bsu.cam.ac.uk

Syntax

`hotdeck` *varlist* [`if` *exp*] [`in` *range*] [`using` *filenameroot*] [`, by(`*varlist*`) store impute(#)`

`noise keep(`*varlist*`) command(`*command*`) parms(`*varlist*`)`]

Options

`using` specifies the root of the imputed datasets filenames. The default is `imp` and hence the datasets will be saved as `imp1.dta`, `imp2.dta`, and so on.

`by(`*varlist*`)` specifies categorical variables defining strata within which the imputation is to be carried out.

`store` specifies whether the imputed datasets are saved to disk.

`impute(#)` specifies the number of imputed datasets to generate. The number needed varies according to the percentage missing and the type of data. Often as few as 5 are sufficient, for more details see Schafer (1995).

`noise` specifies whether the individual analyses, from the `command` option, are displayed.

`keep(`*varlist*`)` specifies the variables saved in the imputed datasets in addition to the imputed variables and the `by` list. By default the imputed variables and the `by` list are always saved.

`command(`*command*`)` specifies the analysis performed on every imputed dataset.

`parms(`*varlist*`)` specifies the parameters of interest from the analysis. If the command is a regression command then the parameter list can include a subset of the variables specified in the regression command. The final output consists of the combined estimates of these parameters.

Description

`hotdeck` will tabulate the missing data patterns within the *varlist*. A row of data with missing values in any of the variables in the *varlist* is defined as a "missing line" of data, similarly a "complete line" is one where all the variables in the *varlist* contain data. The `hotdeck` procedure replaces the *varlist* variables in the "missing lines" with the corresponding values in the "complete lines." `hotdeck` should be used several times within a multiple imputation sequence since missing data are imputed stochastically rather than deterministically. The *nmiss* missing lines in each stratum of the data described by the `by` option are replaced by lines sampled from the *nobs* complete lines in the same stratum. The approximate Bayesian bootstrap method of Rubin and Schenker (1986) is used; first a bootstrap sample of *nobs* lines are sampled with replacement from the complete lines, and the *nmiss* missing lines are sampled at random (again with replacement) from this bootstrap sample.

A major assumption with the hotdeck procedure is that the missing data are either missing completely at random (MCAR) or is missing at random (MAR) Rubin (1987), the probability that a line is missing varying only with respect to the categorical variables specified in the `by` option.

If a dataset contains many variables with missing values then it is possible that many of the rows of data will contain at least one missing value. The `hotdeck` procedure will not work very well in such circumstances. There are more elaborate methods that **only** replace missing values, rather than the whole row, for imputed values. These multivariate multiple imputation methods are discussed by Schafer (1995).

Example

The example comes from a simulation study of a two-phase sampling design. The outcome variable Y is binary and indicates whether a subject is a case or control. The aim is to estimate the effect of the true exposure (X, binary exposure) on the outcome. If the probability of being exposed is low a large sample size is needed to investigate this effect. A more efficient design might be to collect data on is a surrogate measure Z of the true exposure X, and based on this information collect a subsample of true exposure data. This is particularly useful if the cost of obtaining X is large relative to Z.

The simulation model used to generate Y and Z conditional on X are detailed below:

$$P(Y = 1|X) = \frac{\exp(1 + X)}{1 + \exp(1 + X)}$$

$$P(Z = 1|X = 1) = P(Z = 0|X = 0) = 0.99$$

For the example it is assumed that the true exposure is observed only in the subsample, although the surrogate is fully observed. The table of the surrogate against outcome is below

```
           |          surr
         y |         0          1 |     Total
-----------+----------------------+----------
   Control |      2564         71 |      2635
      Case |      6916        449 |      7365
-----------+----------------------+----------
     Total |      9480        520 |     10000
```

The sampling scheme to determine the true exposure takes proportionally more subjects from "rarer" cells than "common" cells. The rarest cell being the exposed controls and the most common the unexposed case. In this example, 80 subjects from the 10,000 are selected to measure the true exposure. Here is the resulting table:

```
           |          xmiss
         y |      Unexp       Exp |     Total
-----------+----------------------+----------
   Control |        29         11 |        40
      Case |        26         14 |        40
-----------+----------------------+----------
     Total |        55         25 |        80
```

Since this is a simulation study, the true exposure is known for everyone, but in practice the analyst will only have the information contained in the last two tables. Using logistic regression to analyze the last table gives the following model:

```
Logit estimates                                   Number of obs   =         80
                                                  LR chi2(1)      =       0.52
                                                  Prob > chi2     =     0.4689
Log likelihood = -55.189481                       Pseudo R2       =     0.0047
------------------------------------------------------------------------------
       y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
---------+--------------------------------------------------------------------
   xmiss |   .3503613   .4850587    0.722   0.470    -.6003363    1.301059
   _cons |  -.1091993    .270082   -0.404   0.686    -.6385503    .4201517
------------------------------------------------------------------------------
```

An important point is that the number of observations is 80 out of the 10,000 possible lines of data. Analyzing complete lines of data is called *case deletion*. The estimates from the case deletion analysis are inaccurate and the constant term's confidence interval does not contain 1. As stated previously, the logistic model's linear predictor is $1 + X$, hence both confidence intervals should contain 1. As the sampling was conditional on the surrogate and the outcome the missing data is not MCAR but rather MAR. This explains the biased estimates.

The hotdeck imputation can be used to impute the true exposure conditional on the surrogate measure of exposure. This should lead to unbiased results and improvements in the standard errors. Since $X|Z$ is imputed and Z is categorical a `by(z)` option is used. The variable with missing data is put first in the variable list and analysis is carried out by the `command(logit y xmiss)` statement. When the analysis is performed the variable `xmiss` has imputed values for each missing datum. The parameters of interest are `_b[xmiss]` and `_b[_cons]` and these are accessed via the `parms(xmiss _cons)` option.

```
. hotdeck xmiss, command(logit y xmiss) parms(xmiss _cons) by(surr) impute(10)
DELETING all matrices....
Table of the Missing data patterns
```

```
 * signifies missing and - is not missing
Varlist order: xmiss
    pattern |      Freq.     Percent        Cum.
------------+-----------------------------------
          * |       9920       99.20       99.20
          - |         80        0.80      100.00
------------+-----------------------------------
      Total |      10000      100.00

                                    Number of Obs.           =  10000
                                    No. of Imputations       =  10
                                    % Lines of Missing Data  =  99.2 %
                                    F( 73.596 ,2)            =  1384.8739
                                    Prob > F                 =     0.0000
------------------------------------------------------------------------------
Variable |  Average   Between   Within   Total      df        t    p-value
         |  Coef.     Imp. SE   Imp. SE   SE
---------+--------------------------------------------------------------------
xmiss    |  0.9002     0.116     0.168   0.207    75.833     4.346   0.000
_cons    |  1.0045     0.005     0.023   0.024  3465.545    42.628   0.000
---------+--------------------------------------------------------------------
Variable |  [95% Conf. Interval]
---------+--------------------------------------------------------------------
xmiss    |    0.4266    1.3739
_cons    |    0.9517    1.0573
------------------------------------------------------------------------------
```

The output from hotdeck is similar to most regression commands. However, the between and within standard errors are also included. There is a global test of whether all the coefficients are 0, the assumptions of this test may be wrong when the number of imputations are small and/or the between imputation standard errors are larger than the within standard errors. One remedy for this is to increase the number of imputations, however, this may not solve the problem. The Wald tests do not have these limitations.

Saving imputed datasets

It may be preferable to use the `hotdeck` routine to just impute datasets and do the analysis from the saved data files.

```
. hotdeck xmiss using myfile, store keep(y) by(surr) impute(10)
```

The imputed datasets saved to disk, in the example above, will be `myfile1.dta` through `myfile10.dta`. Each file will contain the three variables `xmiss`, `y`, and `surr`. By default, the files contain the varlist and the variables in the `by()` option, hence the need to specify the `keep(y)` option.

References

Rubin, D. B. 1987. *Multiple Imputation for Nonresponse in Surveys*. New York: John Wiley & Sons.

Rubin, D. B. and N. Schenker. 1986. Multiple imputation for interval estimation from simple random samples with ignorable nonresponse. *Journal of the American Statistical Association* 81: 366–374.

Schafer, J. L. 1995. *Analysis of Incomplete Multivariate Data by Simulation.* London: Chapman and Hall.

sg117 | Robust standard errors for the Foster–Greer–Thorbecke class of poverty indices

Dean Jolliffe, Center for Economic Research and Graduate Education, Czech Republic, dean.jolliffe@cerge.cuni.cz
Anastassia Semykina, Center for Economic Research and Graduate Education, Czech Republic, anastassia.semykina@cerge.cuni.cz

Description

Several inserts to the *Stata Technical Bulletin* provide programs to estimate a wide array of poverty indices (see for example, Jenkins 1999, and van Kerm 1999), but we are aware of no programs which provide estimates of standard errors for these indices. Yet, in order to answer questions of whether poverty has increased or decreased over time, or whether poverty is worse in certain regions, estimates of the sampling variance for the indices are required. Indeed it is hard to think of a poverty-related question which a policy maker might have, that doesn't require an estimate of whether some difference in indices is statistically significant. This insert describes the command `sepov` which provides estimates of standard errors for the Foster–Greer–Thorbecke (1984; hereafter referred to as FGT) class of poverty indices. By default, `sepov` reports the headcount, poverty-gap, and squared poverty-gap indices, but the user may request any variant of the FGT indices.

Kakwani (1993) presents a simple method for calculating standard errors for the FGT indices, which this program implements as one option. The Kakwani formula for the variance of P_0, the headcount index, is $P_0(1-P_0)/(n-1)$, where n is the sample size. The formula for all other variance estimates of the FGT indices is $(P_{2\alpha} - P_\alpha^2)/(n-1)$ where P_α is defined below. While the Kakwani standard errors are tremendously useful when one doesn't have access to the unit-record data, an unfortunate aspect of the estimated standard errors is that they assume the sample was collected using a simple random design. Poverty estimates, though, are quite often constructed from nationally representative household survey data, and this type of data almost always comes from a complex sample design.

Howes and Lanjouw (1998) present compelling evidence that estimated standard errors for the FGT poverty indices can have large biases when erroneous assumptions are made on the nature of the sample design. In particular they show that the Kakwani standard errors significantly underestimate the correct standard errors when the data come from a multistage sample design. By using Stata's `svymean` command, `sepov` provides estimated standard errors which are robust to complex survey design, including stratification and multi-stages.

Syntax

`sepov` *varlist* [*weight*] [`if` *exp*] [`in` *range*] , `povline(`*varname*`)` [`alpha(`#`)` `strata(`*varname*`)`

`psu(`*varname*`)` `fpc(`*varname*`)` `by(`*varlist*`)` {`complete` | `available`} `subpop(`*varname*`)`

`srssubpop` `nolabel` `level(`#`)` `ci` `deff` `deft` `meff` `meft` `obs` `size`]

`pweights` are allowed.

Options

`povline(`*varname*`)` specifies the poverty line, which can be either a scalar or a variable. By accepting the poverty line as a variable, we allow for the possibility that the poverty line may vary over the sample. A variable poverty line is one way to incorporate information about spatial price variation in obtaining the bundle of goods described by the poverty line.

`alpha(`#`)` specifies the type of poverty index. By default `sepov` reports the headcount, poverty-gap and squared poverty-gap indices, which correspond to alpha taking the values of 0, 1, and 2, respectively. In addition to these, the user may specify any nonnegative value of alpha. As alpha increases, the measure becomes more sensitive to inequality among the poor.

All other options are as specified in Stata's `svymean` command. If the user does not specify the strata or primary sampling unit, the resulting standard errors will be the Kakwani standard errors. It is important to note that using the `if` or `in` conditions may result in incorrect variance estimates, just as with using these conditions with `svymean`. To obtain correct estimates of the standard errors for poverty decomposed into subgroups, it is recommended to use the `by` or `subpop` options.

Examples

To illustrate the use of `sepov`, we use data from the 1997 Egypt Integrated Household Survey (EIHS). The variable `pcexp_r` is a household-level measure of per capita consumption, which is adjusted to control for spatial price variation, and `z_r` is the poverty line. `wt96ind` is a weighting variable which is the product of sample weights resulting from stratification and household size, since we are interested in the welfare of individuals and not households. When we issue the `sepov` command, the following results:

```
. sepov pcexp_r [w=wt96ind], p(z_r)
(sampling weights assumed)
Poverty measures for the variable pcexp_r: Real Per Capita Expenditure
Survey mean estimation
pweight:  wt96ind                                 Number of obs    =      2449
Strata:   <one>                                   Number of strata =         1
PSU:      <observations>                          Number of PSUs   =      2449
                                                  Population size  =  14532.16
------------------------------------------------------------------------------
    Mean |   Estimate    Std. Err.   [95% Conf. Interval]        Deff
---------+--------------------------------------------------------------------
      p0 |   .2651925    .0106545    .2442997    .2860852    1.426075
      p1 |   .0669121    .0035529    .0599451    .0738791    1.465418
      p2 |   .0255641    .0018219    .0219914    .0291368    1.458858
------------------------------------------------------------------------------
```

We learn from the output that, for example, the headcount index is 0.265 and the Kakwani standard error for this index is 0.01. The next example illustrates two additional features. We note again that the Kakwani standard errors listed above are correct only if the data result from a simple random sample. The EIHS data, as with most household survey data, were collected

using a complex design (stratified, two-stage design). By specifying the `psu` and `strata` options, the reported standard errors will be corrected for the design effects. We will also specify alpha equal to three, to examine what happens to poverty when gaps between the poverty line are "penalized" more heavily.

```
. sepov pcexp_r [w=wt96ind], p(z_r) psu(psu) strata(strata) a(3)
Poverty measures for the variable pcexp_r: Real Per Capita Expenditure
Survey mean estimation
pweight:  wt96ind                                 Number of obs    =      2449
Strata:   strata                                  Number of strata =         5
PSU:      psu                                     Number of PSUs   =       126
                                                  Population size  =  14532.16
------------------------------------------------------------------------------
    Mean |   Estimate    Std. Err.   [95% Conf. Interval]        Deff
---------+--------------------------------------------------------------------
      p0 |   .2651925    .0167525    .2320265    .2983584    3.525606
      p1 |   .0669121    .0060812    .0548727    .0789515    4.293216
      p2 |   .0255641     .003091    .0194447    .0316835    4.198881
      p3 |     .01187    .0017642    .0083774    .0153626     3.68886
------------------------------------------------------------------------------
```

Note that the point estimates for P0, P1, P2 did not change, but that the standard errors all increased. If we consider the headcount index (P0), the estimated standard error increases by 57 percent when we correct for the stratification and two-stage aspects of the sample design.

Methods and Formulas

The FGT poverty index, also referred to as P_α is given by

$$P_\alpha = 1/n \sum_i I(y_i < z)[(z - y_i)/z]^\alpha$$

where n is the sample size, i subscripts the household or individual, y is the relevant measure of welfare, z is the poverty line, and I is an indicator function which takes the value of one if the statement is true and zero otherwise.

When $\alpha = 0$, the resulting measure is the headcount index which provides an estimate of the proportion of the population living in poverty. When $\alpha = 1$, the FGT index results in the poverty-gap index which provides a measure of the depth of poverty. The squared poverty-gap index, which is sensitive to the extent of inequality among the poor, results when $\alpha = 2$. In addition to these three measures, which are provided by default, the user may specify any nonnegative value of α.

Foster, et al. (1984) show that for any income vector y, broken down into m subgroup income vectors, $y^{(1)}, \ldots, y^{(m)}$, P_α is additively decomposable with population share weights:

$$P_\alpha(y; z) = \sum_{j=1}^{m} (n_j/n) P_\alpha(y^{(j)}; z)$$

Exploiting this property is really the innovation of `sepov`. This property allows us to treat each observation as a subgroup, which means that the average value resulting from Stata's `summarize` or `svymean` command will be the sample estimate of P_α. `sepov` passes observation-specific estimates of P_α to Stata's `svymean` command, which then provides the user with estimates of the poverty indices as well as their standard errors which are robust to design effects.

References

Foster, J., J. Greer, and E. Thorbecke. 1984. A class of decomposable poverty measures. *Econometrica* 52: 761–765.

Howes, S. and J. O. Lanjouw. 1998. Does sample design matter for poverty comparisons. *Review of Income and Wealth* 44(1): 99–109.

International Food Policy Research Institute. Egypt Integrated Household Survey (EIHS) 1997: Data and Documentation. IFPRI, Washington, DC 1998.

Jenkins, S. 1999. sg104: Analysis of income distributions. *Stata Technical Bulletin* 48: 4–18. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 243–260.

Kakwani, N. 1993. Statistical inference in the measurement of poverty. *Review of Economics and Statistics* 75(4): 632–39.

van Kerm, P. 1999. sg108: Computing poverty indices. *Stata Technical Bulletin* 48: 29–33. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 274–278.

sg118	Partitions of Pearson's χ^2 for analyzing two-way tables that have ordered columns

Rory Wolfe, Royal Children's Hospital, Australia, wolfer@cryptic.rch.unimelb.edu.au

Introduction

This insert describes the command `opartchi` which is useful for exploratory analysis of data that can be summarized in a two-way contingency table with ordered columns. The aim is to describe differences between the rows of the table in terms of the distribution of the row totals across the ordered columns, i.e. where these are viewed as categories of an ordinal outcome variable. The command provides test statistics that summarize the extent to which the rows distributions differ, firstly in their location and secondly in their dispersion, across the ordinal scale.

Two types of test statistics are provided. By default, Pearson's χ^2 statistic is partitioned into components that describe the appropriate row differences (Best 1994, Best and Rayner, 1998). As an option, an analysis of the deviance statistics from fitting log-linear models to the data (Agresti 1984) is produced. Both of these procedures require the attachment of scores to the ordered columns and by default, increasing integer scores are used but user-defined scores can be optionally provided. If midrank scores are used then the location and dispersion components of Pearson's χ^2 test are equivalent to statistics given by Nair (1986).

If, in addition to the columns, the rows of the contingency table have a natural ordering, then an optional test can be performed to assess the extent to which differences between the location of row distributions can be explained by an increasing effect with row order. This "test of trend" requires the attachment of scores to the rows and by default increasing integer scores are used, although an option is provided that allows for user-defined scores. With default row scores and midrank column scores this "test of trend" component of Pearson's χ^2 is equivalent to the test statistic provided by `nptrend`; see the Stata FAQ at `http://www.stata.com/support/faqs/stat/trend.html` for a further discussion of this issue.

The data can be provided either at an individual level (one ordinal outcome value per observation) or at a contingency table level in which case there is a variable in the dataset that contains the observed counts for each cell of the table. The latter requires the use of the optional frequency weighting.

Method

In general the $R \times C$ contingency table has rows $i = 1, \ldots, R$ (possibly ordered in which case there are corresponding scores x_i that increase with i) and ordered columns $j = 1, \ldots, C$ with attached scores y_j that increase with j. The table has observed cell counts n_{ij}, fixed row totals $n_{i.}$ (where $.$ as a subscript indicates summation over the corresponding subscript of n_{ij}) and underlying cell probabilities π_{ij}.

Pearson's χ^2 statistic, X^2, on $(R-1)(C-1)$ degrees of freedom is given by

$$X^2 = \sum_{ij} \frac{(n_{ij} - p_j n_{i.})^2}{p_j n_{i.}}$$

where a null hypothesis of no association provides expected cell probabilities $p_j = n_{.j}/n_{..}$. Best and Rayner (1998) partition X^2 into $C-1$ individual χ^2 statistics, each on $R-1$ degrees of freedom, and each describing a different aspect of the between-row differences in the distribution of observations across the ordered columns. In particular, the first two components, $V_1 = \sum_i v_{1i}^2$ and $V_2 = \sum_i v_{2i}^2$, where

$$v_{1i} = \sum_j \frac{n_{ij}}{n_{i.}} \frac{y_j - \mu}{\sqrt{\mu_2}}$$

$$v_{2i} = \sum_j \frac{n_{ij}}{n_{i.}} \frac{(y_j - \mu)^2 - \frac{\mu_3 (y_j - \mu)}{\mu_2} - \mu_2}{\sqrt{\mu_4 - \mu_3^2/\mu_2 - \mu_2^2}}$$

and

$$\mu = \sum_j y_j p_j$$

$$\mu_r = \sum_j (y_j - \mu)^r p_j$$

summarize row differences in terms of the location and dispersion, respectively, of the distribution of the row totals. The test of trend is obtained by regressing the individual row contributions to the location component, v_{1i}, on the row scores, x_i, as described by Best and Rayner (1998).

The deviance statistics from log-linear modeling arise as follows. The independence model (Agresti 1984)

$$\log(\pi_{ij}) = \theta_0 + \theta_{Xi} + \theta_{Yj}$$

is initially fitted to obtain the independence deviance statistic (where the corner constraints $\theta_{Y1} = \theta_{X1} = 0$ are employed). The change in deviance between this model and the row effects model,

$$\log(\pi_{ij}) = \theta_0 + \theta_{Xi} + \theta_{Yj} + \beta_{Li}(y_j - \bar{y}) \qquad \beta_{L1} = 0$$

gives the location-component deviance statistic. The change in deviance between this second model and the dispersion model

$$\log(\pi_{ij}) = \theta_0 + \theta_{Xi} + \theta_{Yj} + \beta_{Li}(y_j - \bar{y}) + \beta_{Di}(y_j - \bar{y})^2 \qquad \beta_{L1} = \beta_{D1} = 0$$

gives the dispersion-component deviance statistic. If the rows have a natural order then the linear-trend-component deviance statistic is given by the difference between the deviance for the independence model above and the uniform association model

$$\log(\pi_{ij}) = \theta_0 + \theta_{Xi} + \theta_{Yj} + \beta_{XY}(x_j - \bar{x})(y_j - \bar{y})$$

Syntax of the opartchi command

`opartchi` *column_var* [*weight*] [`if` *exp*] [`in` *range*] , `by(`*row_var*`)` [`table` `loglinear`

`score(`*column_score_var*`)` `orows` `rscore(`*row_score_var*`)`]

`fweights` are allowed.

Options

`by(`*row_var*`)` is required. The variable *row_var* gives the rows of the contingency table.

`table` is optional and displays the two-way contingency table (with row percentages) that is to be analyzed.

`loglinear` is an option for displaying deviance statistics from log-linear models alongside the counterpart Pearson's χ^2 partitions.

`score(`*column_score_var*`)` is an option to supply user-defined scores for the column values instead of using the default increasing integer scores.

`orows` is optional and should only be used where *row_var* is an ordinal variable. This option display the nested test for trend within the location component of row differences.

`rscore(`*row_score_var*`)` is an option to supply user-defined scores for the row values instead of using the default increasing integer scores.

Examples

Comparing two multinomial distributions: a $2 \times C$ table

The data reproduced in Table 1 on consumers' subjective rating of the sweetness of chocolate products was analyzed by Best (1994).

Table 1: A cross-cultural study of sweetness

	Sweetness liking score						
Consumer nationality	1	2	3	4	5	6	7
Australian	2	1	6	1	8	9	6
Japanese	0	1	3	4	15	7	1

Suppose these data are stored as a series of responses (in a variable called `sweet_sc`) to the chocolate's sweetness from individuals identified by their country of residence (stored in the variable `country`). To obtain the midrank scores in Stata 6.0 the command

```
. egen midrank=rank(sweet_sc)
```

can be used, and these scores are then attached to the ordered categories of the response scale when analyzing the differences between the two cultures with respect to their liking of the sweetness of the chocolate:

```
. opartchi sweet_sc, by(country) score(midrank)
```

The output

```
                                    Chi-square tests
                           df     Chi-square  P-value
Independence                6      10.699812   0.0981
-------------------------------------------------------
                         Components of independence test
Location                    1      .60488714   0.4367
Dispersion                  1      7.8406825   0.0051
```

agrees with the χ^2 results of Best (1994). The conclusion here is that while Australian and Japanese consumers have the same average liking of chocolate's sweetness, there is a strong difference between the cultures in the spread of responses across the ordinal scale, with Australian consumers being more likely to give extreme responses.

Ordinal by ordinal 2-way tables

Reproduced in Table 2 are data presented by Agresti (1984, 12) on an undesirable side-effect outcome from different operations for treating duodenal ulcer patients.

Cross-classification of Dumping Severity and Operation

	Dumping severity			
Operation	None	Slight	Moderate	Total
Drainage and vagotomy	61	28	7	96
25% resection and vagotomy	68	23	13	104
50% resection and vagotomy	58	40	12	110
75% resection	53	38	16	107
Total	240	129	48	417

When analyzing data that have been summarized in this way, it is most convenient to enter the information into Stata in contingency table format whereby for each cell of the table indicated by the values of the variables `op` (numbered 1–4) and `severity` (numbered 1–3), the number of observations is given by the variable `count`. Note that the operations and the outcome are both ordered in terms of severity. Analysis of this table can be performed with the command

```
. opartchi severity [fweight=count], by(op) loglin orows
```

which produces the output

```
                                Chi-square tests       Log-linear models
                           df     Chi-square  P-value      Deviance   P-value
Independence                6       10.54191   0.1036     10.878224    0.0922
-------------------------------------------------------------------------------
                         Components of independence test  (/ deviance)
Location                    3       6.454436   0.0915     6.4748355    0.0907
Dispersion                  3       4.087474   0.2522     4.4033885    0.2211
                         Ordinal X Ordinal 2-way table
Trend in location effect    1      6.1934378   0.0128     6.2884399    0.0122
```

The log-linear independence and trend model deviances are given by Agresti (1984, 81) and agree with the results above. The Pearson's χ^2 results are very similar as should be the case unless the data are sparse. The differences between operations

can not be explained succinctly by either differences in average dumping severity nor differences in the spread of dumping severity experiences. However, there is a significant trend to more severe dumping with the increasing severity of the operation (operation D corresponds to the greatest removal of stomach).

Saved results

`r(df_i)`	Degrees of freedom for Pearson's χ^2
`r(df)`	Degrees of freedom for components
`r(chi2_i)`	Pearson's χ^2 statistic
`r(chi2_l)`	Location component χ^2 statistic
`r(chi2_d)`	Dispersion component χ^2 statistic

Optionally

`r(chi2_tr)`	Linear trend component χ^2 statistic
`r(D_i)`	Log-linear model: independence deviance statistic
`r(D_l)`	Log-linear model: location-component deviance statistic
`r(D_d)`	Log-linear model: dispersion-component deviance statistic
`r(D_tr)`	Log-linear model: linear-trend-component deviance statistic

Discussion

In the chocolate example, after consideration of the first two components of Pearson's χ^2 there is a "residual" χ^2 of $(10.7 - 0.6 - 7.8) = 2.3$ on $(6 - 1 - 1) = 4$ degrees of freedom. In this example there are no higher order components that could describe a significant part of this residual. In general, I expect higher order components to be unhelpful in most practical examples given the complexity of interpretation.

For analyzing sparse data the properties of the components of Pearson's χ^2 are not well documented but presumably are similar to those of the overall Pearson's χ^2 for association as alluded to by Armitage and Berry (1994, 404). Best (1994) presents p values from Monte-Carlo simulations for sparse tables calculated for the chocolate example given above: Independence $p = 0.08$, location $p = 0.46$, dispersion $p = 0.005$, and for the residual $p = 0.69$. There is no change to our conclusions in the light of these more appropriate p values.

The two sets of statistics provided by this command are among a number of such sets that have been suggested for use in this situation. Investigation of the power of some alternatives, in the specific situation of possible dispersion effects, are given by Hilton (1996) and Best (1994).

Acknowledgments

While accepting full responsibility for the workings of the command, I thank Suzanna Vidmar and Patty Chondros for help debugging the program and accompanying descriptions.

References

Agresti, A. 1984. *Analysis of Ordinal Categorical Data.* New York: John Wiley & Sons.

Armitage, P. and G. Berry. 1994. *Statistical Methods in Medical Research.* 3d ed. Oxford: Blackwell Scientific Publications.

Best, D. J. 1994. Nonparametric comparison of two histograms. *Biometrics* 50: 538–541.

Best, D. J. and J. C. W. Rayner. 1998. Nonparametric analysis of ordinal categorical response data with factorial structure. *Applied Statistics* 47: 439–446.

Hilton, J. F. 1996. The appropriateness of the Wilcoxon test in ordinal data. *Statistics in Medicine* 15: 631–645.

Nair, V. 1986. Testing in industrial experiments with ordered categorical data. *Technometrics* 28: 283–311.

sts14	Bivariate Granger causality test

J. Sky David, Texas A&M University, jsdavid@politics.tamu.edu

This insert describes the program `granger` which implements asymptotic and finite sample bivariate Granger causality tests (Granger 1969) for time-series data, as presented in Hamilton (1994). To illustrate the use of `granger`, I test my research hypothesis that the unemployment rate Granger-causes presidential approval. The time period for this analysis is from the first quarter of 1950 to the fourth quarter of 1994.

Syntax

`granger` *varname1 varname2* [`if` *exp*] [`in` *range*] , `lags(`*#*`)`

Note that `tsset` is required before implementing the procedure.

Options

`lags(`*#*`)` specifies the number of lags to be included in the test. This is not an option. The maximum number of lags allowable is 10.

Example

The presidential approval data (King and Ragsdale 1988, Edwards III and Gallup Opinion Index, Gallup Poll Monthly), were placed into electronic format, and obtained from Burbach (1995). The economic data (unemployment rate) were gathered by the United States Bureau of Labor Statistics (1999).

Since the data for the presidential approval series were gathered at different monthly intervals, I used the average percentage of those respondents that approved of the president's job for a given quarter in order to obtain the quarterly percentages. There was a missing time point for the presidential approval series in the third quarter of 1952. Therefore, I used the average of the presidential approval series from the second and fourth quarters of 1952 to impute the missing time point in the third quarter of 1952. The monthly unemployment rate data were averaged on a quarter-by-quarter basis for this analysis.

I modeled the hypothesized relationship using three lags. The findings indicate that the unemployment rate Granger-causes presidential approval in both asymptotic and finite sample tests.

```
. describe
Contains data from granger.dta
  obs:           172
 vars:             3                          14 Jun 1999 19:52
 size:         2,408 (99.7% of memory free)
------------------------------------------------------------------------------
   1. yrqtr     int    %8.0g                  Year and Quarter
   2. unempqtr  float  %9.0g                  Quarterly Unemployment
   3. yapqtr    float  %9.0g                  Presidential Approval
------------------------------------------------------------------------------
Sorted by:  yrqtr
. tsset yrqtr
        time variable:  yrqtr, 5001 to 9204, but with gaps
. granger unempqtr yapqtr, lags(3)
Granger Causality test (asymptotic)  unempqtr    --->    yapqtr,
 H0:  unempqtr  does not Granger-cause  yapqtr,
F(   3,   165)  =  3.5155199
Prob  >  F  =    .017
Granger Causality test
 H0:  unempqtr  does not Granger-cause  yapqtr,
chi2(3)  =  10.99399
Prob  > chi2   =   0.0118
```

References

Burbach, D. T. 1995. http://web.mit.edu/afs/athena.mit.edu/user/d/b/dburbach/www/papers/presaprv

Edwards III, G. and A. Gallup. 1990. Presidential Approval. Baltimore: Johns Hopkins University Press.

Gallup Opinion Index. (various issues). American Institute of Public Opinion: Princeton, N.J.

Gallup Poll Monthly. (various issues). The Gallup Poll: Princeton, N.J.

Granger, C. 1969. Investigating causal relationships by econometric models and cross-spectral Methods. *Econometrica* 37: 424–438.

Hamilton, J. D. 1994. *Time Series Analysis*. Princeton University Press: Princeton, N.J.

King, G. and L. Ragsdale. 1988. The Elusive Executive: Discovering Statistical Patterns in the Presidency. *Congressional Quarterly Press*: Washington, D.C.

United States Bureau of Labor Statistics. 1999. http://146.142.4.24/cgi-bin/surveymost?bls

STB categories and insert codes

Inserts in the STB are presently categorized as follows:

General Categories:			
an	announcements	*ip*	instruction on programming
cc	communications & letters	*os*	operating system, hardware, & interprogram communication
dm	data management		
dt	datasets	*qs*	questions and suggestions
gr	graphics	*tt*	teaching
in	instruction	*zz*	not elsewhere classified
Statistical Categories:			
sbe	biostatistics & epidemiology	*ssa*	survival analysis
sed	exploratory data analysis	*ssi*	simulation & random numbers
sg	general statistics	*sss*	social science & psychometrics
smv	multivariate analysis	*sts*	time-series, econometrics
snp	nonparametric methods	*svy*	survey sampling
sqc	quality control	*sxd*	experimental design
sqv	analysis of qualitative variables	*szz*	not elsewhere classified
srd	robust methods & statistical diagnostics		

In addition, we have granted one other prefix, *stata*, to the manufacturers of Stata for their exclusive use.

Guidelines for authors

The Stata Technical Bulletin (STB) is a journal that is intended to provide a forum for Stata users of all disciplines and levels of sophistication. The STB contains articles written by StataCorp, Stata users, and others.

Articles include new Stata commands (ado-files), programming tutorials, illustrations of data analysis techniques, discussions on teaching statistics, debates on appropriate statistical techniques, reports on other programs, and interesting datasets, announcements, questions, and suggestions.

A submission to the STB consists of

1. An insert (article) describing the purpose of the submission. The STB is produced using plain TEX so submissions using TEX (or LATEX) are the easiest for the editor to handle, but any word processor is appropriate. If you are not using TEX and your insert contains a significant amount of mathematics, please FAX (409–845–3144) a copy of the insert so we can see the intended appearance of the text.

2. Any ado-files, `.exe` files, or other software that accompanies the submission.

3. A help file for each ado-file included in the submission. See any recent STB diskette for the structure a help file. If you have questions, fill in as much of the information as possible and we will take care of the details.

4. A do-file that replicates the examples in your text. Also include the datasets used in the example. This allows us to verify that the software works as described and allows users to replicate the examples as a way of learning how to use the software.

5. Files containing the graphs to be included in the insert. If you have used STAGE to edit the graphs in your submission, be sure to include the `.gph` files. Do not add titles (e.g., "Figure 1: ...") to your graphs as we will have to strip them off.

The easiest way to submit an insert to the STB is to first create a single "archive file" (either a `.zip` file or a compressed `.tar` file) containing all of the files associated with the submission, and then email it to the editor at `stb@stata.com` either by first using `uuencode` if you are working on a Unix platform or by attaching it to an email message if your mailer allows the sending of attachments. In Unix, for example, to email the current directory and all of its subdirectories:

```
tar -cf - . | compress | uuencode xyzz.tar.Z > whatever
mail stb@stata.com < whatever
```

International Stata Distributors

International Stata users may also order subscriptions to the *Stata Technical Bulletin* from our International Stata Distributors.

Company:	Applied Statistics & Systems Consultants
Address:	P.O. Box 1169 17100 NAZERATH-ELLIT Israel
Phone:	+972 (0)6 6100101
Fax:	+972 (0)6 6554254
Email:	assc@netvision.net.il
Countries served:	Israel

Company:	Axon Technology Company Ltd
Address:	9F, No. 259, Sec. 2 Ho-Ping East Road TAIPEI 106 Taiwan
Phone:	+886-(0)2-27045535
Fax:	+886-(0)2-27541785
Email:	hank@axon.axon.com.tw
Countries served:	Taiwan

Company:	Chips Electronics
Address:	Lokasari Plaza 1st Floor Room 82 Jalan Mangga Besar Raya No. 82 JAKARTA Indonesia
Phone:	62 - 21 - 600 66 47
Fax:	62 - 21 - 600 66 47
Email:	puyuh23@indo.net.id
Countries served:	Indonesia

Company:	Dittrich & Partner Consulting
Address:	Kieler Strasse 17 5. floor D-42697 Solingen Germany
Phone:	+49 2 12 / 26 066 - 0
Fax:	+49 2 12 / 26 066 - 66
Email:	sales@dpc.de
URL:	http://www.dpc.de
Countries served:	Germany, Austria, Italy

Company:	IEM
Address:	P.O. Box 2222 PRIMROSE 1416 South Africa
Phone:	+27-11-8286169
Fax:	+27-11-8221377
Email:	iem@hot.co.za
Countries served:	South Africa, Botswana, Lesotho, Namibia, Mozambique, Swaziland, Zimbabwe

Company:	MercoStat Consultores
Address:	9 de junio 1389 CP 11400 MONTEVIDEO Uruguay
Phone:	598-2-613-7905
Fax:	Same
Email:	mercost@adinet.com.uy
Countries served:	Uruguay, Argentina, Brazil, Paraguay

Company:	Metrika Consulting
Address:	Mosstorpsvagen 48 183 30 Taby STOCKHOLM Sweden
Phone:	+46-708-163128
Fax:	+46-8-7924747
Email:	sales@metrika.se
Countries served:	Sweden, Baltic States, Denmark, Finland, Iceland, Norway

Company:	Ritme Informatique
Address:	34, boulevard Haussmann 75009 Paris France
Phone:	+33 (0)1 42 46 00 42 +33 (0)1 42 46 00 33
Email:	info@ritme.com
URL:	http://www.ritme.com
Countries served:	France, Belgium, Luxembourg

(*List continued on next page*)

International Stata Distributors

(*Continued from previous page*)

Company:	Scientific Solutions S.A.
Address:	Avenue du Général Guisan, 5 CH-1009 Pully/Lausanne Switzerland
Phone:	41 (0)21 711 15 20
Fax:	41 (0)21 711 15 21
Email:	info@scientific-solutions.ch
Countries served:	Switzerland

Company:	Smit Consult
Address:	Doormanstraat 19 5151 GM Drunen Netherlands
Phone:	+31 416-378 125
Fax:	+31 416-378 385
Email:	J.A.C.M.Smit@smitcon.nl
URL:	http://www.smitconsult.nl
Countries served:	Netherlands

Company:	Survey Design & Analysis Services P/L
Address:	249 Eramosa Road West Moorooduc VIC 3933 Australia
Phone:	+61 (0)3 5978 8329
Fax:	+61 (0)3 5978 8623
Email:	sales@survey-design.com.au
URL:	http://survey-design.com.au
Countries served:	Australia, New Zealand

Company:	Timberlake Consultants
Address:	Unit B3 Broomsleigh Bus. Park Worsley Bridge Road LONDON SE26 5BN United Kingdom
Phone:	+44 (0)208 697 3377
Fax:	+44 (0)208 697 3388
Email:	info@timberlake.co.uk
URL:	http://www.timberlake.co.uk
Countries served:	United Kingdom, Eire

Company:	Timberlake Consulting S.L.
Address:	Calle Mendez Nunez, 1, 3 41011 Sevilla Spain
Phone:	+34 (9) 5 422 0648
Fax:	+34 (9) 5 422 0648
Email:	timberlake@zoom.es
Countries served:	Spain

Company:	Timberlake Consultores, Lda.
Address:	Praceta Raúl Brandao, n°1, 1°E 2720 ALFRAGIDE Portugal
Phone:	+351 (0)1 471 73 47
Fax:	+351 (0)1 471 73 47
Email:	timberlake.co@mail.telepac.pt
Countries served:	Portugal

Company:	Unidost A.S.
	Rihtim Cad. Polat Han D:38
	Kadikoy
	81320 ISTANBUL
	Turkey
Phone:	+90 (216) 414 19 58
Fax:	+30 (216) 336 89 23
Email:	info@unidost.com
URL:	http://abone.turk.net/unidost
Countries served:	Turkey

Company:	Vishvas Marketing-Mix Services
Address:	C\O S. D. Wamorkar "Prashant" Vishnu Nagar, Naupada THANE - 400602 India
Phone:	+91-251-440087
Fax:	+91-22-5378552
Email:	vishvas@vsnl.com
Countries served:	India