STATA TECHNICAL BULLETIN

January 1996

STB-29

A publication to promote communication among Stata users

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

an57	Stata is on the Web

Chinh Nguyen, webmaster@stata.com

As of February 1, Stata has a site on the World Wide Web. To view our home page, point your browser to `http://www.stata.com`. This site contains useful materials for Stata users as well as potential users. Let us know what you like, dislike, and what you would like to see added.

crc43	Wald test of nonlinear hypotheses after model estimation

The syntax of `testnl` is

`testnl` *eqnamelist* [`, g(`*matname*$_1$`) r(`*matname*$_2$`)`]

`testnl` tests (linear or nonlinear) hypotheses about the estimated parameters from the most recently estimated model. The equations to be tested must be previously defined by `eq`; see [5s] eq.

Options

`g(`*matname*$_1$`)` specifies a matrix name to be created containing $\mathbf{G}$, the matrix of derivatives of $\mathbf{R}(\mathbf{b})$ with respect to $\mathbf{b}$; see *Methods and Formulas* below. This option is intended for programmers needing an internal ingredient of the calculation.

`r(`*matname*$_2$`)` specifies a matrix name to be created containing $\mathbf{R}(\mathbf{b}) - \mathbf{q}$; see *Methods and Formulas* below. This option is intended for programmers needing an internal ingredient of the calculation.

Remarks

There are three steps to using `testnl`: first, estimate a model using any of Stata's estimation commands (`regress`, `fit`, `logistic`, etc.); second, define the equation(s) to be tested using `eq`; finally, perform the test using `testnl`. For example,

```
. regress y x1 x2 x3 x4
(output omitted)
. eq one: _b[x1]/_b[x2] = _b[x3]
. testnl one
     one:  _b[x1]/_b[x2] = _b[x3]
               F(1, 69) =        0.01
               Prob > F =        0.9322

. eq two: _b[x4] = _b[x1]
. testnl one two
     one:  _b[x1]/_b[x2] = _b[x3]
     two:  _b[x4] = _b[x1]
               F(2, 69) =        3.40
               Prob > F =        0.0392
```

`testnl` reports the constraints being tested followed by an F or χ^2 test, as appropriate. `testnl` is **not** restricted to being used solely after linear regression; it can be used after any estimation command:

```
. logit outcome x1 x2 x3 x4
. testnl one two
     one:  _b[x1]/_b[x2] = _b[x3]
     two:  _b[x4] = _b[x1]
                chi2(2) =        2.66
              Prob > chi2 =        0.2648
```

Using testnl to perform linear tests

`testnl` may be used to test linear constraints, but `test` (see [5s] test) is faster. For instance, in the above example, if you wanted to test constraint `two` by itself, you could type

```
. testnl two
```

but it would take less computer time if you typed

```
. test _b[x4] = _b[x1]
```

Specifying constraints

You specify the constraints to be tested using `eq`. Although `eq` has various syntaxes, the best for storing hypotheses is

`eq` *eqname*`:` *exp* `=` *exp*

You may double the equals sign if you wish:

`eq` *eqname*`:` *exp* `==` *exp*

The algebraic form in which you specify the constraint does not matter; you could type

```
. eq mult:  _b[mpg]*_b[weight] = 1
```

or

```
. eq mult:  _b[mpg] = 1/_b[weight]
```

or you could express the constraint any other way you wished.

You must, however, exercise one caution: Users of `test` often refer to the coefficient on a variable by specifying the variable name, e.g.,

```
. regress price weight mpg
. test mpg = 0
```

More formally, they should type

```
. test _b[mpg] = 0
```

but `test` allows the `_b[]` surrounding the variable name to be omitted. `testnl` does **not** allow this shorthand. Typing

```
. eq zero: mpg=0
```

specifies the constraint that the value of variable `mpg` in the first observation is zero. If you make this mistake, in some cases `testnl` will catch it:

```
. testnl zero
eq zero:  contains reference to X rather than _b[X]
r(198);
```

In other cases `testnl` may not catch the mistake; in that case, the constraint will be dropped without explanation because it does not make sense:

```
. testnl zero
  zero:  mpg=0
         Constraint zero dropped
```

Note that there are other reasons constraints may be dropped; see *Dropped constraints* below.

The worst case, however, is

```
. eq mult: _b[weight]*mpg = 1
```

when what you mean is not that `_b[weight]` equals the reciprocal of the value of `mpg` in the first observation, but rather

```
. eq mult: _b[weight]*_b[mpg] = 1
```

Sometimes this mistake will be flagged by the "contains X and not `_b[X]`" error and sometimes not. Be careful.

Use of testnl after multiple-equation estimation commands

`testnl`, like `test`, can be used after any Stata estimation command. When used after a multiple-equation command such as `mlogit` or `heckman`, you refer to coefficients using Stata's standard syntax: `[`*eqname*`]_b[`*varname*`]`.

Stata's single equation estimation output looks like:

```
             |    Coef   ...
-------------+---------- ...
      weight |    12.27  ...    <-- coefficient is _b[weight]
         mpg |     3.21  ...
-------------------------- ...
```

Stata's multiple equation output looks like:

```
          |    Coef   ...
----------+---------- ...
cat1      |           ...
   weight |    12.27  ...    <-- coefficient is [cat1]_b[weight]
      mpg |     3.21  ...
----------+---------- ...
8         |           ...
   weight |     5.83  ...    <-- coefficient is [8]_b[weight]
      mpg |     7.43  ...
--------------------- ...
```

Dropped constraints

`testnl` automatically drops constraints when

1. They are nonbinding, e.g., `_b[mpg]=_b[mpg]`. More subtle cases include

   ```
   _b[mpg]*_b[weight] = 4
   _b[weight] = 2
   _b[mpg] = 2
   ```

   In this example, the 3rd constraint is nonbinding since it is implied by the first two.

2. They are contradictory, e.g., `_b[mpg]=2` and `_b[mpg]=3`. More subtle cases include

   ```
   _b[mpg]*_b[weight] = 4
   _b[weight] = 2
   _b[mpg] = 3
   ```

   The 3rd constraint contradicts the first two.

Saved Results

`testnl` saves in the global `S_#` macros:

`$S_3`	test (numerator) degrees of freedom
`$S_5`	denominator degrees of freedom (F) or `.` (χ^2)
`$S_6`	F or χ^2 statistic

Methods and Formulas

You have estimated a model. Define $\mathbf{b}$ as resulting the $1 \times k$ parameter vector and $\mathbf{V}$ as the $k \times k$ covariance matrix. The (linear or nonlinear) hypothesis is given by $\mathbf{R}(\mathbf{b}) = \mathbf{q}$, where $\mathbf{R}$ is a function returning a $j \times 1$ vector. The Wald test formula is (Greene 1993, p. 336)

$$W = (\mathbf{R}(\mathbf{b}) - \mathbf{q})'[\mathbf{G}\mathbf{V}\mathbf{G}']^{-1}(\mathbf{R}(\mathbf{b}) - \mathbf{q})$$

where $\mathbf{G} = \partial\mathbf{R}(\mathbf{b})/\partial\mathbf{b}$ is the derivative matrix of $\mathbf{R}(\mathbf{b})$ with respect to $\mathbf{b}$.

W is distributed χ^2 with j degrees of freedom if $\mathbf{V}$ is an asymptotic covariance matrix.

$F = W/j$ is distributed F with j numerator and $n - k$ denominator degrees of freedom in the case of linear regression.

References

Greene, W. H. 1993. *Econometric Analysis.* 2d ed. New York: Macmillan.

dm27.1	Correction to improved collapse

William Gould, Stata Corp., FAX 409-696-4601

A typographical error in `coll2.ado` produced the error message "`x not found`" when the maximum (`max`) of a variable was requested. This error is now fixed.

dm37	Extended merge capabilities

Jon Faust, Board of Governors of the Federal Reserve System, faustj@frb.gov

Stata's `merge` command can merge only two data sets at a time. This insert presents `xmerge` and `xmerged`, two programs that overcome this limitation. `xmerge` match-merges two *or more* Stata data sets, while `xmerged` match-merges two or more Stata dictionary files.

The syntax of these commands is

{ `xmerge` | `xmerged` } *varlist* `using` *file-list*

For `xmerge`, the *file-list* refers to `.dta` files; for `xmerge`, it refers to `.dct` files.

In addition to merging more than two files at a time, `xmerge` and `xmerged` have some other differences from `merge`.

1. `xmerge` and `xmerged` destroy the current data set without warning.
2. `xmerge` and `xmerged` perform only match-merges.
3. The current data set is not included in the merge.
4. `_merge` is not preserved.

Example

```
. set obs 5
obs was 0, now 5
. generate int index = 900 + _n
. generate one = _n
. sort index
. list
       index        one
  1.     901          1
  2.     902          2
  3.     903          3
  4.     904          4
  5.     905          5
. save one
file one.dta saved
. outfile using one, dictionary
. drop one
. generate int two = 2*_n
. save two
file two.dta saved
. outfile using two, dictionary
. drop two
. generate int three = 3*_n
. save three
file three.dta saved
. outfile using three, dictionary
. drop _all
. xmerge index using one two three
. list
       index        one       two     three
  1.     901          1         2         3
  2.     902          2         4         6
  3.     903          3         6         9
  4.     904          4         8        12
  5.     905          5        10        15
. drop _all
. xmerged index using one two three
```

```
. list
       index        one        two      three
  1.     901          1          2          3
  2.     902          2          4          6
  3.     903          3          6          9
  4.     904          4          8         12
  5.     905          5         10         15
```

dm38 A more automated merge procedure

Robert M. Farmer, Alabama Quality Assurance Foundation, Inc., 205-970-1600

In addition to providing a wide range of statistical commands, Stata also provides a full set of data management commands. Perhaps the most powerful of these is `merge`, which combines the current data set (the *master* data set) with a data set on disk (the *using* data set). Commonly, the data sets are merged based on the values of a matching variable that determines which observations from each data set are joined.

The user must complete several steps to perform a successful match-merge in Stata. First, the user must sort both data sets by the matching variable. In addition, the user must make sure that neither data set contains `_merge`, the diagnostic variable created by the `merge` command. Finally, the user must `use` the master data set and type

```
. merge mergevar using mergefile
```

where *mergevar* is the matching variable and *mergefile* is the using data set.

This process is fraught with potential errors:

1. Both the master and the using data sets must contain the matching variable. No provision is made for data sets that contain matching variables with different names.
2. Both data sets set must be sorted by the matching variable.
3. The variable `_merge` cannot exist in either data set.
4. The resulting merged data set must fit in the current memory partition.

This insert presents `mergein`, a program that automates much of the process of merging data sets. `mergein` guarantees that both the master and using data sets are sorted correctly and that neither set contains a variable called `_merge`. In addition, `mergein` permits the matching variable to have a different name in each data set.

Syntax

`mergein` *mergevar mergefile* [*mergevar-2*]

mergevar is the matching variable in the master data set. If *mergevar-2* is omitted, the matching variable in the using data set is assumed to be called *mergevar*, consistent with the behavior of Stata's `merge` command. *mergefile* is the filename (with path, if necessary) of the using data set.

`mergein` has some known problems. On data sets with small numbers of variables or observations, `mergein` will sometimes fail. In these cases, the original master data set is left in place. Also, merging a small master data set with a much larger using data set can cause problems. `mergein` is more robust when the larger data set is the master.

Examples

```
. set obs 10
obs was 0, now 10
. generate int onedex = 900 + _n
. generate int one = _n
. list
       onedex       one
  1.     901          1
  2.     902          2
  3.     903          3
  4.     904          4
  5.     905          5
  6.     906          6
  7.     907          7
  8.     908          8
```

```
  9.      909         9
 10.      910        10

. save one
file one.dta saved

. drop _all

. set obs 5
obs was 0, now 5

. generate int twodex = 900 + _n

. generate int two = 2 * _n

. save two
file two.dta saved

. drop _all

. set obs 25
obs was 0, now 25

. generate int thrdex = 900 + _n

. generate int three = 3 * _n

. save three
file three.dta saved

. use one

. mergein onedex two twodex

. list
       onedex       one       two    _merge
  1.      901         1         2         3
  2.      902         2         4         3
  3.      903         3         6         3
  4.      904         4         8         3
  5.      905         5        10         3
  6.      906         6         .         1
  7.      907         7         .         1
  8.      908         8         .         1
  9.      909         9         .         1
 10.      910        10         .         1

. use one, clear

. mergein onedex three thrdex

. list
       onedex       one     three    _merge
  1.      901         1         3         3
  2.      902         2         6         3
  3.      903         3         9         3
  4.      904         4        12         3
  5.      905         5        15         3
  6.      906         6        18         3
  7.      907         7        21         3
  8.      908         8        24         3
  9.      909         9        27         3
 10.      910        10        30         3
 11.      911         .        33         2
 12.      912         .        36         2
 13.      913         .        39         2
 14.      914         .        42         2
 15.      915         .        45         2
 16.      916         .        48         2
 17.      917         .        51         2
 18.      918         .        54         2
 19.      919         .        57         2
 20.      920         .        60         2
 21.      921         .        63         2
 22.      922         .        66         2
 23.      923         .        69         2
 24.      924         .        72         2
 25.      925         .        75         2
```

dm39 Using .hlp files to document data analysis

Michael Hills, London School of Hygiene and Tropical Medicine, London, mhills@lshtm.ac.uk

As every data analyst knows, without documentation the details of exactly what happened during the analysis fade beyond recall within a few months. Anything which makes the rather tedious chore of documentation easier is welcome. The use of help files is suggested as an additional tool for documentation which can be used along with standard Stata tools such as labeling variables and values, using `describe`, keeping log files, and using `dtainfo` (Schmidt 1995).

Stata users will be familiar with `help` which pulls in the documentation for a command and displays it on the screen. This information is in a file called `name.hlp`, where `name` is the name of the command. The `.hlp` file is usually in same directory as the corresponding `.ado` file, but can be anywhere in the adopath. The file contains text with the additional convention that text enclosed by hat signs, as in `^text^`, is highlighted in the display. To use this facility for documenting data and analysis you simply create documentation files with the extension `.hlp`. These can then be pulled in and displayed at any time during a Stata session using `help`, and the `edit` command can be used to edit them. The convenient place to put these files is in the current working directory, along with the files they document. An example of a simple help file for the data file `diet.dta`, used in teaching, follows.

```
^Diet and heart disease^

These data arose from a pilot study of the use of a weighed diet over
7 days in epidemiological studies. The data in ^diet^ relate subsequent
incidence of ischemic heart disease (IHD) to dietary energy intake.
The data are unpublished - further details about the study are given in
^Morris JN et al, British Medical Journal, 19 Nov 1977, 2, 1307-1314^

^id^          identity number
^agein^       age at entry
^y^           observation time in years
^d^           1=ischemic heart disease, 0 otherwise
^job^         0=driver 1=conductor 2=bank
^month^       month when weighed dietary survey took place
^loweng^      1=total energy less than 2750 kcals, 0 otherwise
^toteng^      total energy (kcals/day)
^height^      height(cms)
^weight^      weight(kgs)
^htgroup^     grouped height with cutpoints min/165/170/175/180/max
            coded 1, 2, 3, 4, 5.
```

For a large project I find it convenient to have a help file for the project which lists all of the data files involved, with a short description of each. For each data file I have a help file which starts with a description of what is in the file, and then lists the variable names (highlighted) together with how they are coded. Opening a log file and using `describe` is a good way of starting this help file. Full details about coding can then be added, and further comments can be included as time passes. Finally I have a help file which documents the various `.do` files which are used to carry out the analysis. Keeping all documentation in the computer in this way has advantages—it means that you can check on coding details at any time, and document as you go along.

Reference

Schmidt, T. J. 1995. dm35: A utility for surveying Stata format data sets. *Stata Technical Bulletin* 28: 7–9.

dm40 Converting string variables to numeric variables

Robert M. Farmer, Alabama Quality Assurance Foundation, Inc., 205-970-1600

Very often, numeric variables obtained from an outside source or another program will be stored as strings. Stata can read these values into string variables, but they must be converted into numeric variables before they can be used in any calculations.

Stata provides the `real()` function for this purpose. However, this function has a couple of inconvenient features. First, `real()` cannot determine the most economical datatype for storing a converted string. Second, `real()` does not recognize some of the suffixes that other programs, such as spreadsheets, commonly attach to numbers. Thus, strings like "`12.5%`" and "72d" are converted to missing values by `real()`.

This insert presents `conv2num`, a utility program that makes string-to-numeric conversion more convenient. The syntax is

`conv2num` *string-var* [, `nocompress` `generate(`*numeric-var*`)` `label(`*variable-label*`)`]

If only the *string-var* is specified, it is converted to a numeric variable in place. Any completely nonnumeric values are converted to missing. `conv2num` converts *string-var* to the most economical datatype possible without losing precision unless the `nocompress` option is specified. If the *numeric-var* is also specified, a new variable with that name is generated to hold the numeric values and the *string-var* is left as is. By default, the *numeric-var* is given the same variable label as the *string-var*. However, if a `label()` is specified, it is used instead.

`conv2num` uses the `real()` function to perform the conversions, hence any precision limitations of `real()` are also limitations of `conv2num`.

Examples

```
. use example
. describe
Contains data from example.dta
  Obs:     6 (max= 71392)
 Vars:     1 (max=    99)                     14 Jan 1996 20:08
Width:    12 (max=   200)
  1. strvar       str12  %12s                 String containing numbers
Sorted by:
. list
           strvar
  1.        12.5%
  2.           27
  3.      -5.0e-2
  4. 0.3333333333
  5.   1234567.89
  6. hello, world
. conv2num strvar, generate(numeric) label("Numeric variable")
. describe
Contains data from example.dta
  Obs:     6 (max= 71386)
 Vars:     2 (max=    99)                     14 Jan 1996 20:08
Width:    20 (max=   200)
  1. strvar       str12  %12s                 String containing numbers
  2. numeric      double %10.0g               Numeric variable
Sorted by:
Note:  Data has changed since last save
. list
           strvar     numeric
  1.        12.5%        12.5
  2.           27          27
  3.      -5.0e-2        -.05
  4. 0.3333333333   .33333333
  5.   1234567.89   1234567.9
  6. hello, world           .
. generate str12 newstr = string(int(10*numeric))
. conv2num newstr, generate(newnum)
. describe
Contains data from example.dta
  Obs:     6 (max= 71386)
 Vars:     4 (max=    99)                     14 Jan 1996 20:08
Width:    36 (max=   200)
  1. strvar       str12  %12s                 String containing numbers
  2. numeric      double %10.0g               Numeric variable
  3. newstr       str12  %12s
  4. newnum       long   %10.0g
Sorted by:
Note:  Data has changed since last save
. list
           strvar     numeric        newstr      newnum
  1.        12.5%        12.5           125         125
  2.           27          27           270         270
  3.      -5.0e-2        -.05             0           0
  4. 0.3333333333   .33333333             3           3
  5.   1234567.89   1234567.9      1.23e+07    12300000
  6. hello, world           .             .           .
```

gr18	Graphing high-dimensional data using parallel coordinates

John R. Gleason, Syracuse University, 73241.717@compuserve.com

Effective methods for visualizing high-dimensional data sets are among the cornerstones of modern data analysis. Many graphical tools have been devised for this task, an example of which is the widely used scatterplot matrix that Stata's `graph` command offers as an option. A scatterplot matrix is easy to understand and often reveals many interesting features of a data set, but no single tool is likely to be best for all data analysis problems. This insert describes a command that implements an alternative visualization tool, the *parallel coordinates plot*, a display that may be useful as an alternative or supplement to the familiar scatterplot matrix. (For the sake of brevity, the phrase *parallel coordinates* will often be replaced with the acronym *ParC*.)

A scatterplot represents the values of two variables (say, y_1 and y_2) as positions along the axes of the Cartesian coordinate system, plotting each observation with some symbol (e.g., a dot). The key idea of a parallel coordinates plot is to rotate one axis so that the y_1- and y_2-axes are parallel rather than perpendicular to each other. An individual observation consists of a position on each axis, and is plotted as the line segment connecting those two positions. Let us pause, briefly, to demonstrate these two approaches to graphing a set of bivariate observations.

Cartesian and parallel coordinates in the bivariate case

Consider a data set used by Campbell (1989) to locate bushfire scars. The raw data consist of satellite measurements on five frequency bands for each of 38 pixels, and appear in Table 4 of Maronna and Yohai (1995), as well as in the file `bushfire.dta` included with this insert.

```
. use bushfire
(Bushfire Scars)
. describe
Contains data from bushfire.dta
  Obs:    38 (max= 10531)                    Bushfire Scars
 Vars:     7 (max=    99)                    29 Oct 1995 14:57
Width:    12 (max=   204)
  1. f1           int    %8.0g               Frequency 1
  2. f2           int    %8.0g               Frequency 2
  3. f3           int    %8.0g               Frequency 3
  4. f4           int    %8.0g               Frequency 4
  5. f5           int    %8.0g               Frequency 5
  6. pixel        byte   %8.0g               Pixel No.
  7. cluster      byte   %9.0g       pix     Pixel cluster
Sorted by:
```

A no-frills scatterplot of `f1` versus `f2` is created by the `graph` command below; the corresponding ParC plot is created by the `parcoord` command below. The resulting graphs are shown in Figures 1 and 2.

```
. graph f1 f2, symbol(o)
(graph appears, see Figure 1)
. parcoord f1 f2
(graph appears, see Figure 2)
```

In Figure 1, each observation is represented by a small circle, and few readers will fail to detect a strong correlation between `f1` and `f2`. (In fact, the Pearson $r = .80$.) In Figure 2, the same data are represented as 38 line segments and, given the relative novelty of this presentation, a bit more effort is required to recognize that `f1` and `f2` are positively correlated. But a strong positive correlation means that `f1` is roughly a linear re-expression of `f2`. Given that one may arbitrarily set the location and scale for the axes of any graph, this means that position along the `f1` axis should strongly resemble position along the `f2` axis. That is, a positive correlation should result in many line segments that are nearly parallel to each other; Figure 2 has just that appearance. (Similarly, a strong negative correlation should result in many lines that cross each other in an X-shaped manner.)

This might seem an arcane way to view a bivariate distribution, though this is partly a matter of greater familiarity with the scatterplot. But extending the scatterplot to $p > 2$ variables is problematic, while extending the ParC plot is straightforward. A scatterplot matrix for p variables is not, in fact, a p-dimensional scatterplot, but a clever arrangement of many 2-dimensional scatterplots. To increase the dimensionality of a ParC plot, however, additional variables are merely represented as further axes positioned parallel to the original axis pair. The main limitation on the number of variables that can be portrayed is a familiar practical one—the resolution of the output device. Further, each observation in a ParC plot is encoded by a set of line segments joined end-to-end, an object that might in other contexts be called a *profile*. An advantage is that the viewer can easily track sets of observations across the variables plotted—a kind of visual cluster analysis. It is rather more difficult, even with clever use of plot symbols and colors, to simultaneously track more than a few observations across the panels of a scatterplot matrix.

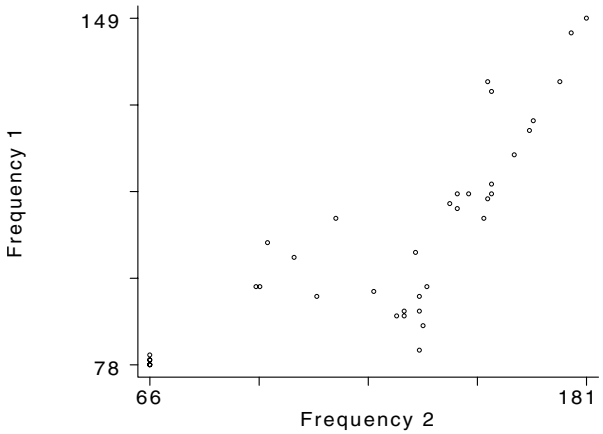

Figure 1

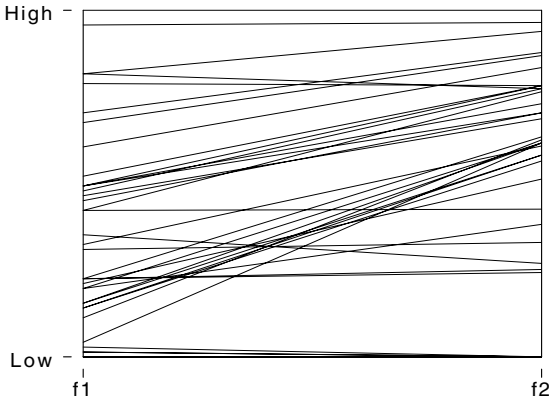

Figure 2

Inselberg (1985) devised the ParC representation of multivariate data for use in computational geometry. Wegman (1990) introduced the ParC plot as a visual tool for statistical analysis with high-dimensional data sets. These authors also explored a number of geometric and statistical properties of data represented in the ParC system. The interested reader is strongly encouraged to consult these and related references for additional information about the ParC representation. The remainder of this insert focuses more narrowly, presenting some examples of ParC plots, examining their structures, and describing a Stata command (`parcoord.ado`) that creates such plot.

A command for parallel coordinates plots

The syntax of `parcoord` is

`parcoord` *varlist* [`if` *exp*] [`in` *range*] [`,` `by(`*byvar*`)` `center` `colorby(`*idvar*`)` `echo` `tour` *graph_options*]

`parcoord` uses Stata's `graph` command to draw a ParC plot in which the variables in `varlist` are represented as parallel vertical axes of identical length, arranged left-to-right across the plot. *varlist* must be present and may not contain string variables. An observation will be ignored if it has missing values for any of the *varlist* variables, the *byvar* variable, or the *idvar* variable; *if* and *in* clauses may also be used to select subsets of observations. The term *graph_options* stands for certain of the options allowed with `graph, twoway`; see Remark 6, below, for details. The other options are explained in the sections that follow.

As an example, using `bushfire.dta`, draw a scatterplot matrix and a ParC plot for the variables `f1-f5`:

```
. graph f1-f5, matrix half
```
(graph appears, see Figure 3)
```
. parcoord f1-f5
```
(graph appears, see Figure 4)

Examining Figure 3, it is clear that variables `f4` and `f5` are very highly correlated ($r = .999$), as are `f3` and `f4` ($r = .974$) and `f3` and `f5` ($r = .976$). These high correlations manifest themselves in Figure 4 as nearly parallel line segments. By contrast, the predominance of intersecting line segments connecting the `f2` and `f3` axes is a sign of negative correlation ($r = -.525$). But, closer examination of the `f2` and `f3` axes points to a more interesting feature; in particular, much of the negative correlation is induced by a small subset of observations with very low `f2` values and very high `f3` values. (Without those observations, $r = +.259$.) Now, track those observations outward away from the `f2-f3` region, and it becomes clear that they form a cluster distinctly different from the remaining observations. In this cluster, `f1` and `f2` have low values, while the `f3-f5` values are the highest in the data set. Scanning from left to right, observations in this cluster follow a path that is nearly a mirror image of the path followed by the remainder of the data set. It is considerably more difficult to detect this kind of behavior in Figure 3, particularly if the presence of such a cluster has not already been signaled in some other way.

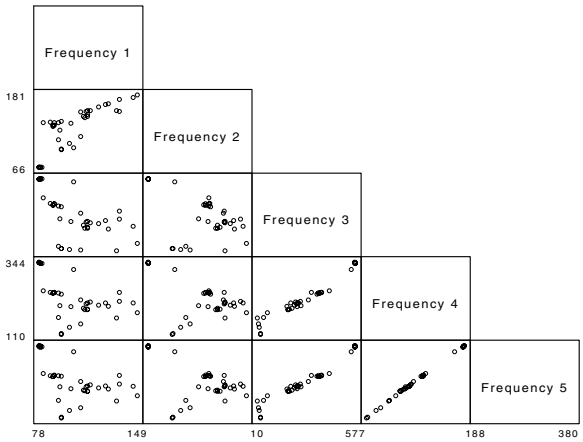

Figure 3

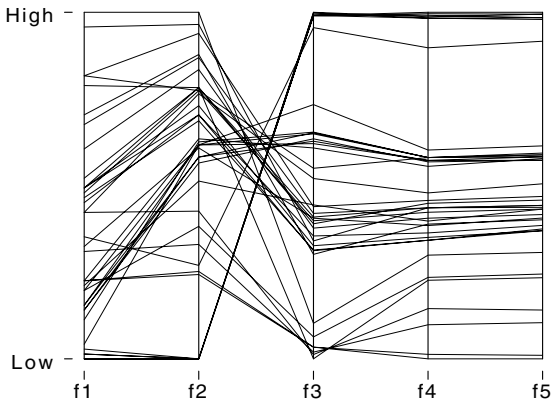

Figure 4

Portraying subsets of observations in parallel coordinates

`parcoord` has two options to aid in examining interesting subsets of observations. The `colorby(`*idvar*`)` option assigns different pens to the values of *idvar*; on color monitors, this draws observations at different values of *idvar* in different colors. To illustrate, Maronna and Yohai (1995) suggested that observations 8–9 and 32–38 represent two distinct types of outliers. These subsets are encoded by the variable `cluster` in `bushfire.dta`:

```
. tab cluster
       Pixel|
     cluster|      Freq.     Percent        Cum.
------------+-----------------------------------
     Others |         29       76.32       76.32
        8-9 |          2        5.26       81.58
      32-38 |          7       18.42      100.00
------------+-----------------------------------
      Total |         38      100.00
```

Then, the ParC plot in Figure 4 can be created with these three subsets drawn in different colors using

```
. parcoord f1-f5, colorby(cluster)
(graph appears)
```

Color-coding is a very effective way to study high-dimensional data, though this is difficult to demonstrate in a monochrome medium. But the `by(`*byvar*`)` option draws a separate ParC plot for each value of *byvar*. For example:

```
. parcoord f1-f5, by(cluster) total
```

(graph appears; see Figure 5)

Thus, the option `colorby(cluster)` produces a single plot resembling the sub-plot labeled `Total` in Figure 5, but with the observations in each of the other sub-plots of Figure 5 drawn in distinct colors. Notice that observations 32–38, in the lower left sub-plot of Figure 5, form the cluster seen earlier in Figure 4.

Permuting the axes of a ParC plot

Unlike a scatterplot matrix, a ParC plot is strongly affected by the order in which the variables are arranged. For example, if the axes of Figure 4 are drawn in the order `f5 f1 f4 f2 f3`, the very high correlations among `f3`, `f4`, and `f5` will be more difficult to discern. For this reason, it is usually important to draw several versions of a ParC plot, varying the left-right order of the axes as required. By default, `parcoord` orders the axes of the plot to match the order of the variables in the *varlist*, so that different views of a data set can be created by repeatedly invoking `parcoord`. However, this would often be inefficient, because of the number of possible permutations of the *varlist*, and because of the (redundant) processing at each invocation to prepare the data set for plotting.

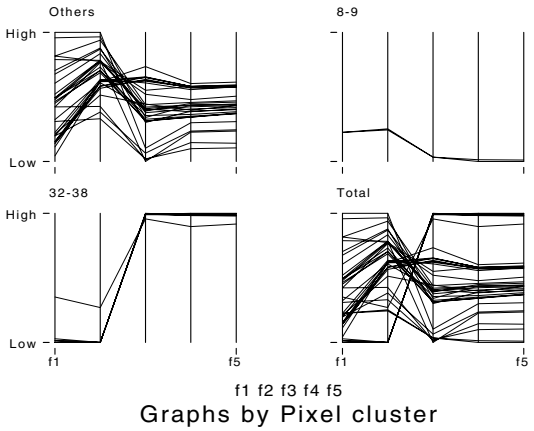

Figure 5

Wegman (1990) showed that each of the $p(p-1)/2$ possible pairs of p variables can be plotted as adjacent axes using just $\lfloor (p+1)/2 \rfloor$ permutations of the variables. The `tour` option initiates Wegman's efficient tour of the possible reorderings of the *varlist*. For example, with the data in `bushfire.dta`,

```
. parcoord f1-f5, tour
(a sequence of graphs appears)
```

produces a sequence of ParC plots, the first three of which present all 10 pairs of the five variables as adjacent axes. The tour pauses at each plot, displays `--more--`, and waits for the user to press a key; the tour then continues, endlessly, until the user presses the *Break* key. However, the tour is largely redundant after the first $\lfloor (p+1)/2 \rfloor$ plots: the next $\lfloor (p+1)/2 \rfloor$ plots are minor variations (usually, left-right reflections) of the initial set of plots, and the $(p+1)$-st plot is identical to the first plot on the tour.

Scaling the axes of a ParC plot

Conventionally, axes in a ParC plot have identical lengths and orientations (upward movements correspond to increases in each variable). By default, `parcoord` scales each axis so that the endpoints are the minimum and maximum values of the variable represented. The ParC plots of Figures 1–3 have been scaled in this way. The `center` option provides another scaling: The axes are drawn so that the median of each variable coincides with the axis midpoint, the length is set so that all observations fit on the axis, and a reference line is drawn at the median.

To illustrate, using the familiar automobile data (`auto.dta`),

```
. parcoord price-gratio, tour center by(foreign) total
(a sequence of graphs appears)
```

begins a tour in which all 45 pairs of the 10 variables appear on adjacent axes during the first five ParC plots; separate plots are drawn for domestic and foreign autos. Figure 6 shows the sixth stop on this tour. Notice that, on each of the leftmost six variables, nearly the entire group of 22 foreign autos falls at or below the overall median of the combined set of 74 autos. It can also be seen from the `Total` panel of Figure 6 that the variables `price` and `mpg` have a strong positive skew; such aspects are more difficult to detect with the default axis scaling.

Remarks

1. In some respects, the ParC plot uses space more efficiently than does a scatterplot matrix. Figure 6 presents three ParC plots for $p = 10$ variables, with enough detail to be usable. The corresponding set of scatterplot matrices will contain 135 scatterplots which, if drawn in the same space, will be too small to be useful.

2. On the other hand, space for labeling the axes in a ParC plot is very limited, and `parcoord` works hard to make the most of that space. If the `by` option is not present, the variable names are drawn at the bottoms of the axes if $p < 10$; for $p \geq 10$, the names are drawn alternately at the bottoms and tops of successive axes. This strategy generally prevents variable names from overwriting each other for $p \leq 18$ or so. Beyond this point, it may be necessary to reset Stata's `textsize` parameter to keep the variable names readable. As a rough guideline, setting `textsize` to $\lfloor 1800/p \rfloor$ if $p > 18$ will generally avoid overlap in the axis names.

3. If the `by` option is present, `parcoord` abandons any attempt to label every axis, and instead labels only the first and last axis in each plot. The full set of axis names—abbreviated, if necessary—is then written as a title at the bottom of the plot. Figures 5 and 6 provide examples. Also, the mapping of colors to values of *idvar* is not identified when both `by` and `colorby` are requested.

4. If the axis names are difficult to read, the `echo` option may be useful. `echo` prints the current axis ordering to the screen before drawing the plot. In windowed versions of Stata, one can alternate between viewing the ParC plot in a *Graph* window and the axis names in the *Results* window. In addition, it is possible to capture the list of axis names (in a log file, perhaps) and, after minor editing, issue a `parcoord` command to reconstruct a particular plot observed during the Wegman tour.

5. The `parcoord` options may be given in any combination. If `by(`*byvar*`)` and `colorby(`*idvar*`)` are both present, *byvar* and *idvar* may or may not be the same variable. In any case, *idvar* may not have more than 20 distinct values; this is a limitation of the `graph` command.

6. The following options of the `graph` command are set by `parcoord` and may not be altered by the user: `by`, `b2title`, `connect`, `noaxis`, `pen`, `symbol`, `tlabel`, `xlabel`, `xline`, `ylabel`, `yline`. Other graph options (especially `saving`) may be supplied, though many of them would not be helpful.

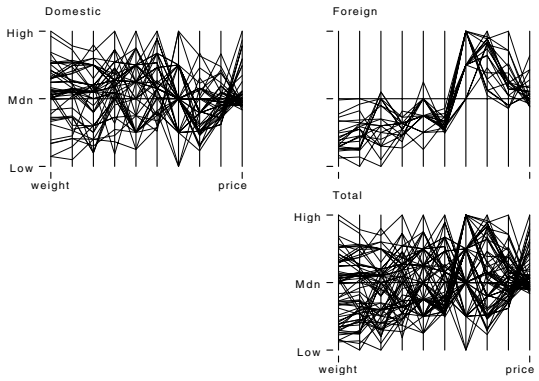

Figure 6

References

Campbell, N. A. 1989. Bushfire mapping using NOAA AVHRR data. Technical Report, CSIRO.

Inselberg, A. 1985. The plane with parallel coordinates. *The Visual Computer* 1: 69–91.

Maronna, R. and Yohai, V. J. 1995. The behavior of the Stahel-Donoho robust multivariate estimator. *Journal of the American Statistical Association* 90: 330–341.

Wegman, E. J. 1990. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association* 85: 664–675.

gr19	Misleading or confusing boxplots

John C. Nash, Faculty of Administration, University of Ottawa
jcnash@aix1.uottawa.ca

Boxplots provide a convenient summary of univariate distributional properties. This note points out that the scaling of data and/or missing values may give rise to misleading or confusing plots. Keen observations by Laszlo Engleman of Systat were the motivations for these notes.

Background

As part of an analysis of data relating to the performance of three optimization algorithms (Nash and Nash 1994; based on Nash and Nocedal 1991), various performance information was recorded for three programs as they minimized twenty large test functions (in 200 to 10000 variables) from prescribed starting points. Table 1 shows the data recorded for two of these programs, labeled CG and TN, for the number of function/gradient evaluations required by each of the problems. The problems are labeled by letters of the alphabet from A to T. Note that we do not have data for the CG code for problem F where the program failed; this program also reached an evaluation limit for problem A. How such important matters should be dealt with in the performance analysis is outside the present discussion.

Table 1 Partial data for algorithm performance analysis

psize	cgfg	tnfg	pname	
200	9999	929	A	
200	2491	456	B	
200	6847	599	C	
500	4889	3446	D	
1000	36	58	E	
1000	.	200	F	← note missing value
1000	456	75	G	
500	98	54	H	
200	14	20	I	
1000	573	208	J	
961	519	387	K	
10000	33	111	L	
1000	56	75	M	
1000	302	160	N	
10000	76	118	O	
1000	615	68	P	
1000	591	210	Q	
1000	91	370	R	
10000	13	19	S	
403	144	100	T	

Graphs, transformations, and missing values

Because the problem sizes are quite different, the count data has a very wide range. A comparative plot, as in Figure 1, has most of the points in the lower left corner, and the marginal boxplots and one-way distributional plots have data compressed into small regions.

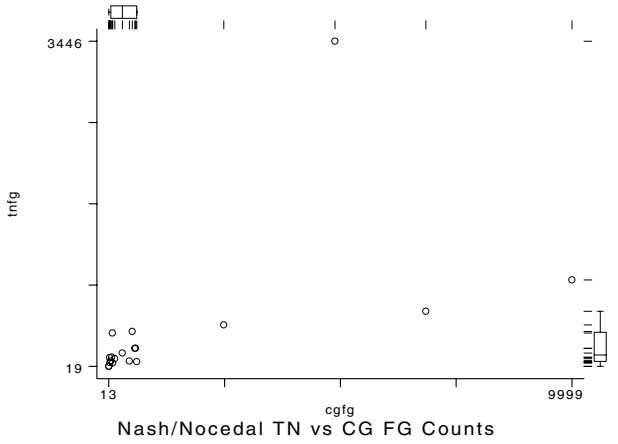

Figure 1. Unscaled data

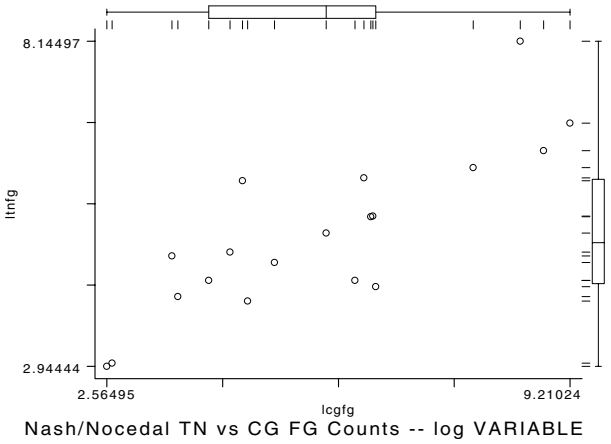

Figure 2. Log-transformed data

A logarithmic transformation of the variables gives the plot in Figure 2. Note that now we have the data nicely distributed over the graph. Points no longer coalesce in the one-way plots. The boxplots show the relative positions of the median and hinges. However, a count of the points on the one-way plot shows some points have still coalesced. Moreover, the missing value in CG implies there are only 19 data points, so the median must correspond to one of the points, which it does. For the TN data, however, the median does *not* correspond to an observation. The boxplot has, in fact, been drawn using all available data. This

is clear from Figure 3, which presents boxplots of logarithmically transformed data for the TN program where we artificially set the appropriate observation to missing in the lower plot. The medians are highlighted with arrows in Figure 3.

More troubling in its initial appearance is Figure 4, where the marginal boxplot at the top appears to have lost its whisker. This graph has been drawn using the original data, but the axes have been logarithmically scaled. In fact, the box at the top of Figure 1 has simply been stretched out. The resulting graph can be considered correct, but requires an unusual interpretation and so may be the cause of confusion.

Artificial examples of axis scaling

We can also use artificial data. As an example, 100 uniformly spaced values 0.05, 0.10, . . . , 5.0 were generated (`uniform`) along with their squares (`square`), square roots (`sqroot`) and exponentials (`exp`). Figure 5 shows the graphs of these drawn with no axis scaling, log scaling of the y-axis, and both x and y-axes log-scaled. Note that the axes are scaled, but the axis labeling refers to the original data scale. A spline summary of the data has been drawn and this overlays the points. Of course, when there is a modest volume of data, we could simply plot the points and not draw connecting lines or a spline summary. Since boxplots are a mechanism to bring out the distributional shape for a set of data, and use distances in the original scale to adjust the *whisker*, *outlier* and *far out* points, we should clearly not carry out transformations of the data scale. However, such transformations may be invoked inadvertently or automatically by statistical software. We can avoid misinterpretation, even with high volumes of data, by noting (as in the examples)

- that the axis ticks tell the reader that a log or similar scale has been used;
- that boxplots and one-way (distributional) plots show how the shape of the two-way graphs may be supported by many or few data points. For example, relatively few points are involved in the lower-left part of the graph of `sqroot` versus `uniform` when both axes are log-scaled.

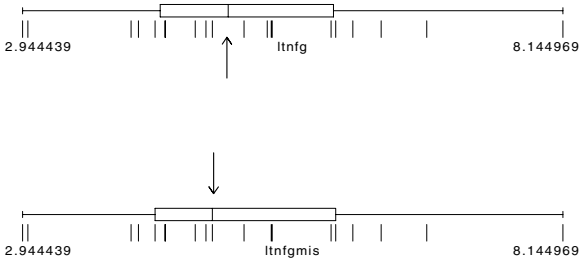

Figure 3. Effect of a missing value

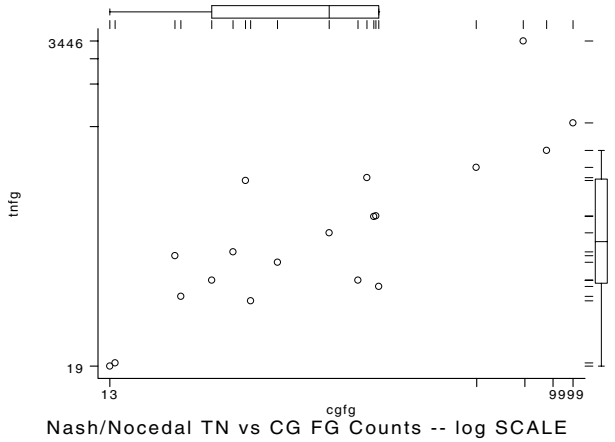

Figure 4. Unscaled data, log-scaled axes

Conclusion

Graphs can, as always, convey misleading impressions. The exercise above points out specific concerns when logarithmic scales are used and/or there is missing data, especially if graphics are combined so that different elements may use or not use all the data, or may be transformed or rendered in ways that could have unintended interpretations.

Acknowledgments

Laszlo Engleman of Systat noted the apparently anomalous boxplots in Figure 4 that led to this note. All the graphs here were prepared with Stata, but no criticism of this product should be drawn from the present discussion. Indeed, the ease with which Stata prepared the graphs allowed this investigation.

References

Nash J. C. and Nash M. M. 1994. Scientific computing with PCs. available by anonymous FTP from ftp.synapse.net/private/n/nis/ or macnash.admin.uottawa.ca (there is a license fee for downloading the entire book, but a table of contents, preface and sample chapter are free).

Nash, S. G. and Nocedal, J. 1991. A numerical study of the limited memory BFGS method and the truncated-Newton method for large-scale optimization, *SIAM Journal of Optimization* 1: 358–372.

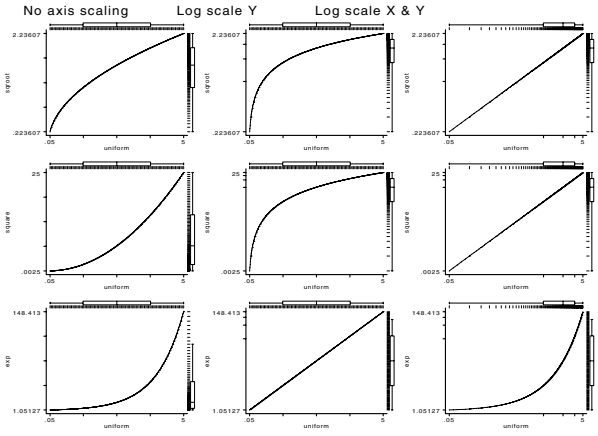

Figure 5. Different axis scales, artificial data

ip11 A tool for manipulating S_# objects

John R. Gleason, Syracuse University, 73241.717@compuserve.com

This insert describes a command (s_no) that displays or erases Stata macros, scalars, or matrices with names of the form S_1, S_2, etc. s_no is principally a tool for programmers, as ordinary Stata users will rarely have need to directly manipulate Stata objects in this way.

Stata programs often make their arguments and results available for use by other programs. Traditionally, this has been accomplished by storing important objects in the S_# macros. However, Stata now has scalars and matrices so that numerical objects can, and perhaps should, be saved in S_# scalars and matrices. In any case, the programming guidelines offered in [4] program_fragments suggest that programs should store in objects named S_#, where # is a small positive integer; often this will result in objects with consecutive names, for example, S_1, S_2, etc. When developing Stata programs, it can be helpful to display S_# objects, perhaps to verify that a program saves results in the intended way. Similarly, it may be useful to erase all S_# objects before testing a program, to guarantee that when those objects are next examined, their contents will have been deposited by the program just invoked.

The command s_no displays or drops objects (macros, scalars, or matrices) named according to Stata's S_# convention. This combines and extends the abilities of the command disp_s (which displays S_# macros) and the undocumented command zap_s (which drops S_# scalars and matrices). The syntax is

s_no [, drop gap(*gcount*) high(*hcount*) numeric]

By default, s_no displays the S_# macros S_1, S_2, etc.; the options are explained in the text that follows.

S_# macros

To illustrate, suppose that no S_# macros are currently defined, as would be true when Stata is launched. As will be explained shortly, this situation can also be created (almost surely) by the command

```
. s_no, drop gap(10)
```

Then, create contents for the following S_# macros.

```
. global S_1 "Here´s S_1"
. global S_2 "and S_2"
. global S_3 "and then S_3"
. global S_5 "skip past S_4"
. global S_8 "as well as S_6 and S_7"
```

The default response of s_no is to display some of those contents:

```
. s_no
S_1:       Here´s S_1
S_2:       and S_2
S_3:       and then S_3
```

This is similar to giving the command disp_s, with an important exception: disp_s checks the first 30 S_# macros, S_1, S_2, . . . , S_30, displaying those that happen to exist. s_no follows a different strategy: it displays S_# macros until it encounters a gap or break in the numbering of the macros. In our example, S_4 does not exist, so that s_no encounters a gap after displaying S_3 and thus exits. The size of the gap that causes s_no to halt is controlled by the gap(*gcount*) option. The default value of *gcount* is 1, but gap(2) requires a gap of size 2 to exit:

```
. s_no, gap(2)
S_1:       Here´s S_1
S_2:       and S_2
S_3:       and then S_3
S_5:       skip past S_4
```

Choosing a sufficiently large value for *gcount* will tolerate large gaps and thus force the display of all currently defined S_# macros:

```
. s_no, gap(10)
S_1:       Here´s S_1
S_2:       and S_2
S_3:       and then S_3
S_5:       skip past S_4
S_8:       as well as S_6 and S_7
```

The high(*hcount)* option restricts attention to S_# objects where # is no greater than *hcount*. The default value of *hcount* is 32766 so that, as just observed, supplying a large value of gcount will normally find all of the S_# macros. On the other hand, combining a large value for *gcount* with a given value of *hcount* finds all S_# objects in a certain range. Continuing our example,

```
. s_no, gap(10) hi(5)
S_1:       Here´s S_1
S_2:       and S_2
S_3:       and then S_3
S_5:       skip past S_4
```

tolerates gaps of size 9 but exits at S_5, and thus does not display macro S_8, while s_no, gap(10) hi(10) would show all currently defined macros in the range S_1. . .S_10.

The option drop causes s_no to silently drop rather than display S_# objects; the gap and high options function as before. Thus,

```
. s_no, drop
```

drops S_# macros, beginning at S_1 until a gap of size 1 is encountered. In our running example, S_1, S_2, and S_3, but not S_5 or S_8, would be dropped:

```
. s_no, gap(5)
S_5:       skip past S_4
S_8:       as well as S_6 and S_7
```

The command s_no, gap(10) hi(10) drop removes all macros in the range S_1. . .S_10, which includes all of the S_# macros defined in this example.

S_# scalars and matrices

For reasons of accuracy and speed, a programmer should often prefer to pass numerical results as scalars or matrices rather than as macros (see [6a] scalars). Stata's scalars and matrices share a common name space so that, in a sense, together they form a single class of numerical objects. The option `numeric` causes `s_no` to operate on objects in that class named according to the `S_#` convention. The other options continue to function as described above so that, for example,

```
. s_no, num
```

lists all currently defined `S_#` scalars and matrices with consecutive names, and

```
. s_no, num drop gap(3) high(10)
```

drops `S_#` scalars and matrices from memory, stopping at the first gap of size 3, or at scalar or matrix `S_10`, whichever occurs first. The latter command differs from the undocumented `zap_s` only in that `zap_s` attempts to drop each of the numerical objects `S_1`...`S_30`.

ip12 Parsing tokens in Stata

Sean Becketti, Stata Technical Bulletin, stb@stata.com

This insert presents `xparse` and `readtok`, two programs that build on and extend Stata's low-level `parse` command.

Background

Stata's `parse` command is the secret to Stata's extensibility ([6a] parse). High-level parsing with `parse` makes it easy to write ado-files that are indistinguishable from internal Stata commands. In high-level parsing, the program author specifies the complete syntax in just a few lines by creating local macros that define the type of variable list permitted or required, whether `in` or `if` clauses are allowed, and so on. Then the `parse` command handles all the drudgery of parsing and error-checking. By forcing the program author to conform to Stata's syntax, the high-level parsing mode of `parse` encourages good programming practices and style.

`parse` also performs low-level parsing, although this feature is used less frequently by novice Stata programmers. In low-level mode, `parse` splits a string into separate tokens. The user specifies a list of parsing characters, that is, characters that mark the boundaries between tokens.

Low-level parsing is used most commonly to process lists of user-specified options. For instance, I might write a program `xyzzy` with the syntax

`xyzzy` *varlist* [`if` *exp*] [`in` *range*] [, `list`(#[,#[,...]])]

where `list()` contains a list of one or more numbers used by `xyzzy`. This program might begin with the following parsing code:

```
program define xyzzy
        version 4.0
        local varlist "required existing"
        local if "optional prefix"
        local in "optional prefix"
        local options "List(str)"
        parse "`*´"
```

Since the number of items specified in the `list` option can vary, the argument of the option is left as a general string. The next step is to use low-level parsing to break this string into separate tokens in order to test them for validity (are they all numbers), to count them (are there too many, not enough), and to apply them in the body of the program. In the mythical `xyzzy` program, the tokens in the `list` option are separated by commas (and, optionally, spaces). Thus, the program might continue as follows:

```
        if "`list´"=="" {
                noi di in red "You must specify a list of numbers"
                error 98
        }
        parse "`list´", parse(" ,")
```

Low-level parsing in Stata does *not* remove the parsing characters. Instead, it leaves them behind as space-delimited tokens. As a consequence, `xyzzy` must contain logic to ignore commas when the `list()` option is processed.

Normally, this characteristic of `parse` is a minor nuisance. However, Stata's conventions for separating arguments in lists are sometimes ambiguous. In some commands and options, arguments are separated by spaces. In others, arguments are separated by commas, even when spaces would be sufficient to distinguish the arguments. For example, according to the syntax diagram, the following is a legal use of `xyzzy`:

```
. xyzzy price weight if foreign, list(1,2,3,5,99)
```

Note that the meaning of this statement could be conveyed just as well by typing

```
. xyzzy price weight if foreign, list(1 2 3 5 99)
```

To my mind, it is poor design to force the user to remember whether commas or spaces are required to separate the numbers in `list`. A lazy programmer will require the arguments to be separated by spaces, since that design will simplify parsing the option. I prefer to make the program smarter in order to allow the user to separate arguments with either commas or spaces. I wrote `xparse` to simplify this task.

`xparse`: Excluding parsing characters

`xparse` splits a string into tokens and removes the parsing characters. The following, somewhat unusual data set is used to illustrate `xparse` and `readtok`.

```
. use example
. describe
Contains data from example.dta
  Obs:     4 (max= 14460)
 Vars:     1 (max=   500)                     15 Jan 1996 12:14
Width:    44 (max=  1002)
  1. line         str44  %44s
Sorted by:
. list
                                          line
  1.            27 | 15 | California | unemployed
  2.          18:27501:sbeckett:mach21:zkdjewiodj
  3.                              |Hello|:|:|world|
  4.                    The quick brown fox jumped
```

This file contains lines with tokens separated by colons and vertical bars. Stata's `parse` command will not remove these characters. `xparse` will.

```
. local l = line[1]
. parse "`l´", parse(":|")
. local n : word count `*´
. display "`n´ tokens: `*´"
7 tokens: 27  | 15  | California  | unemployed
. xparse, parse(":|") string("`l´")
. local n : word count $S_1
. display "`n´ tokens: $S_1"
4 tokens:  27 15 California unemployed
```

`readtok`: Breaking a string variable into token variables

`xparse` breaks a string into tokens and stores the list of tokens in the global macro `S_1`, separated by spaces. `readtok` operates on a string variable, breaking each observation into tokens, and generating new variables to hold the individual tokens.

```
. readtok line, field(:|) prefix(v)
. describe
Contains data from example.dta
  Obs:     4 (max= 14458)
 Vars:     6 (max=   500)                     15 Jan 1996 12:14
Width:    84 (max=  1002)
  1. line         str44  %44s
  2. v1           str5   %9s
```

```
  3. v2           str5   %9s
  4. v3           str10  %10s
  5. v4           str10  %10s
  6. v5           str10  %10s
Sorted by:
Note:  Data has changed since last save
. list v*
           v1         v2          v3          v4          v5
  1.       27         15  California  unemployed
  2.       18      27501    sbeckett      mach21  zkdjewiodj
  3.    Hello      world
  4.      The      quick       brown         fox      jumped
```

Syntax

xparse , parse(*str*) string(*str*)

readtok *string-variable* [, field(*str*) prefix(*str*)]

The parse() option in xparse specifies the parsing characters, while the string() option specifies the string to be parsed. The field() option in readtok specifies the field separators (that is, parsing characters). Spaces are the default field separators. The prefix() option specifies a prefix for the new variables. The default prefix is _v, and the suffixes are the integers from 1 to the largest number of tokens found.

sg29.1	Tabulation of observed/expected ratios and confidence intervals (Update)

Peter Sasieni, Imperial Cancer Research Fund, London, FAX (011)-44-171-269 3429

This insert updates *sg29*, my earlier article on standardized mortality ratios. SMRs are commonly used in epidemiology to summarize the results of cohort studies. Observed incidences of a particular condition in a cohort are compared to expected incidences to see whether they are unusually large or small. Under standard assumptions, the total observed count, O, is a Poisson random variable with mean E, the expected count. E is assumed to be known without error. The SMR is just $100 \times O/E$.

This insert replaces my original program, smr, with an improved version called smrby. This new program has a slightly different syntax and improved formatting. smr calculated a separate SMR for each observation. smrby allows the user to collapse the data according to the levels of a variable specified in the by option.

Consider, for instance, an example used in the previous insert. In this example, we had data on the observed and expected numbers of cancers at different sites in a cohort who previously had melanoma. The data consisted of one observation per site. Suppose, however, that we had recorded many observations per site. There might have been a separate observation for each combination of age-at-diagnosis, year-of-diagnosis, time-since-melanoma, and so on. We could have handled this situation with collapse and the old smr, but the new smrby permits us to complete the analysis more conveniently, without using collapse.

Syntax

smrby *obsvar expvar* [if *exp*] [in *range*] [, by(*byvar*) hetero level(#) ordinal total trend]

Users of smr will notice that the options icd, rowlab, and sumonly have been eliminated. Instead of icd or rowlab, use by(). Instead of sumonly, just omit the by() option.

Options

by specifies groups for which the observed/expected ratios are to be calculated separately. These groups are used to label the rows. *byvar* can be either a numeric or a string variable. If it is a noninteger number, it is advisable to read and store the variable as a double rather than as a float to avoid unattractive display. If the by() option is not used, smrby treats all observations as a single group (see total below).

hetero produces the chi-squared test for unequal SMRs (heterogeneity).

ordinal is only effective in conjunction with the trend option (see below).

total specifies that the total observed count together with its expectation, the ratio O/E, and confidence interval should be calculated and displayed in addition to the usual output.

`trend` produces the score test for a linear trend in SMRs against the *byvar*. If the `ordinal` option is also specified, then the test is carried out using the values $1, 2, 3, \ldots$

Examples and discussion

In a recent STB, Clayton and Hills (1995) provided a suite of programs for analyzing follow-up studies. One of their programs, `tabrate`, is also designed to compute SMRs. However they have a different data structure in mind, thus their syntax is rather different. In our notation, their syntax is

`tabrate` *obsvar byvar* [`if` *exp*] [`in` *range*] , `exposure(`*expvar*`)`

[`graph level(`#`) per(`#`) smr trend` *graph-options*]

The syntax is similar to that used for regression-type commands, that is, the outcome variable is followed by the covariates and additional variables such as the offset or censoring are given as options. Note that, by default, `tabrate` assumes the exposure variable contains the number of person-years at risk, so the ratios of *obsvar* over *expvar* will be the rates of the observations. The command has options `graph` and `trend`. When `smr` was written, I wanted to produce SMRs for different diseases in a single cohort of individuals. It would not have been appropriate to perform a trend test or even to graph the ratios against the ICD number. The *byvar* in `tabrate` is considered to be a covariate (they call it *xvar*), such as different levels of exposure, time since exposure, or age groups. In such circumstances, the trend test and graph are certainly appropriate and provide a nice addition to the program. The `trend` option (but not the `graph` option) has been added to `smrby`.

Example 1

The first example uses the same data as in *sg29*, except now we have labeled the values of the variable `icd_f` and named it `"ICD Site"`. The new program uses the value labels in the tabulated display.

```
. use eye
. describe icd_f
 13. icd_f        float  %9.0g       icd      ICD Site
. smrby m_o m_e if icd<155 & icd>145, by(icd_f)
             Observed    Expected                    -- Poisson  Exact --
ICD Site    | Male Obs    Male Exp       O/E (%)     [95% Conf. Interval]
------------+------------------------------------------------------------------
Oesophag    |        2      1.4304       139.8            17      505
Stomach     |       11      8.6345       127.4            64      228
Sm. inte    |        0      0.3177         0.0             0     1161+
Colon       |        9      8.6651       103.9            47      197
Rectum      |       11      7.3252       150.2            75      269
(+)   one-tail, 97.5% confidence interval
```

Example 2

The second example is modified from Clayton and Hills (1995).

```
. use kcal,clear
(Heart disease and diet survey)
. lexis agein d y, gen(ageband) br(40,50,60,70)
26   records start before first break - left censored
392  extra records created
NOTE: Following lexis expansion on agein
the following variables have been updated: agein
. sort height
. generate int ht5 = autocode(height,5,152,192)
(10 missing values generated)
. replace ht5 = ht5 - 4
(719 real changes made)
. generate e_d = y/100
. label variable e_d "Expected no. ihd"
. tabrate d loweng, e(e_d) smr
```

```
table of failures (D), expected failures (E), and SMR´s
  loweng        _D        _E      _SMR    ci_low   ci_high
       0        18      24.8    72.523    45.693   115.108
       1        28      20.3   138.246    95.453   200.224
Chisq test for unequal SMRs =     4.72 (1 df, p =  0.030 )
. tabrate d loweng, e(e_d) smr trend
table of failures (D), expected failures (E), and SMR´s
  loweng        _D        _E      _SMR    ci_low   ci_high
       0        18      24.8    72.523    45.693   115.108
       1        28      20.3   138.246    95.453   200.224
chi-squared for trend      4.62 ( 1 df, p =  0.032 )
. smrby d e_d, by(loweng) trend hetero
              Observed    Expected                   -- Poisson  Exact --
low energy  | ihd deaths  e_d            O/E (%)     [95% Conf. Interval]
------------+------------------------------------------------------------
0           |       18     24.8197        72.5            43      115
1           |       28     20.2537       138.2            92      200
Chi-squared for trend     4.72 ( 1 df, p =  0.030 )
Chisq test for unequal SMRs =     4.72 (1 df, p =  0.030 )
```

Notice how `smrby` displays both the trend test and the test for unequal SMRs. When there are only two groups, these tests should give identical answers. (There is a very minor error in the way that `tabrate` calculates the variance of the test for unequal SMRs.)

The confidence intervals produced by `tabrate` are based on a normal approximation to the Poisson distribution. The lower limit is too high even when the observed number of deaths is relatively large.

Example 3

The final example uses data from Breslow and Day (1987, Table 3.12). Note how the ordinal trend test gives the same statistic as that quoted by Breslow and Day.

```
. use bdp105, clear
. describe
Contains data from bdp105.dta
  Obs:     5 (max= 30415)
 Vars:     4 (max=    99)                     20 Nov 1995 16:43
Width:    11 (max=   200)
  1. age          byte   %9.0g                Age employ
  2. exp          float  %9.0g                Exp. nasal
  3. obs          byte   %9.0g                Obs. No. nasal ca.
  4. Age          str5   %9s
Sorted by:
. smrby obs exp, by(age)
              Observed    Expected                   -- Poisson  Exact --
Age employ  | obs         Exp. nasal     O/E (%)     [95% Conf. Interval]
------------+------------------------------------------------------------
16          |        2      5.3600        37.3             5      135
20          |        9     11.3000        79.6            36      151
25          |       13     12.2600       106.0            56      181
30          |        8      6.3400       126.2            54      249
35          |        8      4.7300       169.1            73      333
. smrby obs exp, by(age) trend ord tot
              Observed    Expected                   -- Poisson  Exact --
Age employ  | obs         Exp. nasal     O/E (%)     [95% Conf. Interval]
------------+------------------------------------------------------------
16          |        2      5.3600        37.3             5      135
20          |        9     11.3000        79.6            36      151
25          |       13     12.2600       106.0            56      181
30          |        8      6.3400       126.2            54      249
35          |        8      4.7300       169.1            73      333
------------+------------------------------------------------------------
Total       |       40     39.9900       100.0            71      136
Chi-squared for trend (coded 1,2,...)    5.20 ( 1 df, p =  0.023 )
. smrby obs exp, by(Age) trend hetero
```

```
              Observed    Expected                    -- Poisson  Exact --
Age         | obs         Exp. nasal     O/E (%)     [95% Conf. Interval]
------------+------------------------------------------------------------
20-24       |        9     11.3000        79.6            36      151
25-29       |       13     12.2600       106.0            56      181
30-34       |        8      6.3400       126.2            54      249
<20         |        2      5.3600        37.3             5      135
>=35        |        8      4.7300       169.1            73      333
Trend test not possible on string variable
Chisq test for unequal SMRs =     5.31 (4 df, p =  0.257 )
```

References

Breslow N. E. and N. E. Day. 1987. *Statistical Methods in Cancer Research: Volume II—The Design and Analysis of Cohort Studies.* Lyon: International Agency for Research on Cancer.

Clayton D. and M. Hills. 1995. ssa7: Analysis of follow-up studies. *Stata Technical Bulletin* 27: 19–26.

Sasieni, P. 1995. sg29: Tabulation of observed/expected ratios and confidence intervals. *Stata Technical Bulletin* 23: 18–20.

sg46 | Huber correction for two-stage least squares estimates

Mead Over, The World Bank, aover@worldbank.org
Dean Jolliffe, The World Bank, djolliffe@worldbank.org
Andrew Foster, University of Pennsylvania, afoster@pop.upenn.edu

In applied microeconometric analysis, instrumental variable estimation by two-stage least squares is frequently used to estimate structural parameters when explanatory variables are endogenous. In Stata's `regress` command, instrumental variable estimation is neatly implemented by placing the list of exogenous variables in parentheses, after the list of independent variables and before the comma that demarcates the options. Because the Stata command for least squares with Huber- (or White-) corrected standard errors residuals, `hreg`, has almost the same syntax as the `regress` command, it is natural to infer that it would also accept a list of instrumental variables in parentheses before the comma as a signal to perform instrumental variable estimation before correcting the standard errors. In fact, `hreg` does not correctly interpret the variables in parentheses. (It simply ignores the parentheses and treats the list of supposed instrumental variables as if they were additional members of the list of independent variables.)

`hreg2sls` is an altered version of `hreg` which does recognize the set of variables in parentheses as a set of instrumental variables. It is identical to `hreg` in all respects except that it allows instrumental variable estimation.

Syntax

`hreg2sls` [*depvar* [*varlist1* [(*varlist2*)]]] [*weight*] [`if` *exp*] [`in` *range*] [, `group`(*varname*) `level`(#) *regress-options*]

Example

A typical use of `hreg2sls` will follow the pattern

`.hreg2sls` y_1 y_2 x_1 x_2 x_3 (x_1 x_2 x_3 z_1 z_2), `group`(*cluster*)

where y_1 and y_2 are endogenous variables, x_1-x_3 are exogenous variables, z_1-z_2 are the excluded instruments, and *cluster* is a variable designating the first stage of a two-stage sample design (for example, the town or city in a household survey). If residuals in the same region are correlated or residual variances differ systematically by region then a 2SLS procedure such as `regress` that assumes homoscedasticity and independence will in general produce inconsistent standard errors.

For another example of `hreg2sls`, consider a slightly modified version of the model used in the Stata manual to describe two-stage least squares estimation.

$$\texttt{hsngval} = \alpha_0 + \alpha_1\texttt{faminc} + \alpha_2\texttt{pcturban} + \epsilon$$
$$\texttt{rent} = \beta_0 + \beta_1\texttt{hsngval} + \beta_2\texttt{pcturban} + \nu$$

`hsngval` is the median value of housing in each state, `rent` is the state-level, median monthly rent, `faminc` is the median value of family income, and `pcturban` is the percentage of the state population living in urban areas. The data are found in the `hsng.dta` file distributed with Stata. The only difference between this example and the one used in the Stata manual is that the region dummy variables have been dropped from the `hsngval` equation. For this example, it is assumed that the inter-region

variation of the residuals is different from the intra-region variation, which results in a heteroscedastic error structure. Use of the `group()` option in `hreg2sls` will correct the estimated standard errors for this form of heteroscedasticity. Below are the two-stage least squares estimates of this model, and then following are the two-stage least squares estimates with Huber-corrected standard errors.

```
. use hsng
(1980 Census housing data)
. regress rent hsngval pcturban (pcturban faminc)
                                                                      (2SLS)
  Source |       SS       df       MS                  Number of obs =      50
---------+------------------------------               F(  2,    47) =   24.18
   Model |  17681.4852     2  8840.74262               Prob > F      =  0.0000
Residual |  43561.6348    47  926.843293               R-squared     =  0.2887
---------+------------------------------               Adj R-squared =  0.2584
   Total |    61243.12    49  1249.85959               Root MSE      =  30.444
------------------------------------------------------------------------------
    rent |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
 hsngval |   .0031938   .0006401      4.990   0.000       .0019062    .0044815
pcturban |  -.5064118   .4966869     -1.020   0.313      -1.505617    .4927933
   _cons |   113.8143   21.17164      5.376   0.000       71.22248    156.4062
------------------------------------------------------------------------------

. hreg2sls ren hsngval pcturban (pcturban faminc), group(region)
(obs=50)
Regression with Huber standard errors (2SLS)        Number of obs    =      50
                                                    R-square         =  0.2887
                                                    Adj R-square     =  0.2584
                                                    Root MSE         = 30.4441
Grouping variable: region
------------------------------------------------------------------------------
    rent |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
 hsngval |   .0031938   .0004785      6.674   0.000       .0022311    .0041565
pcturban |  -.5064118   .7125682     -0.711   0.481      -1.939914    .9270905
   _cons |   113.8143   21.43369      5.310   0.000        70.6953    156.9334
------------------------------------------------------------------------------
```

Methods and Formulas

The Huber variance–covariance matrix for ordinary least squares estimates of β in the linear expression $y_i = \beta' x_i + u_i$ is

$$V(\widehat{\beta}) = \left(\sum_i \frac{x_i x_i'}{n}\right)^{-1} \left(\sum_j \frac{\widehat{u}_j^2 x_j x_j'}{n}\right) \left(\sum_i \frac{x_i x_i'}{n}\right)^{-1} \tag{1}$$

where $\widehat{u}_j$ is the estimated residual for observation j. The formula corresponding to equation (1) for two-stage least squares estimation (White 1984, p. 141) is obtained by replacing the vector x_i in (1) with $\widehat{x}_i$, its predicted value from the first stage regressions. After this correction, we have

$$V(\widetilde{\beta}) = \left(\sum_i \frac{\widehat{x}_i \widehat{x}_i'}{n}\right)^{-1} \left(\sum_j \frac{\widehat{u}_j^2 \widehat{x}_j \widehat{x}_j'}{n}\right) \left(\sum_i \frac{\widehat{x}_i \widehat{x}_i'}{n}\right)^{-1} \tag{2}$$

Extension of these formulae to the case of clustered data is straightforward as illustrated, for equation (1), in the Stata manual ([5s] huber).

`hreg2sls` takes advantage of the similarity between equations (2) and (1) by replacing each of the x variables in the data set by its respective predicted value and then calling Stata's Huber engine, `_huber`. The `preserve` command is used to ensure that the x variables are restored to their original values upon termination of the procedure.

References

Huber, P. J. 1967. The behavior of maximum likelihood estimates under non-standard conditions. *Proceeding of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* 1: 221–233.

White, H. 1984. *Asymptotic Theory for Econometricians*. New York: Academic Press.

sg47	A plot and a test for the χ^2 distribution

Patrick Royston, Royal Postgraduate Medical School, London, FAX (011)-44-181-259-8573

Stata's `qnorm` and `swilk` commands are designed to produce normal probability plots and to test samples for departure from normality. Here I present two programs, `qchi` and `a2`, which perform similar tasks for the χ^2 distribution.

Quantile–quantile plots for χ^2

`qchi` produces a quantile–quantile (Q–Q) plot for the χ^2 distribution. If a variable Y has approximately a χ^2 distribution with ν degrees of freedom, the plot produced by `qchi` will be roughly a straight line. Due to the skewness, there will be a much greater concentration of points in the lower-left quadrant of the diagram than in the upper right. The values in the upper-right quadrant will be highly variable and often will not lie on the "ideal" line, even if Y really does. The slope in the lower-left quadrant indicates whether the degrees of freedom are about right or not. If ν is too small, the slope will be > 1 and the points will form a curve above the line. If ν is too large, the slope will be < 1 and the points will fall mostly below the line.

The syntax of `qchi` is

`qchi` *varname* [`if` *exp*] [`in` *range*] , `df(`#`)` [`transform` *graph_options*]

Options

`df()` is *not* optional; it specifies the degrees of freedom for the χ^2 variable.

`transform` transforms the values on both axes to their cube roots, which transforms a χ^2 variable to approximate normality. Since this transformation suppresses the straggly upper tail, the graph may be easier to interpret.

graph_options are any of the options allowed with `graph, twoway`.

The Anderson–Darling goodness-of-fit test

`a2` performs the Anderson–Darling A^2 goodness-of-fit test for three different distributions: normal, uniform and χ^2. The A^2 test is one of a family of tests based on the empirical distribution function or EDF (see Stephens 1974). A large, significant value of the test statistic indicates departure from the hypothesized distribution. (As always, a non-significant value does *not* prove that the data follow the hypothesized distribution.)

The syntax of `a2` is

`a2` *varlist* [`if` *exp*] [`in` *range*] , `dist(normal | uniform | chisquare)` [`df(`#`)`]

Options

`dist(normal|uniform|chisquare)` is *not* optional; it specifies the assumed distribution to be tested. Only the first letter of the name of the distribution is required.

`df()` is required for `dist(chisquare)`; it specifies the degrees of freedom for the χ^2 distribution.

Example

The Pearson X^2 statistic for association between proportions of observations in the r rows and c columns in a two-way contingency table has a χ^2 distribution with $(r-1)(c-1)$ degrees of freedom in large samples. Here I use the `qchi` and `a2` commands to investigate the χ^2 assumption in 3×3 tables based on four sample sizes: 5, 10, 20, 45.

Traditionally, the χ^2 assumption is taken to be adequate when the cell occupancies exceed five. With nine cells this condition will hold on average for $n = 45$ and above.

For each of the four sample sizes, I simulated 500 random samples with no association between the rows and columns and tabulated the results by using the commands

```
generate row=1+int(3*uniform())
generate col=1+int(3*uniform())
tabulate row col, chi
```

Stata stores the χ^2 statistic in `_result(4)`, so the results for each randomly-generated table were displayed, saved to a log file for further processing and re-read into Stata.

The figure shows plots of the simulated χ^2 statistics for each of the 4 sample sizes. The mean cell occupancy for $n = 5$ is 5/9 and the distribution of X^2 is discrete, which makes the plot appear as a step function. The lines for $n = 10$ and $n = 20$ are curved, indicating departure from a χ^2 distribution, but perhaps less markedly than one might expect given the small sample sizes. The plot for $n = 45$ is close to linear.

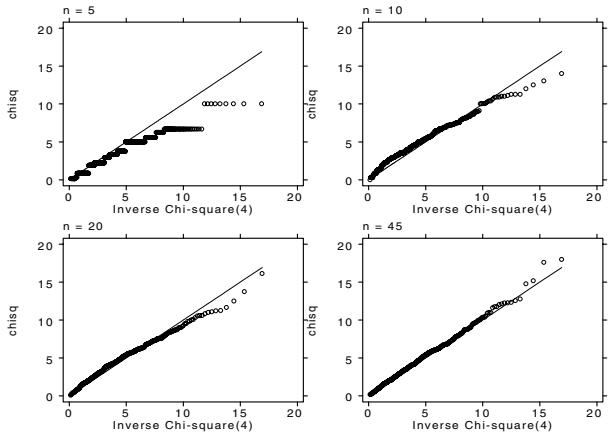

Figure 1

The p-values from the Anderson–Darling A^2 statistic are 0.000, 0.000, 0.001, and 0.094 for $n = 5$, 10, 20, 45, respectively, so only the results for $n = 45$ suggest that the χ^2 assumption is not rejected at the 5% level of significance. Nevertheless, it is clear from the shapes of the graphs that even for $n = 10$ the χ^2 assumption is not badly violated.

For the benefit of readers who wish to experiment further (for example, with rectangular rather than square tables), I have also included on the STB-29 diskette the do-files which do the work. `sim.do` simulates the X^2 statistics, saving to a log-file (`sim.sto`). `sim2.do` reads back the data (suitably edited from `sim.sto`) and carries out the analyses.

References

Stephens, M. A. 1974. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association* 69: 730–737.

sg48 | Making predictions in the original metric for log-transformed models

Richard Goldstein, Qualitas, Inc., richgold@netcom.com

The advice often given, for many relatively simple regression problems, is to transform the dependent variable. If the transformed version makes sense (is interpretable), then this is fine. However, if the transformation is made only for estimation purposes and predicted values are desired in the original metric (for example, dollars), then there can be problems because the obvious retransformation is often not what is wanted. For example, if one transforms the dependent variable by taking logarithms, the simple retransformation of just exponentiating the predicted values will give you a conditional median rather than the expected, and usually desired, conditional mean. If the conditional mean is actually what is wanted, then the median is a biased retransformation.

In several disciplines, ranging from ecology to economics, a different retransformation has been suggested: exponentiate the sum of the predicted value and one half the square of the root mean square error from the regression. (See, for instance, Miller 1984. Note that Miller has similar retransformations for other common transforms, including square root and reciprocal). However, Duan (1983) has shown that in many cases there is a better retransformation.

This insert presents `predlog`, an ado file that calculates all three retransformations. These predictions are added to the data as three new variables: `YHATRAW` (just exponentiated), `YHTNAIVE` (the Miller version) and `YHTSMEAR` (Duan's smearing estimate retransform).

The syntax of `predlog` is

`predlog` *varlist* [`if` *exp*] [`in` *range*] [, `fp`predict]

`predlog` estimates, but does not display, the regression that would be estimated by the command

```
. regress varlist
```

If the `fpredict` option is used, the predictions will be calculated only for the cases used in the regression; otherwise predictions will be calculated for all possible cases. In effect, Stata's `predict` ([5s] predict) command is used if there is no option, and Stata's `fpredict` ([5s] fit) command is used if the user adds the option.

Duan argues that his retransformation is less biased than the naive one if there is any skewness remaining after transforming the dependent variable; this is especially so if the underlying data are a mixture of normals rather than a truly skewed variable. He shows in his paper that the naive retransform is no more efficient than his as long as the square of the RMSE is no more than 0.5. It is not much less efficient even if the square of the RMSE is about 1.0 and not much less efficient in any case where there are at least ten independent variables in the model (see the examples below).

`predlog` does not display the underlying regression, since `predlog` is likely to be used only after one has examined a number of regressions to come up with one best model (or a small number of competitive models).

Examples

The examples use the familiar automobile data. These examples consist of three regressions that use the log of price as the dependent variable. Note the relationship between `YHTNAIVE` and `YHTSMEAR` as the RMSE gets better.

```
. use auto
(1978 Automobile Data)
. generate weightsq = weight*weight
. generate logprice=log(price)
. fit logprice mpg
  Source |       SS       df       MS                  Number of obs =      74
---------+------------------------------               F(  1,    72) =   22.87
   Model |  2.70578153     1  2.70578153               Prob > F      =  0.0000
Residual |  8.51775155    72  .118302105               R-squared     =  0.2411
---------+------------------------------               Adj R-squared =  0.2305
   Total |  11.2235331    73  .153747029               Root MSE      =  .34395
------------------------------------------------------------------------------
logprice |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
     mpg |   -.033277   .0069581     -4.782   0.000      -.0471478   -.0194062
   _cons |   9.349342    .153489     60.912   0.000       9.043367    9.655317
------------------------------------------------------------------------------
. predlog price mpg
. summarize Y* price
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+-----------------------------------------------------
 YHATRAW |      74    5755.228   1022.923   2936.537   7708.038
YHTNAIVE |      74    6105.926   1085.255   3115.476    8177.73
YHTSMEAR |      74    6127.065   1089.012   3126.262   8206.043
   price |      74    6165.257   2949.496       3291      15906
. fit logprice foreign mpg
  Source |       SS       df       MS                  Number of obs =      74
---------+------------------------------               F(  2,    71) =   17.80
   Model |  3.74819416     2  1.87409708               Prob > F      =  0.0000
Residual |  7.47533892    71  .105286464               R-squared     =  0.3340
---------+------------------------------               Adj R-squared =  0.3152
   Total |  11.2235331    73  .153747029               Root MSE      =  .32448
------------------------------------------------------------------------------
logprice |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
 foreign |   .2824445   .0897634      3.147   0.002       .1034612    .4614277
     mpg |  -.0421151   .0071399     -5.899   0.000      -.0563517   -.0278785
   _cons |    9.4536   .1485422     63.643   0.000       9.157415    9.749785
------------------------------------------------------------------------------
```

```
. predlog price mpg foreign
. summarize Y* price
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+--------------------------------------------------------
 YHATRAW |      74    5796.027   1250.718   3008.888   9380.918
YHTNAIVE |      74    6109.323   1318.324   3171.529   9887.989
YHTSMEAR |      74    6131.136   1323.031   3182.853   9923.294
   price |      74    6165.257   2949.496       3291      15906
. fit logprice foreign weight* turn hdroom
  Source |       SS       df       MS                  Number of obs =      74
---------+------------------------------                F(  5,    68) =   24.60
   Model |  7.22811292     5  1.44562258                Prob > F      =  0.0000
Residual |  3.99542016    68  .058756179                R-squared     =  0.6440
---------+------------------------------                Adj R-squared =  0.6178
   Total |  11.2235331    73  .153747029                Root MSE      =   .2424
------------------------------------------------------------------------------
logprice |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
 foreign |   .4285643   .0822997      5.207   0.000       .2643378    .5927907
  weight |  -.0002047   .0002962     -0.691   0.492      -.0007958    .0003864
weightsq |   1.34e-07   4.65e-08      2.891   0.005       4.16e-08    2.27e-07
    turn |  -.0322146   .0131998     -2.441   0.017      -.0585544   -.0058749
  hdroom |   -.080939   .0383812     -2.109   0.039      -.1575276   -.0043505
   _cons |   9.345836   .6135071     15.233   0.000       8.121602    10.57007
------------------------------------------------------------------------------
. testparm weight*
 ( 1)  weight = 0.0
 ( 2)  weightsq = 0.0
       F(  2,    68) =   39.82
            Prob > F =    0.0000
. predlog price foreign weight* turn hdroom
. summarize Y* price
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+--------------------------------------------------------
 YHATRAW |      74    5963.625   2180.224   3036.252   15128.19
YHTNAIVE |      74    6141.424   2245.225   3126.775   15579.22
YHTSMEAR |      74    6134.279   2242.613   3123.137    15561.1
   price |      74    6165.257   2949.496       3291      15906
```

References

Duan, N. 1983. Smearing estimate: a nonparametric retransformation method. *Journal of the American Statistical Association* 78: 605–610.

Miller, D. M. 1984. Reducing transformation bias in curve fitting. *The American Statistician* 38: 124–126.

snp9 | Kornbrot's rank difference test

Richard Goldstein, Qualitas, Inc., richgold@netcom.com

Kornbrot's rank difference test (Kornbrot 1990), used exactly as one would use Wilcoxon's signed-ranks test ([5s] signrank), is an alternative that is useful when one has ordinal data. This insert presents `kornbrot`, a Stata ado-file that calculates Kornbrot's rank difference test.

Wilcoxon's signed-ranks test generally is used in two situations: (1) with continuous data that are, or may be, distributed non-normally, and (2) with ordinal data. However, as Kornbrot points out, "a procedure is meaningful for ordinal data if it gives the same result when applied to the original data, or any strictly monotone transformation of the data." As she points out, and as we illustrate below, Wilcoxon's signed-ranks test fails this criterion (also see Maritz 1985). Kornbrot's rank difference test overcomes this problem with Wilcoxon's test.

The syntax of `kornbrot` is

`kornbrot` *varname* `=` *exp* [`if` *exp*] [`in` *range*]

This syntax is exactly the same as the syntax for `signrank`, Stata's command to perform Wilcoxon's signed-ranks test. Further, the result is in the same format since Stata's `signrank` command is used. Unfortunately, the statement of the hypothesis in the test results uses temporary variables so the names are not interpretable; however, the command shows the names, and you can always use `signrank` also if you want.

Example

This example uses the data from Kornbrot's article.

```
. use kornbrot
. list
          id  placebo    drug  placebo2   drug2
  1.       1      4.6     2.9     13.0    20.5
  2.       2      4.3     2.8     13.8    21.1
  3.       3      6.7    12.0      9.0     5.0
  4.       4      5.8     3.8     10.3    16.0
  5.       5      5.0     5.9     12.0    10.1
  6.       6      4.2     6.5     14.2     9.2
  7.       7      6.0     3.3     10.0    18.0
  8.       8      2.0     2.3     30.0    26.5
  9.       9      2.6     2.1     23.0    29.0
 10.      10     10.0    14.3      6.0     4.2
 11.      11      3.4     2.4     17.7    24.6
 12.      12      7.1    14.0      8.4     4.3
 13.      13      8.6     4.9      7.0    12.2
```

Note that the final two variables are equal to the first occurrence of the variable divided into 60; this transformation turns a time result into a rate result. Following are the results from using `signrank` on the two sets of variables; note the difference in the results (a p-value of .86 versus a p-value of .09!).

```
. signrank placebo=drug
Wilcoxon signed-rank test
    sign |      obs   sum ranks    expected
---------+--------------------------------
positive |        7          43        45.5
negative |        6          48        45.5
    zero |        0           0           0
---------+--------------------------------
     all |       13          91          91
unadjusted variance      204.75
adjustment for ties        0.00
adjustment for zeros       0.00
                     ----------
adjusted variance        204.75
Ho: median of placebo = drug
            z =  -0.175
    Prob > |z| =   0.8613
. signrank placebo2=drug2
Wilcoxon signed-rank test
    sign |      obs   sum ranks    expected
---------+--------------------------------
positive |        6          21        45.5
negative |        7          70        45.5
    zero |        0           0           0
---------+--------------------------------
     all |       13          91          91
unadjusted variance      204.75
adjustment for ties        0.00
adjustment for zeros       0.00
                     ----------
adjusted variance        204.75
Ho: median of placebo2 = drug2
            z =  -1.712
    Prob > |z| =   0.0869
```

Here are the results from using Kornbrot's test on the two sets; now the p-values agree.

```
. kornbrot placebo=drug

Wilcoxon signed-rank test

    sign |      obs   sum ranks    expected
---------+--------------------------------
positive |        6          29        45.5
negative |        7          62        45.5
    zero |        0           0           0
---------+--------------------------------
     all |       13          91          91

unadjusted variance      204.75
adjustment for ties       -1.00
adjustment for zeros       0.00
                         ----------
adjusted variance        203.75

Ho: median of __000038 = __000039
            z =  -1.156
    Prob > |z| =   0.2477

. kornbrot placebo2=drug2

Wilcoxon signed-rank test

    sign |      obs   sum ranks    expected
---------+--------------------------------
positive |        7          62        45.5
negative |        6          29        45.5
    zero |        0           0           0
---------+--------------------------------
     all |       13          91          91

unadjusted variance      204.75
adjustment for ties       -1.00
adjustment for zeros       0.00
                         ----------
adjusted variance        203.75

Ho: median of __00003U = __00003V
            z =   1.156
    Prob > |z| =   0.2477
```

Caveat on p-values

Kornbrot provides extensive tables of p-values, based on simulations, for the Kornbrot test in samples smaller than twenty. These tables appear to be particularly important for samples smaller than eight. However, I have followed Stata by ignoring these adjustments and simply using the normal approximation. If you use tables for the Wilcoxon test itself for small samples, you will be a little conservative. Note that I have also followed Stata in not using a continuity correction, something which Kornbrot strongly advocates.

Reference

Kornbrot, D. E. 1990. The rank difference test: A new and meaningful alternative to the Wilcoxon signed ranks test for ordinal data. *British Journal of Mathematical and Statistical Psychology* 43: 241–264.

Maritz, J. S. 1985. Models and the use of signed rank tests. *Statistics in Medicine* 4: 145–153.

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	data sets	*qs*	questions and suggestions
gr	graphics	*tt*	teaching
in	instruction	*zz*	not elsewhere classified
Statistical Categories:			
sbe	biostatistics & epidemiology	*srd*	robust methods & statistical diagnostics
sed	exploratory data analysis	*ssa*	survival analysis
sg	general statistics	*ssi*	simulation & random numbers
smv	multivariate analysis	*sss*	social science & psychometrics
snp	nonparametric methods	*sts*	time-series, econometrics
sqc	quality control	*sxd*	experimental design
sqv	analysis of qualitative variables	*szz*	not elsewhere classified

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

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:	14/26 Yizreel St., P.O. Box 1169 Nazerath-Ellit 17100, Israel
Phone:	+972 6554254
Fax:	+972 6554254
Email:	sasconsl@actcom.co.il
Countries served:	Israel

Company:	Dittrich & Partner Consulting
Address:	Prinzenstrasse 2 D-42697 Solingen Germany
Phone:	+49 212-3390 99
Fax:	+49 212-3390 90
Email:	available soon
Countries served:	Austria, Germany, Italy

Company:	Metrika Consulting
Address:	Roslagsgatan 15 113 55 Stockholm Sweden
Phone:	+46-708-163128
Fax:	+46-8-6122383
Email:	available soon
Countries served:	Baltic States, Denmark, Finland, Iceland, Norway, Sweden

Company:	Oasis Systems BV
Address:	Lekstraat 4 3433 ZB Nieuwegein The Netherlands
Phone:	+31 30 6066336
Fax:	+31 30 6065844
Countries served:	The Netherlands

Company:	Ritme Informatique
Address:	34 boulevard Haussmann 75009 Paris France
Phone:	+33 1 42 46 00 42
Fax:	+33 1 42 46 00 33
Email:	ritme.inf@applelink.apple.com
Countries served:	Belgium, France, Luxembourg, Switzerland

Company:	Timberlake Consultants
Address:	47 Hartfield Crescent West Wickham Kent BR4 9DW, U.K.
Phone:	+44 181 462 0495
Fax:	+44 181 462 0493
Email:	100412.2603@compuserve.com
Countries served:	Ireland, U.K.

Company:	Timberlake Consultants Satellite Office
Address:	Rua Julião Quintinha, No. 5°–7° Esq. 1500 Lisbon Portugal
Phone:	+351 1 7140009
Fax:	+351 1 7140009
Email:	100412.2603@compuserve.com
Countries served:	Portugal