STATA TECHNICAL BULLETIN

September 1997
STB-39

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.

Contents of this issue — page

stata47 lookup now indexes on-line FAQs

William Gould, Stata Corporation, wgould@stata.com

There are currently about 100 FAQs totaling 300 pages available on Stata's web site and that is growing. FAQ stands for Frequently Asked Question and it is web jargon for pages recording answers to popular questions. As you can calculate for yourself from the statistics, the average FAQ runs about 3 pages. To access the FAQs,

1. point your browser to *http://www.stata.com*;
2. click on *User Support*;
3. click on *FAQs*.

A FAQ is a document that opens with a question and then provides the answer. For instance, one of the FAQs on Stata's web site appears as

What does "completely determined" mean in my logistic regression output?

Title: Interpreting "completely determined" when running logistic
Author: William Sribney, Stata Corporation
Date: July 1996

There are two causes for messages like

```
Note:  4 failures and 0 successes completely determined
```

after the commands `logistic`, `logit`, and `probit`. Let us deal with the most unlikely case first. This case occurs when . . .

This particular FAQ runs about 7 pages.

Finding the FAQ that interests you can be a daunting task. On the web site, we currently categorize FAQs according to

- Questions about Windows
  - Installation
  - Memory allocation
  - Crashes
  - Printing
  - Miscellaneous
- Questions about Macintosh
  - Installation
  - Memory allocation
  - Crashes
  - Miscellaneous
- Questions about Unix
- Questions about Statistics
  - Cross-sectional time-series (panel) data
  - Logistic and probit regression
  - Survival analysis
  - Survey and robust estimators
  - Estimation commands
  - Tests
  - Probability distributions
  - Other commands
- Questions about Programming
- Questions about Data Management
  - Reading and inputting data
  - Combining datasets
  - Memory usage
  - Data manipulation
  - Data reporting
  - Dates
  - Other
  - FAQs concerning releases before Stata 5.0

To make finding the FAQs easier, we have updated the lookup database to include them.

`lookup` is the Stata command that indexes the Stata on-line help, the printed documentation, and now, the FAQs. Windows and Macintosh users can also access `lookup` by pulling down *Help* from the top menu bar.

```
. lookup completely determined

FAQ     . . . . . . Interpreting "completely determined" when running logistic
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Sribney
        7/96    What does completely determined mean in my logistic
                regression output?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions
```

The referenced URL *http://www.stata.com/support/faqs/stat/* will take you directly to the statistics FAQ page; alternatively you could go to *http://www.stata.com*, click on *User Support*, click on *FAQs*, and click on *Statistics*. However you get to the page, from there you click on *Logistic and probit regression* and then you scroll down the sublist until you find the completely-determined question.

It is unlikely that you would ever think to lookup the phrase “completely determined” or that we will remember to index such a phrase, but if you type `lookup logistic` or `lookup logit`, you will find this FAQ.

Listing only the FAQs

Those who use Stata for Windows or Macintosh will find pulling down *Help* from the top menu bar the easier way to access `lookup`. A major advantage is that they can then scroll through the list.

No matter which operating system you use, however, you can use the `lookup` command and restrict what is listed. Typing

```
. lookup logistic
(references to 11 manual entries omitted)
(references to 11 STB inserts omitted)
(references to 8 FAQs omitted)
```

(or pulling down *Help* and typing `logistic`) produces a list of $11 + 11 + 8 = 30$ references. If you want just the FAQs, you can include the `class(faq)` option. Here are the eight FAQs relevant to logistic regression:

```
. lookup logistic, class(faq)

FAQ     . . . . . . .  Interpreting the cut points in ordered probit and logit
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Gould
        4/97    In ordered probit and logit, what are the cut points?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions

FAQ     . . . . . . Interpreting "outcome does not vary" when running logistic
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . P. Lin
        11/96   Why do I get the message, "outcome does not vary" when
                I perform a logistic or logit regression?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions

FAQ     . . . . . . . . . . . . . . . . . . . Conditional logit model and clogit
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Sribney
         3/97   I am running clogit and get the message "Note: multiple
                positive outcomes within groups encountered." Is this
                something that I should worry about or is this a normal
                message?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions

FAQ      Comparison of the nested logit and the constrained multinomial models
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Gould
        8/95    Is there a nested logit program for Stata?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions

FAQ     . . . . . . Interpreting "completely determined" when running logistic
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Sribney
        7/96    What does completely determined mean in my logistic
                regression output?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions

FAQ     . . . . . . . . . . . . A terminology problem:  odds ratio versus odds
        . . . . . . . . . . . . . . . . . . . . . . . . . W. Gould and J. Hardin
        3/96    Why does the manual claim that the odds ratio is constant
                in a logistic regression?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions
```

```
FAQ     . . . . . . . . . . . . . Convergence of maximum-likelihood estimators
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Gould
        11/96   Why does my mlogit take so long to converge?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Logistic and probit regressions
FAQ     . . . . . . . .  The appropriate command for matched case-control data
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Sribney
        3/96    Can I do n:1 matching with the mcc command?
                On-line FAQ at http://www.stata.com/support/faqs/stat/
                Click on Other commands
```

If you wanted to obtain just the relevant STB inserts, you could type

```
. lookup logistic, class(stb)
STB-36  sbe14  . OR´s & ci´s for logistic reg. models with effect modification
        (help effmod if installed) . . . . . . . . . . . . . . . .  J. Garrett
        3/97    STB Reprints Vol 6, pages 104--114
        calculates odds ratios and confidence intervals for logistic
        regression models with significant interaction terms
STB-28  sbe12  . . Using lfit and lroc to evaluate mortality prediction models
        . . . . . . . . . . . . J. M. Tilford, P. K. Roberson, and D. H. Fiser
        11/95   STB Reprints Vol 5, pages 77--81
        discussion of how the new lfit and lroc commands (see crc41)
        can be used to evaluate mortality prediction models
STB-35  sg63 .  Logistic regression: standardized coef. & partial correlations
        (help lstand if installed) . . . . . . . . . . . . . . . . .  J. Hilbe
        1/97    STB Reprints Vol 6, pages 162--163
        command for use after logistic; displays predictor, its coefficient,
        odds ratio, standardized coefficient, partial correlation, and p-value
STB-30  sg50 . . . . . . . . . . . . . .  Graphical assessment of linear trend
        (help lintrend if installed) . . . . . . . . . . . . . . J. M. Garrett
        3/96    STB Reprints Vol 5, pages 152--160
        graphically examines the relationship between the log odds of
        a binary outcome by categories of an ordinal or interval
        independent variable
STB-33  sg42.1 . . . . . . . . . . . . . . . . Extensions to the regpred command
        (help regpred2 if installed) . . . . . . . . . . . . . . . . . . M. Over
        9/96    STB Reprints Vol 6, pages 117--121
        adds four additional options to regpred including the
        capability to perform instrumental variable estimation
STB-26  sg42 . . Plotting predicted values from linear & logistic regr. models
        (help regpred and logpred if installed)  . . . . . . . . .  J. Garrett
        7/95    STB Reprints Vol 5, pages 111--116
        calculate and plot predicted values and 95 percent confidence
        intervals of the predictions for one predictor holding constant
        the other predictors
```

(*remaining output omitted*)

Actually, you can type `lookup logistic, class(stb)` or you can type `lookup logistic, stb`; typing just `stb` is a synonym for `class(stb)`. For the new FAQs, however, you must type `class(faq)` if you wish to restrict the listing.

Updates

When you install the official updates from the STB, the lookup database is also updated. All STB inserts up to and including the previous issue are indexed along with all current FAQs.

stata48	Updated reshape

William Gould, Stata Corporation, wgould@stata.com

Below is all-new documentation on the all-new `reshape` command which has all-new syntax. Before you panic, we hasten to add that the new `reshape` command understands the old command's syntax without you specifying version numbers or anything else. It just works, both the old way and the new.

The new `reshape` command has a number of improvements:

1. The syntax is easier to specify and you need specify less.
2. The syntax allows variable names to be suffixed with numbers—as did the old—and it allows numbers to appear in the middle of variable names, such as `inc80m` and `inc81m`. It allows groups to be indicated by characters as well as numbers, such as `incm` and `incf` for income of males and females or even `minc` and `finc`.

3. The command provides better diagnostics when there are problems—including a new `reshape error` command for use when the data have problems. `reshape error` lists the problem observations.
4. The command checks assumptions about the data more carefully so that if there are problems, you are told ahead of time rather than after you obtain an unexpected result.
5. The code is structured so that the conversion process requires less free memory.
6. The problem `reshape` had with allowing only 10 constant-within-group variables is fixed.
7. All the improvements Weesie (1997) incorporated into his `reshape2` command have been included in the new `reshape`.

Basic syntax

`reshape wide` *varnames*`,` `i(`*varlist*`)` [`j(`*varname* [*values*]`)` *other_options*]

`reshape long` *varnames*`,` `i(`*varlist*`)` [`j(`*varname* [*values*]`)` *other_options*]

`reshape wide`

`reshape long`

`reshape error`

where *values* is *#*[-*#*] [*#*[-*#*] ...] and *other_options* is `string` `atwl(`*chars*`)` and both are seldom specified.

Advanced syntax

`reshape i` *varlist*

`reshape j` *varname* [*#*[`-`*#*] [*#*[`-`*#*] ...]] [`, string`]

`reshape xij` *fvarnames* [`, atwl(`*chars*`)`]

`reshape xi` [*varlist*]

`reshape query`

`reshape`

`reshape wide`

`reshape long`

`reshape error`

`reshape clear`

where *fvarnames* are either *varnames*, *varnames* with `@` characters, or a mix of the two. The `@` character denotes where the *#* (j) suffix appears.

Description of basic syntax

Think of the data as a collection of observations X_{ij}. One such collection might be:

```
           (wide form)                                (long form)

-i-         ------- X_ij --------             -i-  -j-         -X_ij-
id  sex    inc80   inc81   inc82              id   year   sex    inc
------------------------------                ----------------------
 1    0     5000    5500    6000               1     80     0   5000
 2    1     2000    2200    3300               1     81     0   5500
 3    0     3000    2000    1000               1     82     0   6000
                                               2     80     1   2000
                                               2     81     1   2200
                                               2     82     1   3300
                                               3     80     0   3000
                                               3     81     0   2000
                                               3     82     0   1000
```

`reshape` converts data from one form to the other:

```
. reshape long inc, i(id) j(year)       (goes from left-form to right)
. reshape wide inc, i(id) j(year)       (goes from right-form to left)
```

In this example one observation is, at least logically speaking,

```
+-------- in the wide form -------+        +------ in the long ------+
| . list if id==1                 |        | . list if id==1         |
|                                 |        |                         |
|    id  sex  inc80  inc81  inc82 |   OR   |    id  sex  year    inc |
| 1.  1    0   5000   5500   6000 |        | 1.  1    0    80   5000 |
+ --------------------------------+        | 2.  1    0    81   5500 |
                                           | 3.  1    0    82   6000 |
                                           + ------------------------+
```

and you want to think of this single "observation" as X_{ij}.

The i variable denotes the logical observation and is often called the group identifier. i is variable `id` in our data.

j denotes the subobservation and so is often called the subgroup or within-group identifier. j is year in our data, or at least, variable `year` when the data is in the long form. There is no j variable in the wide form. Instead, the `inc` variable is suffixed with the values of j, forming `inc80`, `inc81`, and `inc82`.

That leaves only the variable `sex`, which we did not specify when we typed

```
. reshape long inc, i(id) j(year)
```

or

```
. reshape wide inc, i(id) j(year)
```

Since `sex` is not specified, `sex` is assumed to be constant within i and `reshape` verifies that assumption before converting the data.

Here is an example with two X_{ij} variables with the data in wide form:

```
. list
       id   sex   inc80   inc81   inc82  ue80  ue81  ue82
  1.    1     0    5000    5500    6000     0     1     0
  2.    2     1    2000    2200    3300     1     0     0
  3.    3     0    3000    2000    1000     0     0     1
```

To convert this into the long form, we type

```
. reshape long inc ue, i(id) j(year)
(note:  j = 80 81 82)
Data                                   wide   ->  long
-----------------------------------------------------------------------------
Number of obs.                            3   ->       9
Number of variables                       8   ->       5
j variable (3 values)                         ->   year
xij variables:
                      inc80 inc81 inc82   ->   inc
                         ue80 ue81 ue82   ->   ue
-----------------------------------------------------------------------------
```

Note that there is no variable named `year` in our original, wide dataset. `year` will be a new variable in our long dataset. After conversion, we have

```
. list
       id   year   sex     inc    ue
  1.    1     80     0    5000     0
  2.    1     81     0    5500     1
  3.    1     82     0    6000     0
  4.    2     80     1    2000     1
  5.    2     81     1    2200     0
  6.    2     82     1    3300     0
  7.    3     80     0    3000     0
  8.    3     81     0    2000     0
  9.    3     82     0    1000     1
```

Similarly, if we took this dataset and typed

```
. reshape wide inc ue, i(id) j(year)
(note:  j = 80 81 82)
Data                               wide   ->  long
-----------------------------------------------------------------------------
Number of obs.                        9   ->       3
Number of variables                   5   ->       8
j variable (3 values)              year   ->   (dropped)
xij variables:
                                    inc   ->   inc80 inc81 inc82
                                     ue   ->   ue80 ue81 ue82
-----------------------------------------------------------------------------
```

we would be right back to our original data,

```
. list
        id   inc80  ue80   inc81  ue81   inc82  ue82   sex
  1.     1    5000     0    5500     1    6000     0     0
  2.     2    2000     1    2200     0    3300     0     1
  3.     3    3000     0    2000     0    1000     1     0
```

except for variable order.

Converting from wide to long creates the j (**year**) variable. Converting from long to wide drops the j (**year**) variable.

Mistakes

The following wide data contain a mistake:

```
. list
        id   sex   inc80   inc81   inc82
  1.     1     0    5000    5500    6000
  2.     2     1    2000    2200    3300
  3.     3     0    3000    2000    1000
  4.     2     0    2400    2500    2400
. reshape long inc, i(id) j(year)
(note:  j = 80 81 82)
i=id does not uniquely identify the observations;
there are multiple observations with the same value of id.
Type "reshape error" for a listing of the problem observations.
r(9);
```

The i variable must be unique when the data are in the wide form; we said `i(id)` yet we have two observations for `id` $= 2$. Is person 2 a male or female?

Spotting the error in this small dataset is easy. Regardless of the size of the data, `reshape error` will assist us in spotting the problem. `reshape error` will provide assistance whenever the original error message says “type `reshape error` for a listing of the problem observations”.

```
. reshape error
(note:  j = 80 81 82)
i (id) indicates the top-level grouping such as subject id.
The data are currently in the wide form; there should be a single
observation per i.
2 out of 4 observations have duplicate i values:
        id
  2.     2
  3.     2
(data now sorted by id)
```

It is not a mistake when the i variable is repeated in long data, but the following long data have a similar kind of mistake:

```
. list
```

```
       id   year   sex     inc
  1.    1     80     0    5000
  2.    1     81     0    5500
  3.    1     82     0    6000
  4.    2     80     1    2000
  5.    2     81     1    2200
  6.    2     82     1    3300
  7.    3     80     0    3000
  8.    3     81     0    2000
  9.    3     82     0    1000
 10.    3     82     0    1000

. reshape wide inc, i(id) j(year)
(note:  j = 80 81 82)
year not unique within id;
there are multiple observations at the same year within id.
Type "reshape error" for a listing of the problem observations.
r(9);
```

In the long form, `i(id)` does not have to be unique, but `j(year)` must be unique within `i()`; otherwise, what is the value of `inc` in 1981 for `id` $= 1$? As before, `reshape error` will assist in finding the problem:

```
. reshape error
(note:  j = 80 81 82)

i (id) indicates the top-level grouping such as subject id.
j (year) indicates the subgrouping such as time.
The data are in the long form;  j should be unique within i.

There are multiple observations on the same year within id.

The following 2 out of 10 observations have repeated year values:

       id   year
  9.    3     82
 10.    3     82

(data now sorted by id year)
```

Finally, consider the following (long) data which has no mistake in it:

```
. list
       id   year   sex     inc    ue
  1.    1     80     0    5000     0
  2.    1     81     0    5500     1
  3.    1     82     0    6000     0
  4.    2     80     1    2000     1
  5.    2     81     1    2200     0
  6.    2     82     1    3300     0
  7.    3     80     0    3000     0
  8.    3     81     0    2000     0
  9.    3     82     0    1000     1
```

What if we forget to specify `ue`, a variable that varies within i ?

```
. reshape wide inc, i(id) j(year)
(note:  j = 80 81 82)
ue not constant within id
Type "reshape error" for a listing of the problem observations.
r(9);
```

In this case `reshape` observed that `ue` was not constant within i and so could not restructure the data so that there were single observations on i. Of course, we should have typed `reshape wide inc ue, i(id) j(year)`. We could type `reshape error` if the problem were not obvious.

In summary, there are three cases in which `reshape` will refuse to convert the data:

1. the data are in the wide form and i is not unique;
2. the data are in the long form and j is not unique within i;
3. the data are in the long form and an unmentioned variable is not constant within i.

Other mistakes

There are obviously other mistakes one might make but in such situations `reshape` will probably convert the data and produce a surprising result because there is no way of knowing this is not what you intend.

For instance, consider the following wide data where we forget to mention that variable `ue` varies within `id`:

```
. list
       id   sex   inc80   inc81   inc82  ue80  ue81  ue82
  1.    1     0    5000    5500    6000     0     1     0
  2.    2     1    2000    2200    3300     1     0     0
  3.    3     0    3000    2000    1000     0     0     1

. reshape long inc, i(id) j(year)
(note:  j = 80 81 82)

Data                               wide   ->  long
-----------------------------------------------------------------------------
Number of obs.                        3   ->       9
Number of variables                   8   ->       7
j variable (3 values)                     ->   year
xij variables:
                      inc80 inc81 inc82   ->   inc
-----------------------------------------------------------------------------
```

`reshape` does not complain but here is the result:

```
. list
       id   year   sex  ue80  ue81  ue82     inc
  1.    1     80     0     0     1     0    5000
  2.    1     81     0     0     1     0    5500
  3.    1     82     0     0     1     0    6000
  4.    2     80     1     1     0     0    2000
  5.    2     81     1     1     0     0    2200
  6.    2     82     1     1     0     0    3300
  7.    3     80     0     0     0     1    3000
  8.    3     81     0     0     0     1    2000
  9.    3     82     0     0     0     1    1000
```

We did not state that `ue` varied within i and so the variables `ue80`, `ue81`, and `ue82` were left as-is. There is no real problem here because no information has been lost. In fact, this may be the result we wanted.

Probably, however, we simply forgot to include `ue` among the X_{ij} variables. If you obtain an unanticipated result here is how to undo it:

If you typed '`reshape long` ...' to produce it, type `reshape wide` (without arguments) to undo it.

If you typed '`reshape wide` ...' to produce it, type `reshape long` (without arguments) to undo it.

So we can type `reshape wide` to get back to our original data and then type the `reshape long` command we intended: `reshape long inc ue, i(id) j(year)`.

reshape long and reshape wide without arguments

Whenever you type a `reshape long` or `reshape wide` command with arguments, `reshape` remembers it. Thus you might

```
. reshape long inc ue, i(id) j(year)
```

and work with the data like that. You could then type

```
. reshape wide
```

to convert it back to the wide form. You might do some more work. You could type

```
. reshape long
```

to convert it back to the long form. If you save the data, you can even continue using `reshape wide` and `reshape long` without arguments the next day.

Be a little careful, however. If you create new X_{ij} variables, you must tell **reshape** about them by typing out the full **reshape** command, although no real damage will be done if you forget. If you are converting from long to wide, **reshape** itself will catch your error and refuse to do the conversion. If you are converting from wide to long, **reshape** will convert the data, but the result will be surprising (remember what happened when we forget to mention variable `ue` and ended up with `ue80`, `ue81`, and `ue82` in our long data). You can '`reshape long`' to undo it and then try again.

Missing variables

When converting data from the wide to the long forms, **reshape** does not demand that all the variables exist. Missing variables are treated like variables with missing observations. For instance, consider

```
. list
        id   sex   inc80   inc81   inc82  ue80  ue82
  1.     1     0    5000    5500    6000     0     0
  2.     2     1    2000    2200    3300     1     0
  3.     3     0    3000    2000    1000     0     1
```

Note that variable `ue81` is missing. We can still type

```
. reshape long inc ue, i(id) j(year)
(note:  j = 80 81 82)
(note:  ue81 not found)
Data                               wide   ->  long
-----------------------------------------------------------------------------
Number of obs.                        3   ->       9
Number of variables                   7   ->       5
j variable (3 values)                     ->   year
xij variables:
                      inc80 inc81 inc82   ->   inc
                         ue80 ue81 ue82   ->   ue
-----------------------------------------------------------------------------
```

The result is

```
. list, nodisplay
        id   year   sex     inc     ue
  1.     1     80     0    5000      0
  2.     1     81     0    5500      .
  3.     1     82     0    6000      0
  4.     2     80     1    2000      1
  5.     2     81     1    2200      .
  6.     2     82     1    3300      0
  7.     3     80     0    3000      0
  8.     3     81     0    2000      .
  9.     3     82     0    1000      1
```

Were we to reshape these data back to the wide form by typing `reshape wide inc ue, i(id) j(year)`, the variable `ue81` would be created and it would contain all missing values.

Advanced issues with basic syntax: i()

The syntax is

reshape ..., **i(***i_variable***)** ...   or   **reshape** ..., **i(***i_variables***)** ...

Typically i is one variable—such as subject id—but it could be multiple variables such as hospital id and hospital's patient id.

Examples include

```
reshape ..., i(id)
reshape ..., i(hid pid)
```

Advanced issues with basic syntax: j()

The syntax is

reshape ..., ... **j(***j_variable***)** ...   or   **reshape** ..., ... **j(***j_variable j_values***)** ...

The second syntax defines the j variable and its values. When you leave the values unspecified, `reshape` works them out for itself.

`reshape` never makes a mistake when the data is in long form—when you type `reshape wide`. The values can easily be obtained by tabulating the existing j variable.

`reshape` can make a mistake when your data is in the wide form—when you type `reshape long`—if you have named your variables poorly. Pretend you have the variables `inc80`, `inc81`, and `inc82`, recording income in each of the indicated years, and you have a variable named `inc2`, which is not income but when the area was reincorporated. You type

```
. reshape long inc, i(id) j(year)
```

`reshape` would see the variables `inc2`, `inc80`, `inc81`, and `inc82` and decide that there are four groups: $j = 2$, 80, 81, and 82. The best way to address such problems is to name your variables appropriately. The date of reincorporation would be better named something other than `inc` followed by a number; `reinc` would be a good name.

Alternatively, you can keep the name and specify the j values. To perform the reshape, you would type

```
. reshape long inc, i(id) j(year 80-82)
```

or

```
. reshape long inc, i(id) j(year 80 81 82)
```

The dash notation for ranges can be mixed with individual numbers. `reshape` would understand `80 82-87 89 91-95`.

At the other extreme, you can omit the `j()` option altogether. If you do, `reshape long` will name the j variable `_j` and `reshape wide` will assume the variable is named `_j`.

Advanced issues with basic syntax: X_ij

When specifying the variable names you may include `@` characters to indicate where the numbers go. For instance, in the following data

```
. list
       id   sex  inc80r  inc81r  inc82r  ue80  ue81  ue82
  1.    1     0    5000    5500    6000     0     1     0
  2.    2     1    2000    2200    3300     1     0     0
  3.    3     0    3000    2000    1000     0     0     1
```

you would type

```
. reshape long inc@r ue, i(id) j(year)
(note:  j = 80 81 82)
Data                                   wide   ->  long
-----------------------------------------------------------------------------
Number of obs.                            3   ->       9
Number of variables                       8   ->       5
j variable (3 values)                         ->   year
xij variables:
                   inc80r inc81r inc82r   ->   incr
                         ue80 ue81 ue82   ->   ue
-----------------------------------------------------------------------------
```

At most one `@` character may appear in each name. If no `@` character appears, results are as if the `@` character appeared at the end of the name. So equivalent to the above is

```
. reshape long inc@r ue@, i(id) j(year)
```

Either way, the result of the `reshape long` will be

```
. list
       id   year   sex    incr    ue
  1.    1     80     0    5000     0
  2.    1     81     0    5500     1
  3.    1     82     0    6000     0
(output omitted)
  9.    3     82     0    1000     1
```

That is, `inc@r` specifies variables named `inc#r` in the wide format and `incr` in the long.

The `@` notation may similarly be used for converting long to wide datasets:

```
. reshape wide inc@r ue, i(id) j(year)
(output omitted)
```

Advanced issues with basic syntax: the atwl() option

Option `atwl()` is for use when `@` characters are also specified:

reshape *varnames_with_@*, ... **atwl(***chars***)**

`atwl()` stands for at-when-long. When you specify a name such as `inc@r` or `ue@` in the long form, the name becomes `incr` and `ue`; the `@` character is changed into nothing. `atwl()` lets you change it into something.

If you specify `atwl(X)`, the long-form names would become `incXr` and `ueX`. If you specified `atwl(yr)`, the long-form names would become `incyrr` and `ueyr`.

`atwl()` is seldom specified.

Advanced issues with basic syntax: string identifiers for j()

The `string` option allows j to take on string values:

reshape ..., ... **string**

For instance, consider the following wide data on husbands and wives:

```
. list
          id    kids    incm    incf
  1.     101       0    5000    5500
  2.     102       2    2000    2200
  3.     103       1    3000    2000
```

In this data, `incm` is the income of the man and `incf` the income of the woman. This data can be reshaped into separate observations for males and females by typing

```
. reshape long inc, i(id) j(sex) string
(note:  j = f m)
Data                               wide   ->  long
-----------------------------------------------------------------------------
Number of obs.                        3   ->       6
Number of variables                   4   ->       4
j variable (2 values)                     ->   sex
xij variables:
                              incf incm   ->   inc
-----------------------------------------------------------------------------
```

The `string` option specifies that j will take on nonnumeric values. The result is

```
. list
          id     sex    kids     inc
  1.     101       f       0    5500
  2.     101       m       0    5000
  3.     102       f       2    2200
  4.     102       m       2    2000
  5.     103       f       1    2000
  6.     103       m       1    3000
```

`sex` will be a string variable. Similarly, these data can be converted from long to wide by typing

```
. reshape wide inc, i(id) j(sex) string
(note:  j = f m)
Data                               wide   ->  long
-----------------------------------------------------------------------------
Number of obs.                        6   ->       3
Number of variables                   4   ->       4
j variable (2 values)               sex   ->   (dropped)
xij variables:
                                    inc   ->   incf incm
-----------------------------------------------------------------------------
```

Strings are not limited to being single characters or even of the same length. Had the original data been

```
. list
         id    kids    incmale     incfem
  1.    101       0       5000       5500
  2.    102       2       2000       2200
  3.    103       1       3000       2000
```

then **reshape long inc, i(id) j(sex) string** would have resulted in

```
. list
         id         sex    kids         inc
  1.    101         fem       0        5500
  2.    101        male       0        5000
  3.    102         fem       2        2200
  4.    102        male       2        2000
  5.    103         fem       1        2000
  6.    103        male       1        3000
```

Where the string identifier appears may be specified using the **@** notation. In the following wide data, the **m** and **f** appear as a prefix rather than a suffix:

```
. list
         id    kids    minc    finc
  1.    101       0    5000    5500
  2.    102       2    2000    2200
  3.    103       1    3000    2000
```

The **reshape** command is

```
. reshape long @inc, i(id) j(sex) string
```
(*output omitted*)

The resulting variable in the long format will be named **inc**.

Just as with numbers, strings may be placed in the middle. If the variables are named **incMome** and **incFome**, the **reshape** command would be

```
. reshape long inc@ome, i(id) j(sex) string
```
(*output omitted*)

Be careful with string identifiers because it is easy to be surprised by the result. Consider a person with wide data having variables named **incm**, **incf**, **uem**, **uef**, and **agem**, **agef**. To make the data long, the person types

```
. reshape long inc ue age, i(id) j(sex) string
```

Along with these variables, the person innocently has the variable **agenda**. **reshape** will decide the sexes are **m**, **f**, and **nda**. This would not happen without the **string** option if the variables were named **inc0**, **inc1**, **ue0**, **ue1**, and **age0**, **age1** even with variable **agenda** present in the data.

Advanced issues with basic syntax: second-level nesting

You have data from a household survey of the income of husbands and wives. There are four ways these data might be organized.

They might be organized in the long-long form

```
       hid    sex    year     inc
 1.    107      m      90    4500
 2.    107      f      90    3200
 3.    107      m      91    4600
 4.    107      f      91    4700
```

or in the long-year wide-sex form

```
       hid    year    finc    minc
 1.    107      90    3200    4500
 2.    107      91    4700    4600
```

or in the wide-year long-sex form

```
       hid    sex   inc90   inc91
 1.    107      f    3200    4700
 2.    107      m    4500    4600
```

or the wide-wide form

```
       hid  finc90  minc90  finc91  minc91
 1.    107    3200    4500    4700    4600
```

reshape can convert any of these forms to any other. Converting all the way from the long-long form to the wide-wide form (or from the wide-wide form to the long-long) requires two **reshape** commands.

Starting in the wide-wide form, we can convert to (say) the long-year wide-sex form by typing

```
. reshape wide @inc, i(hid year) j(sex) string
```

This in turn can be converted to the wide-wide form by typing

```
. reshape wide minc finc, i(hid) j(year)
```

We can go back from the wide-wide to the long-long by repeating the two steps in reverse order:

```
. reshape long minc finc, i(hid) j(year)
. reshape long @inc, i(hid year) j(sex) string
```

Pairs of **reshape** commands can result in remarkable dataset transformations. For instance, you can convert

```
      gid   year    rs1    rs2    rs3    sex
 1.    26     90     57     28     30      m
 2.    26     91     52     33     32      m
```

into

```
      gid   visit   rs90   rs91    sex
 1.    26       1     57     52      m
 2.    26       2     28     33      m
 3.    26       3     30     32      m
```

by typing

```
. reshape long rs, id(gid year) j(visit)
. reshape wide rs, id(gid visit) j(year)
```

Description of advanced syntax

The advanced syntax is simply a different way of specifying the **reshape** command and it has one seldom-used feature providing a little extra control. Rather than typing a single **reshape** command to describe the data and perform the conversion, such as

```
. reshape long inc, i(id) j(year)
```

you type a sequence of `reshape` commands. The initial commands describe the data and the last performs the conversion:

```
. reshape i id
. reshape j year
. reshape xij inc
. reshape long
```

`reshape i` corresponds to `i()` in the basic syntax.

`reshape j` corresponds to `j()` in the basic syntax.

`reshape xij` corresponds to the variables specified in the basic syntax.

In addition, there is one more specification which has no counterpart in the basic syntax:

`reshape xi` *varlist*

In the basic syntax, all unspecified variables are assumed to be constant within i. The advanced syntax works the same way unless you specify the `reshape xi` command. `reshape xi` names the constant-within-i variables. If you specify `reshape xi`, then any variables that are not explicitly specified are dropped from the data during the conversion.

As a practical matter, it would probably be better if you explicitly dropped the unwanted variables before conversion. For instance, imagine the data have variables `inc80`, `inc81`, `inc82`, `sex`, `age`, and `age2` and that you no longer want the `age2` variable. You could specify

```
. reshape xi sex age
```

or you could

```
. drop age2
```

and leave `reshape xi` unspecified. `reshape xi` does have one minor advantage: It saves `reshape` from going to the work of determining which variables are unspecified. This saves a relatively small amount of computer time.

Another advanced-syntax feature is `reshape query`—equivalent to typing `reshape` by itself. `reshape query` presents a report on what has been defined:

```
. reshape
 (data is wide)
+------------------------------------------------------------------------------+
| Xij                           | Command/contents                             |
+------------------------------------------------------------------------------+
| Subscript i,j definitions:    |                                              |
|  group id variable(s)         | reshape i hid                                |
|  within-group variable        | reshape j year                               |
|   and its range               |                                              |
|                               |                                              |
| Variable X definitions:       |                                              |
|  varying within group         | reshape xij minc finc                        |
|  constant within group (opt)  | reshape xi  <varlist>                        |
+------------------------------------------------------------------------------+
Optionally type "reshape xi" to define variables that are constant within i.
Type "reshape long" to convert the data to long form.
```

`reshape i`, `reshape j`, `reshape xij`, and `reshape xi` specifications may be given in any order and may be regiven to change or correct what has been specified.

Finally, `reshape clear` clears the definitions. Remember that `reshape` definitions are stored with the dataset when you save it. `reshape clear` provides a way to erase the definitions if you do not want this.

The basic syntax of reshape is implemented in terms of the advanced syntax. That means you can mix basic and advanced syntaxes.

Comparison with prior version of reshape

The syntax of the prior version of reshape was

reshape cons *varname* [*varname* ...]

reshape groups *groupvar* #[-#] [#[-#] ...]

reshape vars *varname* [*varname* ...]

reshape query

reshape wide

reshape long

You may continue to use the old syntax; the new `reshape` understands that. The new syntax is, however, better.

Here is the mapping between old and new syntax:

old syntax	new syntax
`reshape cons ...`	`reshape i ...`
`reshape groups ...`	`reshape j ...`
`reshape vars ...`	`reshape xij ...`
`reshape query`	`reshape query`
`reshape wide`	`reshape wide`
`reshape long`	`reshape long`

In the old `reshape cons` command you specified the identification variables and any variables that were constant within group. The new way to do that is

reshape i *identification_variables*

reshape xi *constant_within_group_variables*

and then, it would better if you left the *constant_within_group_variables* unspecified because the new `reshape` can figure that out for itself. If you specify the list, you are likely to forget a variable or two.

In addition, the old `reshape cons` command was limited to 10 variables. The new `reshape i` is also limited to 10 variables, but the constraint does not matter because it will certainly not take that many variables to identify the groups and the constant-within-group variables are specified either with the `reshape xi` command or left unspecified. There is now no limit as to the number of constant-within-group variables (explicitly specified or implied).

The new `reshape` uses less memory and is slightly faster once it gets going; it spends more time thinking about what it is going to do, however. The new `reshape` has better error checking—especially for data problems such as nonunique id variables that might otherwise go undiscovered. All those advantages accrue whether you use the old or new syntax.

Acknowledgment

We would like to thank Jeroen Weesie of Utrecht University for his `reshape2` command (Weesie 1997), the ideas of which were incorporated into this version of `reshape`.

Saved results

So that `reshape long` and `reshape wide` can be given without arguments, `reshape` stores the following characteristics with the data:

`_dta[ReS_ver]`	`v.1` if old-syntax commands were used `v.2` if new-syntax commands were used
`_dta[ReS_i]`	i variable name(s)
`_dta[ReS_j]`	j variable name
`_dta[ReS_jv]`	j values if specified
`_dta[ReS_Xij]`	X_ij variable name(s)
`_dta[ReS_Xi]`	X_i variable name(s) if specified
`_dta[ReS_atwl]`	`atwl()` value if specified
`_dta[ReS_str]`	1 if option `string` specified; otherwise 0.

References

Weesie, J. 1997. dm48: An enhancement of reshape. *Stata Technical Bulletin* 38: 2–4.

dm49	Some new matrix commands

Jeroen Weesie, Utrecht University, Netherlands, weesie@weesie.fsw.ruu.nl

This insert describes a collection of new matrix commands for Stata. Most of these commands operate on "explicit" Stata matrix objects. In my work, I frequently encounter situations in which I want to know some "linear algebra" property of a collection of variables. For example, "What is the matrix rank of these variables?" We might also ask "How many linear relationships exist between these variables and what are they?" Forming a matrix with `mkmat` is laborious and only possible with small datasets. Thus, I wrote additional "interface" programs that operate on "implicit matrices" formed by variables and row selection with standard `if` and `in` syntax.

In this insert, I do not seek to explicate linear algebra concepts, numerical linear algebra, and the applications of linear algebra in statistics. Rather, the reader should refer to the references.

Finally, I stayed away from implementing one of the most curious omissions in Stata: A linear equation solver. Stata's high-level interpreted language is simply too slow.

Contents

The matrix-oriented programs provided with this insert include (see help for `matfunc` for interactive help):

`matginv`	Moore–Penrose generalized-inverse
`matrank`	rank (number of independent rows/columns)
`matcond`	condition number
`matorth`	orthogonal basis of column space (image)
`matnull`	orthogonal basis of null space
`matnorm`	L2-norm of a matrix
`matsum`	row/column/overall sum
`matmax`	row/column/overall maximum
`matmin`	row/column/overall minimum
`matrand`	random matrix

The variable-oriented programs provided with this insert include (see help for `varfunc` for interactive help):

`varcond`	condition number of implied matrix
`varrank`	rank of implied matrix
`varorth`	Gram–Schmidt orthogonalization of variables
`varnull`	"null space" of variables

The matrix-oriented commands share a number of features:

1. They accept valid matrix expressions such as the name of a matrix, the sum of two matrices, the inverse of a matrix, and so on. Note that Stata currently does not support more general matrix expressions such as inv($X'X$).

2. If names for output are not specified, the output is displayed.

Matrix functions based on the singular value decomposition

`matginv` *matname* [, **g**`inv(`*name_matrix*`)` **r**`ank(`*name_scalar*`)` **t**`ol(`*#*`)` **d**`isplay` **f**`ormat(`*fmt*`)`]

returns the Moore–Penrose (MP) inverse of the *matname* in a matrix named in `ginv`. The scalar named in `rank` returns the rank of *matname*. If A is a $(n \times m)$ matrix, its MP-inverse B is $(m \times n)$, and is the unique solution of 1) $ABA = A$, 2) $BAB = B$, and 3) AB and BA are orthogonal projections. If A is square and regular, the MP-inverse is the ordinary matrix inverse. If A is square and singular, Stata's matrix function `syminv()` returns a g-inverse, which is a different kind of generalized matrix inverse. The row (column) names of the MP-inverse are the column (row) names of *matname*. See Campbell and Meyer (1979) for details on the applications of g-inverse in statistics, difference equations, and so on.

`matcond` *matname* [, **c**`ond(`*name_scalar*`)` **d**`isplay` **f**`ormat(`*fmt*`)`]

returns the condition number (ratio of largest and smallest singular value) of a matrix in the scalar named in `cond`.

`matrank` *matname* [, `rank(`*name_scalar*`)` `display` `tol(`*#*`)` `format(`*fmt*`)`]

returns the rank (number of independent rows or columns) of a matrix in the scalar named in `rank`.

`matorth` *matname* [, `orth(`*name_matrix*`)` `rank(`*name_scalar*`)` `display` `tol(`*#*`)` `format(`*fmt*`)`]

returns an orthogonal basis of the column space (image) of a matrix in the matrix named in `orth`, and the rank of the matrix in the scalar named in `rank`.

`matnull` *matname* [, `null(`*name_matrix*`)` `rank(`*name_scalar*`)` `display` `tol(`*#*`)` `format(`*fmt*`)`]

returns an orthogonal basis of the null space of a matrix in the matrix named in `null`, and the rank of the matrix in the scalar named in `rank`.

Other matrix functions

`matnorm` *matname* [, `norm(`*name_scalar*`)` `display` `format(`*fmt*`)`]

returns the L2-norm of a matrix in the scalar named in `norm`.

`matmax` *matname* [, `row(`*rmatname*`)` `column(`*cmatname*`)` `all(`*amatname*`)` `display` `format(`*fmt*`)`]

`matmin` *matname* [, `row(`*rmatname*`)` `column(`*cmatname*`)` `all(`*amatname*`)` `display` `format(`*fmt*`)`]

`matsum` *matname* [, `row(`*rmatname*`)` `column(`*cmatname*`)` `all(`*amatname*`)` `display` `format(`*fmt*`)`]

compute and/or display the row-wise, column-wise and over-all maximum, minimum, and sum over the elements of a matrix. More than one operator may be specified simultaneously and the result of each applied operator is placed in the matrix named in the `row`, `column`, and `all` options. These commands default to the column-wise operators.

`matrand` *n m* *matname* [, { `normal(`*mn var*`)` | `uniform(`*lo hi*`)` | `const(`*lo hi*`)` } `display` `format(`*fmt*`)`]

creates an $(n \times m)$ matrix (overwriting the named matrix if it already exists) with elements generated independently from

`normal`	distribution with mean *mn* and variance *var*
`uniform`	distribution on the interval [*lo*, *hi*]
`const`	distribution on the constants *lo*, *lo*+1,. . ., *hi*

If no distribution is specified, `matrand` defaults to the U(0,1) distribution.

Options for matrix-oriented programs

`display` specifies that the computed results are displayed. All programs that are described here display the results if output names for matrices (scalars) are not specified.

`format(`*fmt*`)` specifies a format (e.g., `%10.5f`, `%10.0g`) to display matrix results.

`tol(`*#*`)` specifies the tolerance for deciding what singular values are 'really' zero. See below for details. You typically don't have to set this option.

Numerical details

The main tool used by these programs is the singular value decomposition of the matrix $A = USV'$, where U and V are (column) orthonormal and S is diagonal with positive entries. We heavily lean on Stata's singular value decomposition command (`matrix svd`). Note that Stata can only compute the "economy" solution, and hence cannot be used directly to obtain the null space of a matrix.

Following Matlab, we compute the rank $r(A)$ of A as the number of singular values that exceed `tol`, where

$$\mathrm{tol} = \max(\text{singular values of } A) * \mathrm{size}(A) * \mathrm{eps}$$

where eps is a measure of machine precision.

This is also used in computing the MP-inverse (see below) and the column orthogonalization. The condition number $c(A)$ of A is

$$c(A) = \max(\text{singular values of } A)/\min(\text{singular values of } A)$$

If the minimum of the singular values is 0, the condition number is defined as missing.

The MP-inverse of a matrix is computed as $V * IS * U$, where IS is a diagonal matrix with the reciprocal of the singular values S, or 0 if the singular values are smaller than `tol` (see above).

Examples

```
. matrand 5 3 A                      (random matrix A, iid U(0,1))
. matrand 5 7 B, n(0 10)             (random matrix B, iid N(0,1))
. matginv A                          (display Moore–Penrose inverse)
. matginv A, ginv(AI)                (return Moore–Penrose inverse)
. matcond A+B                        (display condition number of A+B)
. matrank A, rank(ra)                (return rank of matrix A)
. matsum A, row(rsumA)               (return row-wise sum of A)
```

Details on variable-oriented programs

These programs return linear algebra characteristics of data, i.e., of an "implied matrix" with the variables in a *varlist* as columns, and with the (`if/in` selected) cases without missing values as the rows. All programs make it easy to append a variable consisting of all 1's to *varlist*.

`varcond` [*varlist*] [`if` *exp*] [`in` *range*] [*weight*] [, `cond(`*scalar_name*`)` `cons` `display` `format(`*fmt*`)`]

`varrank` [*varlist*] [`if` *exp*] [`in` *range*] [*weight*] [, `rank(`*scalar_name*`)` `cons` `display`]

`varorth` [*varlist*] [`if` *exp*] [`in` *range*] [*weight*] [, `prefix(`*str*`)` `cons` `norm` `eps(`*#*`)`]

`varnull` [*varlist*] [`if` *exp*] [`in` *range*] [*weight*] [, `null(`*matrix_name*`)` `rank(`*scalar_name*`)` `cons` `display` `format(`*fmt*`)`]

Description

`varcond` computes the condition number (ratio of largest to smallest singular values) of the implied matrix.

`varrank` computes the rank (number of independent columns or rows) of the implied matrix.

`varorth` computes a series of orthogonal variables that span the same column space as *varlist* by Gram–Schmidt orthogonalization.

`varnull` computes the "null space" of the implied matrix, i.e., a matrix of coefficients of linear dependencies between the variables.

Options

`cons` specifies that a constant variable consisting of 1's should be appended to the *varlist*. In `varorth`, the constant is prepended to the *varlist*, so that the generated variables are orthogonal to 1.

`display` specifies that the computed characteristics should be displayed. Recall that the programs automatically display results if no names for the results are provided.

`format(`*fmt*`)` specifies a format (e.g., `%10.4f`) used to display scalars or matrices.

Options for varorth

`prefix(`*str*`)` specifies the string prefixed to the variables in *varlist* to obtain the names of orthogonalized variables.

`norm` specifies that the returned variables are normalized, i.e., have unit length. The (weighted) covariance matrix of orthonormalized variables is a unity matrix.

`eps(`*#*`)` specifies a tolerance to decide whether a variable is linear dependent on the previous variables in the *varlist*. A more reliable way to obtain the number of linear dependencies is via `varrank`.

Examples

```
. varrank price weight turn              (rank of data matrix with 3 columns)
. varrank price weight turn if foreign, cons rank(r)
            (scalar r will contain rank of foreign data matrix with 4 columns)
. varorth x1 x2 x3 x4              (Gram-Schmidt orthogonalization of x1-x4)
. for 1-4, l(num) : gen I@ = inc^ @                    (polynomials in inc)
. varorth I*                                            (orthogonalize them)
. gen pw = price + weight                   (silly, but ok for illustration)
. varnull price weight pw    (varnull will show a matrix with coefficients that)
                             (indicate that pw is the sum of price and weight)
```

Technical details

The programs `varcond`, `varrank`, `varorth`, and `varnull` are variable-oriented versions of the matrix-oriented commands `matcond`, `matrank`, `matorth`, and `matnull`. It is important to understand that the variable-oriented and the matrix-oriented programs use different algorithms. The listed matrix-functions base their results on the singular value decomposition (SVD) of a matrix (using `matrix svd`). The SVD provides a numerically reliable algorithm for these functions, though slower than algorithms based on, e.g., the QR decomposition. However, the SVD may require much more memory than Stata's matrix modules can cope with. Thus, the variable-oriented versions are provided as low-precision alternatives. If you fear that your data are ill-conditioned, you have enough memory, and the number of cases does not exceed `matsize`, you should use `mkmat` to make a matrix from your data and use the matrix programs.

`varrank` counts the number of nonzero eigenvalues of $X'X$ (or $X'WX$ if weights are specified) based on the spectral decomposition (`matrix symeigen`).

`varcond` computes the condition number of the variables as the square root of the ratio of the largest to smallest eigenvalues of $X'X$ (or $X'WX$).

`varorth` uses Gram–Schmidt orthogonalization rather than the SVD of the set of orthogonal variables.

`varnull` returns the eigenspace of $X'X$ (or $X'WX$) associated with zero (or small) eigenvalues, obtained from the spectral decomposition.

References

Belsey, D. A., E. Kuh, and R. E. Welsch. 1980. *Regression Diagnostics. Identifying Influential Data and Sources of Collinearity*. New York: John Wiley & Sons.

Campbell, S. L. and C. D. Meyer, Jr. 1979. *Generalized Inverses of Linear Transformations*. London: Pitman.

Golob, G. H. and C. F. van Loan. 1983. *Matrix Computations*. Oxford: North Oxford Academic.

The MathWorks. *Matlab Reference Manual*. V4. Cambridge, MA: The MathWorks, Inc.

ip19	Using expressions in Stata commands

Jeroen Weesie, Utrecht University, Netherlands, weesie@weesie.fsw.ruu.nl

On numerous occasions, I wondered why Stata does not support a list of expressions (e.g., the log-transform of a variable; the sum of variables) in place of a list of variables. Currently, one has to generate "temporary" variables manually. Support for expressions where Stata now permits variables only, was requested a number of times by other Stata users on the Stata discussion list. In this insert, I describe two programs that add support for "expression expansion" in "regular" Stata commands, that is, commands satisfying the syntax

command [*varlist*] [`if` *exp*] [`in` *range*] [`,` *options*]

Examples of regular commands are the descriptive commands `summarize` and `tab` and the single-equation estimation commands `regress`, `logistic`, and `stcox`. Some new commands such as `heckman` and `mvreg` do not support equations.

Syntax

`expr` is a prefix command that enables expressions in regular commands.

`expr` [, nodetail]: *cmd expression_list* [if *exp*] [in *range*] [*weight*] [using *filename*] [,*cmd_options*]

`exprcmd` [, nodetail]

An *expression_list*, is a list of expressions, separated by one or more blanks. Thus, expressions may *not* contain embedded spaces as `expr` is not smart enough to parse such expressions. In addition, a variable list is interpreted as a list of "atomic" expressions, namely the untransformed existing variables.

Expression expansion is limited to the part of the input string prior to the first comma. Thus expression expansion does not involve options, equations, etc.

`expr` silently generates variables (of the default type, thus they are of type `float` if you haven't changed the default type; doing so is very dangerous anyway) named `_Expr0`, `_Expr1`, and so on. Any existing variables that fit the mask `_Expr` are silently dropped. The generated variables are labeled using the defining expressions.

Example

We illustrate `expr` with the automobile data distributed with Stata. We want to see summary statistics of the log price and log weight of cars. For comparison, we also include the untransformed variables `price` and `weight`:

```
. expr: summ price ln(price) weight ln(weight)
Expression _Expr0 := ln(price)
Expression _Expr1 := ln(weight)
-> summ  price _Expr0 weight _Expr1
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+-----------------------------------------------------
   price |      74    6165.257   2949.496       3291      15906
  _Expr0 |      74    8.640633   .3921059   8.098947   9.674452
  weight |      74    3019.459   777.1936       1760       4840
  _Expr1 |      74    7.978751   .2663436   7.473069    8.48467
```

Note that `expr` displays the meanings (defining expressions) of the variables that are defined as expressions.

The command `expr` can also be used to include transformed variables in linear (linear-form) models. Thus, to include the log price of a car as a predictor of its energy preservation, one can issue the command

```
. expr: reg mpg ln(price) weight foreign
Expression _Expr0 := ln(price)
-> reg  mpg _Expr0 weight foreign
  Source |       SS       df       MS                  Number of obs =      74
---------+------------------------------               F(  3,    70) =   45.84
   Model |  1619.29662     3  539.765539               Prob > F      =  0.0000
Residual |  824.162842    70  11.7737549               R-squared     =  0.6627
---------+------------------------------               Adj R-squared =  0.6483
   Total |  2443.45946    73  33.4720474               Root MSE      =  3.4313

------------------------------------------------------------------------------
     mpg |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  _Expr0 |  -.0419323   1.523501     -0.028   0.978      -3.08046    2.996595
  weight |  -.0065686   .0009508     -6.908   0.000     -.0084649   -.0046722
 foreign |  -1.627584   1.356207     -1.200   0.234     -4.332454    1.077285
   _cons |   41.97704   11.02088      3.809   0.000      19.99658     63.9575
------------------------------------------------------------------------------
```

If you use expressions in estimation commands, you may benefit from the more readable output of the parameter estimates that is yielded by the command `diest`, see Weesie (1996).

```
. diest
```

```
------------------------------------------------------------------------------
     mpg Mileage (mpg)                 |    Coef.   Std. Err.     t    P>|t|
---------------------------------------+--------------------------------------
  _Expr0 ln(price)                     | -.0419323  1.523501  -0.028   0.978
  weight Weight (lbs.)                 | -.0065686  .0009508  -6.908   0.000
 foreign Car type                      | -1.627584  1.356207  -1.200   0.234
   _cons                               |  41.97704  11.02088   3.809   0.000
------------------------------------------------------------------------------
```

When using `expr`, one can still use Stata's post-estimation commands such as `predict`, `test`, `fit`, and so on. For instance,

```
. predict res, res
. test _Expr0

 ( 1)  _Expr0 = 0.0

       F(  1,    70) =    0.00
            Prob > F =    0.9781
```

If one wants to use expression expansion interactively for a long series of commands, it may become tedious to type the prefix `expr :` before all commands. The command `exprcmd` "switches on" expression expansion in all subsequent commands that are entered interactively. To indicate that expression expansion is on, the command prompt is modified from Stata's period (.) to the string "`(expr) .`". Thus, the example session shown before could also be entered as

```
. exprcmd
(expr) . summ price ln(price) weight ln(weight)
(expr) . reg mpg ln(price) weight foreign
```

Since expression expansion is only supported in "regular commands," you may want to suppress expansion in some commands, for example `set`, `recode`, and so on. This is achieved simply by prefixing a command with a period (.), for example

```
(expr) . .set more off
```

Finally, interrupting a command with the Break key terminates the command, it does *not* terminate expression expansion. To terminate expansion, one should enter the command (.), i.e., a period.

References

Weesie, Jeroen. 1996. sg60: Enhancements for the display of estimation results. *Stata Technical Bulletin* 33: 12–15.

sbe17	Discrete time proportional hazards regression

Stephen P. Jenkins, ESRC Research Centre on Micro-Social Change, University of Essex, UK, stephenj@essex.ac.uk

Hazard rate models are widely used to model duration data in a wide range of disciplines, from biostatistics to economics. The aim of this insert is to supplement the portfolio of duration data analysis tools which Stata provides.

`pgmhaz` estimates, by maximum likelihood, two discrete time (grouped duration data) proportional hazards regression models, one of which incorporates a gamma mixture distribution to summarize unobserved individual heterogeneity (or "frailty"). Covariates may include regressor variables summarizing observed differences between persons (either fixed or time-varying), and variables summarizing the duration dependence of the hazard rate. With a suitable definition of covariates, models with a fully nonparametric specification for duration dependence may be estimated; so too may parametric specifications. `pgmhaz` thus provides a useful complement to `cox` and `st stcox`, `weibull` and `st stweib`.

The two models estimated are: (1) the Prentice–Gloeckler (1978) model; and (2) the Prentice–Gloeckler (1978) model incorporating a gamma mixture distribution to summarize unobserved individual heterogeneity, as proposed by Meyer (1990). These are referred to as Model 1 and Model 2 respectively below. My exposition of the models draws heavily on that of Stewart (1996).

The models

Suppose there are individuals $i = 1, \dots, n$, who each enter a state (e.g. unemployment) at time $t = 0$. The instantaneous hazard rate function for person i at time $t > 0$ is assumed to take the proportional hazards form

$$\lambda_{it} = \lambda_0(t) \exp(X'_{it}\beta) \tag{1}$$

where $\lambda_0(t)$ is the baseline hazard function, β is a vector of parameters to be estimated, and X_{it} is a vector of covariates summarizing observed differences between individuals at t. Let $\mathbf{X}_{it}$ represent the path of the covariate vector between time 0 and time t.

The underlying continuous durations are only observed in disjoint time intervals $[\,0 = a_0, a_1), [\,a_1, a_2), [\,a_2, a_3), \ldots, [\,a_{k-1}, a_k = \infty)$. (Alternatively durations are intrinsically discrete.) Assume that any time-dependent covariates only vary between duration intervals but not within them, i.e. they follow a piece-wise constant (step function) path over time. Thus, letting $a_t = t$, X_{it} is constant within each duration interval $[\,a_{t-1}, a_t)$.

The continuous-time survivor function is given by

$$S(t;\mathbf{X}_{it}) = \exp\left[-\sum_{j=a_1}^{a_t} \exp[X'_{ij}\beta + \gamma_j]\right], \text{ where } \gamma_j \equiv \log \int_{a_{j-1}}^{a_j} \lambda_0(\tau)d\tau \tag{2}$$

If there are no time-varying covariates ($X_{it} \equiv X_i$, for all t), (2) simplifies to

$$S(t;X_i) = \exp\left\{-\exp\left[X_i'\beta + \log(H_t)\right]\right\} \tag{3}$$

where $H_t \equiv \int_0^t \lambda_0(\tau)d\tau$ is the integrated baseline hazard at t.

The probability of exit in the jth interval for person i is

$$\Pr\{T \in [a_{j-1}, a_j)\} = S(a_{j-1};\mathbf{X}_{ij-1}) - S(a_j;\mathbf{X}_{ij})$$

and the survivor function at the start of the jth interval is

$$\Pr\{T \geq a_{j-1}\} = S(a_{j-1};\mathbf{X}_{ij-1})$$

The hazard of exit in the jth interval is thus given by

$$h_j(X_{it}) \equiv \Pr\{T \in [a_{j-1}, a_j) \mid T \geq a_{j-1}\} = 1 - [S(a_j;\mathbf{X}_{ij})/S(a_{j-1};\mathbf{X}_{ij-1})]$$

If there are no time-varying covariates, the discrete-time survivor function has exactly the same form as (3):

$$S(a_j;X_i) = \exp\left\{-\exp\left[X_i'\beta + \delta_j\right]\right\} \text{ where } \delta_j = \log(H_{a_j}) \text{ for } j = 1,\ldots,k$$

To simplify, let us now suppose that all intervals are of unit length (e.g. a week, or a month), so the recorded duration for each person i corresponds to the interval $[\,t_i - 1, t_i)$. Persons are also recorded as either having left the state during the interval, or as still remaining in the state. The former group, contributing completed spell data, are identified using the censoring indicator $c_i = 1$. For the latter group, contributing right-censored spell data, $c_i = 0$. Observe that the number of intervals comprising a censored spell is defined here to include the last interval within which the person is observed.

The sample log-likelihood can be written

$$\log L(\beta,\delta) = \sum_{i=1}^{n} \{c_i \log[S(t_i - 1;\mathbf{X}_{it_i-1}) - S(t_i;\mathbf{X}_{it_i})] - (1 - c_i)\log S(t_i;\mathbf{X}_{it_i})\}$$

This expression can be rewritten in terms of the hazard function as

$$\log L = \sum_{i=1}^{n} \left\{ c_i \log\left\{ h_{t_i}(X_{it_i}) \prod_{s=1}^{t_i-1} [1 - h_s(X_{is})] \right\} + (1 - c_i)\log\left\{ \prod_{s=1}^{t_i} [1 - h_s(X_{is})] \right\} \right\}$$

where the discrete-time hazard in the jth interval is

$$h_j(X_{ij}) = 1 - \exp[-\exp(X'_{ij}\beta + \gamma_j)]$$

This specification allows for a fully nonparametric baseline hazard with a separate parameter, γ_j, for each duration interval, which can be interpreted as the logarithm of the integral of the baseline hazard over the relevant interval. Alternatively, the sequence of the γ_j may be described by some semiparametric or parametric function.

If one defines an indicator variable $y_{it} = 1$ if person i exits the state during the interval $[\,t-1, t)$, $y_{it} = 0$ otherwise, then the log likelihood can be rewritten in sequential binary response form:

$$\log L = \sum_{i=1}^{n} \sum_{j=1}^{t_i} \{y_{ij} \log h_j(X_{ij}) + (1 - y_{ij}) \log[1 - h_j(X_{ij})]\}$$

This is the version of the Model 1 log likelihood which is estimated by `pgmhaz`.

Model 2 incorporates a gamma-distributed random variable to describe unobserved heterogeneity between individuals. For a discussion of, and comparison with, other types of mixed proportional hazards models, see Stewart (1996).

The instantaneous hazard rate is now specified as (cf. (1))

$$\lambda_{it} = \lambda_0(t)\varepsilon_i \exp(X_{it}) = \lambda_0(t) \exp[X_{it} + \log(\varepsilon_i)]$$

where ε_i is a gamma-distributed random variate with unit mean and variance $\sigma^2 \equiv v$, and the corresponding discrete-time hazard function is now

$$h_j(X_{ij}) = 1 - \exp\{-\exp[X_{ij}'\beta + \gamma_j + \log(\varepsilon_i)]\}.$$

Conveniently, the survivor function for the augmented model has a closed form expression (see Meyer 1990 or Dolton and van der Klaauw 1995, for details), and hence so too does the log likelihood function.

The Model 2 log likelihood function is

$$\log L = \sum_{i=1}^{n} \log\{(1 - c_i)A_i + c_i B_i\}$$

where

$$A_i = \left[1 + v \sum_{j=1}^{t_i} \exp\left[X_{ij}'\beta + \theta(j)\right]\right]^{-(1/v)}$$

$$B_i = \begin{cases} \left[1 + v \sum_{j=1}^{t_i - 1} \exp\left[X_{ij}'\beta + \theta(j)\right]\right]^{-(1/v)} - A_i, & \text{if } t_i > 1 \\ 1 - A_i, & \text{if } t_i = 1 \end{cases}$$

and $\theta(j)$ is a function describing duration dependence in the hazard rate, including the nonparametric baseline hazard specification (γ_j). The functional form for $\theta(j)$ is chosen by the user and specified by defining appropriate covariates—see below. Model 1's log likelihood function is the limiting case as $v \to 0$.

For suitably organized data, the log likelihood function for Model 1 is the same as the log likelihood for a generalized linear model of the binomial family with complementary log-log link: see Allison (1982) or Jenkins (1995). Model 1 is estimated by maximum likelihood using Stata's `glm` command. Model 2 is estimated using Stata's `ml deriv0` command, with starting values taken from Model 1's estimates. Given the potential fragility of models incorporating unobserved heterogeneity, estimates for both models are always reported.

Syntax

The syntax of `pgmhaz` is

`pgmhaz` *covariates* [`if` *exp*] [`in` *range*] `,` `id(`*idvar*`)` `dead(`*deadvar*`)` `seq(`*seqvar*`)`
[`lnvar0(`*#*`)` `eform` `level(`*#*`)` `nolog` `trace` `nocons`]

Options

`lnvar0(#)` specifies the value for $\ln(v)$ which is used as the starting value in the maximization. The default is -1 (i.e. $v \simeq 0.37$).

`eform` reports coefficients transformed to relative risk format, i.e. $\exp(\beta)$ rather than β. Standard errors and confidence intervals are similarly transformed. `eform` may be specified at estimation or when redisplaying results.

`level(#)` specifies the significance level, in percent, for confidence intervals of the parameters.

`nolog` suppresses the iteration logs.

`trace` reports the current value of the estimated parameters of Model 2 at each iteration; see [R] **maximize**.

`nocons` specifies no intercept term in the index function $X'_{ij}\beta$.

`Saved results` include the global macros set by `ml post`, plus `S_1` which contains the Model 2 log likelihood value at maximum, and `S_2` which contains the Model 1 log likelihood value at maximum.

Estimated coefficients and standard errors may be accessed in the usual way: see [U] **20.5 Accessing coefficients** and [R] **matrix get**.

Data organization and mandatory variables

The dataset must be organized before estimation so that, for each person, there are as many data rows as there are time intervals at risk of the event occurring for each person. Given the definitions above, this means t_i rows for each person $i = 1, \ldots, n$. In effect an unbalanced panel data set-up is required. This data organization is closely related to that required for estimation of Cox regression models with time-varying covariates. `expand` is useful for putting the data in this form: see [R] **expand**, and the example below. Also see the "data step" discussion in Jenkins (1995).

Three variables must be defined by the user:

`id(`*idvar*`)` specifies the variable uniquely identifying each person, i.

`seq(`*seqvar*`)` is the variable uniquely identifying each interval at risk for each person. For each i, *seqvar* is the integer sequence $1, 2, \ldots, t_i$.

`dead(`*deadvar*`)` summarizes censoring status during each interval at risk, and corresponds to the indicator variable y_{it} described earlier. If $c_i = 0$, *deadvar* $= 0$ for all $j = 1, 2, \ldots, t_i$; if $c_i = 1$, *deadvar* =0 for all $j = 1, 2, \ldots, t_i - 1$, and *deadvar* $= 1$ for $j = t_i$.

An example of how to construct these variables is given below.

Example

This illustration uses the cancer dataset (`cancer.dta`) supplied with Stata. The dataset provides information about survival times for 48 participants in a cancer drug trial. Of the 48 people, 28 receive the experimental drug treatment (`drug` = 1) and 20 receive the control treatment (`drug` = 0). The participants range in `age` from 47 to 67 years. We wish to analyze time until death, measured in months. The variable `studytim` records either the month of their death or the last month that they were known to be alive (the maximum value in the data is 39). The persons known to have died have variable `died` = 1 (contributing completed duration data); those still alive have `died` = 0 (contributing censored duration data).

First we `use` the data and recode `drug` so that it matches the Manual example:

```
. use cancer
(Patient Survival in Drug Trial)
. replace drug = 0 if drug == 1
(20 real changes made)
. replace drug = 1 if drug > 1
(28 real changes made)
```

To run `pgmhaz` we must reorganize the dataset and create the mandatory variables. To understand what is going on, look at how the data for the first four people is currently organized, and compare this with their data in the reorganized dataset later on.

```
. gen id = _n  /* create unique person identifier */
. list id studytim died drug age in 1/4
           id  studytim      died      drug       age
  1.        1         1         1         0        61
  2.        2         1         1         0        65
  3.        3         2         1         0        59
  4.        4         3         1         0        52
```

Now expand the dataset so that there is one data row per person per month at risk of dying, and create `seqvar` and `dead`:

```
. expand studytim
(696 observations created)
. sort id
. quietly by id: gen seqvar = _n
. quietly by id: gen dead = died & _n==_N
```

Compare this data format with the earlier one, taking the same four persons:

```
. list id studytim seqvar died dead age if id <= 4
           id  studytim     seqvar      died       dead       age
  1.        1         1          1         1          1        61
  2.        2         1          1         1          1        65
  3.        3         2          1         1          0        59
  4.        3         2          2         1          1        59
  5.        4         3          1         1          0        52
  6.        4         3          2         1          0        52
  7.        4         3          3         1          1        52
```

At this stage, with the data reorganized into person-month form, it would be straightforward to generate time-varying covariates. None are available in `cancer.dta` however. The illustrative estimations use the fixed covariates **drug** and **age**, capturing observed heterogeneity, and use the gamma mixing distribution to capture unobserved heterogeneity.

The first models estimated using **pgmhaz** assume that duration dependence in the hazard rate is summarized by a parametric "Weibull" specification. This is achieved by including a covariate defined as the logarithm of *seqvar*. (If the estimated coefficient on this regressor is greater than zero, the hazard increases monotonically with duration; if less than zero, it decreases monotonically.) The Model 1 estimates are precisely those which would be produced by the command

```
. gen logd = ln(seqvar)
. glm deadvar logd drug age, f(b) l(c)
```

except that in **pgmhaz** I have added output giving log likelihood values. Incidentally, the logistic hazard counterpart to this proportional hazards model could have been estimated with `logit` applied to the same reorganized dataset (Allison 1982, Jenkins 1995).

```
. pgmhaz logd drug age, id(id) s(seqvar) d(dead)
(1) PGM hazard model without unobserved heterogeneity
Iteration 1 : deviance =  298.3504
Iteration 2 : deviance =  237.2426
Iteration 3 : deviance =  224.1963
Iteration 4 : deviance =  222.5673
Iteration 5 : deviance =  222.5275
Iteration 6 : deviance =  222.5274
Iteration 7 : deviance =  222.5274
Residual df  =       740                          No. of obs =       744
Pearson X2   =  650.3937                          Deviance   =  222.5274
Dispersion   =  .8789105                          Dispersion =  .3007127
Bernoulli distribution, cloglog link
------------------------------------------------------------------------------
    dead |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
    logd |   .6402733   .2448109      2.615   0.009       .1604526    1.120094
    drug |   -2.18907   .4125618     -5.306   0.000      -2.997676   -1.380463
     age |    .119348   .0369335      3.231   0.001       .0469596    .1917364
   _cons |  -9.928747   2.262543     -4.388   0.000      -14.36325   -5.494243
------------------------------------------------------------------------------
Log likelihood (-0.5*Deviance) = -111.26371
   Cf. log likelihood for intercept-only model (Model 0) = -128.86467
   Chi-squared statistic for Model (1) vs. Model (0) = 35.201924
   Prob. > chi2(3) = 1.104e-07
(2) PGM hazard model with gamma distributed unobserved heterogeneity
Iteration 0:  Log Likelihood = -112.22135
Iteration 1:  Log Likelihood = -111.09624
Iteration 2:  Log Likelihood = -111.08967
Iteration 3:  Log Likelihood = -111.08965
Iteration 4:  Log Likelihood = -111.08965
PGM hazard model with gamma heterogeneity           Number of obs    =     744
                                                    Model chi2(3)    =       .
                                                    Prob > chi2      =       .
Log Likelihood =   -111.0896470
```

```
------------------------------------------------------------------------------
    dead |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
hazard   |
    logd |   .8664734   .4787207      1.810   0.070       -.071802    1.804749
    drug |  -2.578879   .8275313     -3.116   0.002      -4.200811   -.9569476
     age |    .141193   .0569466      2.479   0.013       .0295798    .2528062
   _cons |  -11.29142   3.510489     -3.216   0.001      -18.17185   -4.410984
---------+--------------------------------------------------------------------
ln_varg  |
   _cons |  -1.247006   1.845572     -0.676   0.499      -4.864262    2.370249
------------------------------------------------------------------------------
Gamma variance, exp(ln_varg) = .28736375; Std. Err. = .53035058; z = .54183734

Likelihood ratio statistic for testing models (1) vs (2) = .34812954
Prob. test statistic > chi2(1) = .55517386
```

Comparing **pgmhaz** Model 1 and Model 2 estimates, we see that the duration dependence parameter is larger in the latter. Moreover, the coefficients in Model 2 on `drug` and `age` are slightly larger in absolute value. These differences are not unexpected: not accounting for unobserved heterogeneity induces an underestimate of the extent to which the hazard rate increases with duration (or an overestimate of the decline) and attenuates the magnitude of the impact of covariates on the hazard rate (see Lancaster 1990, chapter 4).

The size of the variance of the gamma mixture distribution relative to its standard error suggests, however, that unobserved heterogeneity is not significant in this dataset. A likelihood ratio test of Model 2 versus Model 1 also suggests the same conclusion. Users should be aware though that standard likelihood ratio tests cannot, strictly speaking, be used to choose between Models 1 and 2, because the former is not a nested version of the latter.

The discrete-time “Weibull” estimates can be compared with the estimates of a continuous-time Weibull model derived using **stweib**:

```
. stset seqvar dead, id(id)

note:  making entry-time variable t0
       (within id, t0 will be 0 for the 1st observation and the
       lagged value of exit time seqvar thereafter)

   data set name:  cancer.dta
              id:  id
      entry time:  t0
       exit time:  seqvar
  failure/censor:  dead

. stweib drug age, nohr
```

(output omitted)

```
Weibull regression -- entry time t0
log relative hazard form

No. of subjects =          48                   Log likelihood =  -42.931336
No. of failures =          31                   chi2(2)        =       35.39
Time at risk    =         744                   Prob > chi2    =      0.0000

------------------------------------------------------------------------------
  seqvar |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
    drug |  -2.197157    .408785     -5.375   0.000      -2.998361   -1.395953
     age |   .1202128   .0371591      3.235   0.001       .0473823    .1930433
   _cons |  -10.58395   2.326241     -4.550   0.000       -15.1433   -6.024599
---------+--------------------------------------------------------------------
    ln p |   .5203303   .1389099      3.746   0.000       .2480718    .7925887
       p |   1.682583                                    1.281552    2.209108
     1/p |   .5943242                                    .4526714    .7803039
------------------------------------------------------------------------------
```

As it happens, the coefficient estimates are very similar to corresponding estimates in the discrete time “Weibull” model without unobserved heterogeneity. The duration dependence parameters are similar too: compare $1-p = 0.683$ with the coefficient on `logd`, 0.640.

We should be wary about drawing conclusions about duration dependence from parametric models like the “Weibull” which tightly constrain the shape of the baseline hazard function, when in fact the hazard may vary nonmonotonically with duration. Moreover it is well known that conclusions about the importance of unobserved heterogeneity are more reliably drawn if a flexible specification for the baseline hazard has been used; for a recent discussion, see Dolton and van der Klaauw (1995).

Let us therefore compare some models which allow for more flexibility in the shape of the baseline hazard function. One obvious reference point is the continuous time Cox model. Estimates for this are reported in *Stata Reference Manual*, vol. 3, p. 257, and can be reproduced with the command:

```
. stcox drug age, nohr baseh(coxbaseh)
(output omitted)
Cox regression -- entry time t0

No. of subjects =          48                   Log likelihood =  -83.323546
No. of failures =          31                   chi2(2)        =       33.18
Time at risk    =         744                   Prob > chi2    =      0.0000
------------------------------------------------------------------------------
  seqvar |
    dead |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
    drug |  -2.254965   .4548338     -4.958   0.000      -3.146423   -1.363507
     age |   .1136186   .0372848      3.047   0.002       .0405416    .1866955
------------------------------------------------------------------------------
```

The Cox baseline hazard function is graphed on p. 279 of *Stata Reference Manual, Release 4.0*, vol. 2, and the figure suggests that the hazard increases nonmonotonically with duration. (The picture can be reproduced with the command **gr coxbaseh studytim, xlab ylab**.) The estimates of the baseline hazard estimate are as follows, and confirm the nonmonotonicity:

```
. sort seqvar
. list seqvar coxbaseh if coxbaseh~=.

        seqvar    coxbaseh
 16.         1   .00013425
 46.         1   .00013425
 70.         2   .00007571
112.         3   .00007924
149.         4   .00018067
151.         4   .00018067
196.         5    .0002276
201.         5    .0002276
253.         6   .00026013
255.         6   .00026013
301.         7   .00013608
306.         8   .00044866
307.         8   .00044866
335.         8   .00044866
380.        10    .0001891
404.        11   .00041499
429.        11   .00041499
433.        12   .00056331
435.        12   .00056331
476.        13   .00035089
526.        15   .00038132
532.        16   .00043044
559.        17   .00047959
639.        22   .00149966
641.        22   .00149966
652.        23   .00249846
662.        23   .00249846
672.        24   .00168347
680.        25   .00189012
705.        28   .00228993
734.        33   .00437874
```

Let us now compare the Cox model estimates with various discrete time proportional hazard model specifications. One example of a flexible parametric specification for the baseline hazard function is a polynomial in duration. One could

```
. gen seqvar_2 = seqvar^2
. gen seqvar_3 = seqvar^3
```

and include `seqvar`, `seqvar_2`, and `seqvar_3`, as covariates instead of `logd` in order to specify a cubic baseline hazard function.

`pgmhaz` also allows the estimation of fully nonparametric specifications for the baseline hazard (analogously to the Cox model). The duration interval-specific baseline hazard can only be identified for those intervals during which events ('deaths') occur, i.e. values of `seqvar` for which there are observations (person-months) with `dead` = 1. If there are intervals for which this is not true, then either one must change the baseline hazard function specification, or one must drop the relevant person-month observations from the estimation. (see also the discussion of identification of the logit model in *Stata Reference Manual*, vol. 2, pp. 371–375.)

To estimate the nonparametric baseline model, first one has to create binary dummy variables corresponding to each duration interval. It is the user's responsibility to do this and also to check identifiability. This is straightforward. We can create interval-specific dichotomous variables, one for each spell month at risk (the maximum number is 39 here), with the following command:

```
. quietly for 1-39, ltype(numeric): gen byte d@ = seqvar== @
```

Next we check identifiability of the baseline hazard at each duration interval with

```
. tab seqvar dead
           | dead
    seqvar |         0          1 |     Total
-----------+----------------------+----------
         1 |        46          2 |        48
         2 |        45          1 |        46
         3 |        44          1 |        45
         4 |        42          2 |        44
         5 |        40          2 |        42
         6 |        38          2 |        40
         7 |        36          1 |        37
         8 |        33          3 |        36
         9 |        32          0 |        32
        10 |        30          1 |        31
        11 |        27          2 |        29
        12 |        24          2 |        26
        13 |        23          1 |        24
        14 |        23          0 |        23
        15 |        22          1 |        23
        16 |        20          1 |        21
        17 |        19          1 |        20
        18 |        18          0 |        18
        19 |        18          0 |        18
        20 |        16          0 |        16
        21 |        15          0 |        15
        22 |        13          2 |        15
        23 |        11          2 |        13
        24 |        10          1 |        11
        25 |         9          1 |        10
        26 |         8          0 |         8
        27 |         8          0 |         8
        28 |         7          1 |         8
        29 |         6          0 |         6
        30 |         6          0 |         6
        31 |         6          0 |         6
        32 |         6          0 |         6
        33 |         3          1 |         4
        34 |         3          0 |         3
        35 |         2          0 |         2
        36 |         1          0 |         1
        37 |         1          0 |         1
        38 |         1          0 |         1
        39 |         1          0 |         1
-----------+----------------------+----------
     Total |       713         31 |       744
```

There are no deaths during months 9, 14, 18–21, 26, 27, 29–32, 34–39, and so a month-specific hazard rate cannot be estimated for these intervals.

The nonparametric baseline model is estimated by including all the relevant duration interval dichotomous variables, excluding observations to ensure identifiability (if necessary), and excluding the intercept using the `nocons` option. (An alternative estimation strategy would be to include the intercept term and exclude one of the duration interval dichotomous variables.)

```
. pgmhaz d1-d8 d10-d13 d15-d17 d22-d25 d28 d33 drug age
> if (seqvar>=1 & seqvar<=8) | (seqvar>=10 & seqvar<=13)
> | (seqvar>=15 & seqvar<=17) | (seqvar>=22 & seqvar<=25)
> | seqvar==28 | seqvar==33 ,
> i(id) s(seqvar) d(dead) nocons

(1) PGM hazard model without unobserved heterogeneity
```

```
Iteration 1 : deviance =  255.2345
Iteration 2 : deviance =  206.7409
Iteration 3 : deviance =  195.2580
Iteration 4 : deviance =  193.6490
Iteration 5 : deviance =  193.5947
Iteration 6 : deviance =  193.5944
Iteration 7 : deviance =  193.5943
Iteration 8 : deviance =  193.5943

Residual df  =       550                          No. of obs =       573
Pearson X2   =  590.1132                          Deviance   =  193.5943
Dispersion   =  1.072933                          Dispersion =  .3519897

Bernoulli distribution, cloglog link
------------------------------------------------------------------------------
    dead |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
      d1 |  -9.321505   2.325432     -4.009   0.000      -13.87927   -4.763742
      d2 |  -9.888197   2.408603     -4.105   0.000      -14.60897   -5.167421
      d3 |  -9.841984   2.411291     -4.082   0.000      -14.56803   -5.115941
      d4 |  -9.008131   2.296365     -3.923   0.000      -13.50892   -4.507338
      d5 |  -8.758806   2.240112     -3.910   0.000      -13.14934   -4.368267
      d6 |  -8.617634   2.211921     -3.896   0.000      -12.95292    -4.28235
      d7 |  -9.269882    2.31374     -4.006   0.000      -13.80473   -4.735034
      d8 |  -8.075462   2.152716     -3.751   0.000      -12.29471   -3.856216
     d10 |  -8.931846   2.322521     -3.846   0.000       -13.4839   -4.379788
     d11 |  -8.144971   2.201254     -3.700   0.000      -12.45935   -3.830592
     d12 |  -7.819553   2.202429     -3.550   0.000      -12.13624   -3.502872
     d13 |   -8.27514   2.282109     -3.626   0.000      -12.74799   -3.802288
     d15 |  -8.190081   2.265841     -3.615   0.000      -12.63105   -3.749113
     d16 |  -8.068544   2.291659     -3.521   0.000      -12.56011   -3.576975
     d17 |  -7.959319   2.257287     -3.526   0.000      -12.38352   -3.535118
     d22 |  -6.799641   2.161635     -3.146   0.002      -11.03637   -2.562914
     d23 |  -6.231227    2.12435     -2.933   0.003      -10.39488   -2.067578
     d24 |  -6.597669   2.287659     -2.884   0.004       -11.0814    -2.11394
     d25 |  -6.481679   2.285573     -2.836   0.005      -10.96132   -2.002038
     d28 |  -6.293319   2.302273     -2.734   0.006      -10.80569   -1.780946
     d33 |  -5.654198   2.350609     -2.405   0.016      -10.26131    -1.04709
    drug |   -2.45515   .4668781     -5.259   0.000      -3.370214   -1.540086
     age |   .1208959    .037461      3.227   0.001       .0474738     .194318
------------------------------------------------------------------------------
Log likelihood (-0.5*Deviance) = -96.797174
   Cf. log likelihood for intercept-only model (Model 0) = -120.56974
   Chi-squared statistic for Model (1) vs. Model (0) = 47.545131
   Prob. > chi2(22) = .0012454
```

(2) PGM hazard model with gamma distributed unobserved heterogeneity

```
Iteration 0:  Log Likelihood =   -97.7371
(nonconcave function encountered)
Iteration 1:  Log Likelihood = -97.178695
Iteration 2:  Log Likelihood = -96.672111
Iteration 3:  Log Likelihood =  -96.66698
Iteration 4:  Log Likelihood = -96.666692
Iteration 5:  Log Likelihood = -96.666692

PGM hazard model with gamma heterogeneity          Number of obs    =     573
                                                   Model chi2(23)   =       .
                                                   Prob > chi2      =       .
Log Likelihood =    -96.6666923

------------------------------------------------------------------------------
    dead |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
hazard   |
      d1 |  -10.80181   3.943248     -2.739   0.006      -18.53043   -3.073185
      d2 |  -11.30775   3.896149     -2.902   0.004      -18.94406   -3.671436
      d3 |  -11.24518   3.875214     -2.902   0.004      -18.84046   -3.649898
      d4 |  -10.35178   3.703692     -2.795   0.005      -17.61088   -3.092676
      d5 |  -9.991745   3.504439     -2.851   0.004      -16.86032    -3.12317
      d6 |   -9.78455   3.385348     -2.890   0.004      -16.41971    -3.14939
      d7 |  -10.42769   3.437018     -3.034   0.002      -17.16412   -3.691257
      d8 |  -9.193994   3.290664     -2.794   0.005      -15.64358   -2.744411
     d10 |  -10.01127   3.344376     -2.993   0.003      -16.56612   -3.456411
     d11 |  -9.191331   3.235603     -2.841   0.005        -15.533   -2.849666
     d12 |  -8.820771   3.177368     -2.776   0.006       -15.0483   -2.593245
     d13 |  -9.261921   3.224425     -2.872   0.004      -15.58168   -2.942165
     d15 |  -9.145481   3.191924     -2.865   0.004      -15.40154   -2.889424
```

```
      d16 |  -8.978681   3.134165     -2.865   0.004      -15.12153    -2.83583
      d17 |  -8.819529   3.071867     -2.871   0.004      -14.84028   -2.798779
      d22 |  -7.626859   2.946399     -2.589   0.010      -13.40169   -1.852023
      d23 |  -7.076276   2.946105     -2.402   0.016      -12.85053   -1.302017
      d24 |  -7.436352    3.02082     -2.462   0.014      -13.35705   -1.515653
      d25 |  -7.276834    2.97176     -2.449   0.014      -13.10138   -1.452292
      d28 |  -7.038744   2.933167     -2.400   0.016      -12.78765   -1.289843
      d33 |  -6.276739   2.859726     -2.195   0.028       -11.8817   -.6717789
     drug |  -2.863101   .9693994     -2.953   0.003      -4.763088   -.9631126
      age |   .1468383   .0667209      2.201   0.028       .0160677    .2776089
---------+--------------------------------------------------------------------
ln_varg  |
    _cons |  -1.114756   2.041279     -0.546   0.585      -5.115589    2.886076
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Gamma variance, exp(ln_varg) = .32799525; Std. Err. = .66952971; z = .48988902
Likelihood ratio statistic for testing models (1) vs (2) = .26096361
Prob. test statistic > chi2(1) = .60945891
```

The results suggest that unobserved heterogeneity is not significant in this context, so our preferred specification is Model 1. Parameter estimates for this model correspond quite closely to the Cox ones. In particular, there is a close match in the pattern of variation of the baseline hazard with duration: compare the duration interval dummy variable coefficient estimates with the Cox model estimates listed earlier. The coefficients on `drug` and `age` are each somewhat larger in absolute value in the discrete-time model compared to the Cox model.

The earlier tabulation showed that, even for the months in which there were deaths, the number of deaths was relatively small. A model with a piece-wise constant baseline hazard function is an example of a compromise model which allows some nonparametric flexibility in the duration dependence specification, but may help estimation precision when there are few spell endings per duration interval. To specify a baseline hazard which is constant within six month intervals up to durations of 30 months and constant thereafter, but allowed to vary between these intervals, one would simply

```
. gen dur1 = d1+d2+d3+d4+d5+d6
. gen dur2 = d7+d8+d9+d10+d11+d12
. gen dur3 = d13+d14+d15+d16+d17+d18
. gen dur4 = d19+d20+d21+d22+d23+d24
. gen dur5 = d25+d26+d27+d28+d29+d30
. gen dur6 = d31+d32+d33+d34+d35+d36+d37+d38+d39
```

and use the command

```
. pgmhaz dur1-dur6 drug age, i(id) s(seqvar) d(dead) nocons
```

Estimates of Models 1 and 2 for this case (not shown here) indicate that unobserved heterogeneity is not significant and there is again evidence of a nonmonotonic increase in the baseline hazard with duration. The coefficients on age and drug are similar to those estimated by both the Cox model and the discrete time proportional hazards model with nonparametric baseline hazard.

Computational and other issues

`pgmhaz` can be slow, or rather estimation of Model 2 can be. This is partly because the maximization procedure uses numerical derivatives, and also partly because reorganized datasets can be relatively “large”. Models with fully nonparametric baseline hazard function specifications also take significantly longer to estimate than models with parsimonious parametric specifications. Using a Pentium P-120 “smrm PC with 32MB RAM and Stata 5.0 for Windows 3.11 for Workgroups, the “Weibull” `pgmhaz` model took about one minute to run, and the nonparametric baseline hazard model about seven minutes. Using a different dataset from `cancer.dta`, one with 7410 person-month observations, a model with one covariate and thirteen duration interval dummy variables took about 30 minutes to complete.

The log likelihood function for Model 2 is not globally concave, but convergence is usually achieved without problems. If there are maximization difficulties, users may find the `trace` option useful for diagnosing problems. Setting different starting values for the logarithm of the gamma variance with the `lnvar0(`*#*`)` option may also be helpful.

A warning. Because of the particular ordered sequence person-month structure of the data, the `if` option should be used with great care (and the `in` option should probably never be used). An `if` expression which refers to all the data rows for each person will be handled correctly, e.g., selection of an estimation sub-sample according to values of a fixed covariate. Do not select cases using an expression referring to a duration-varying variable or the results may be unpredictable. One exception to this rule arises when some observations need to be excluded to ensure identifiability of a model with a nonparametric baseline hazard function, as illustrated earlier. (In this context, there is one situation I am aware of in which the program will be incorrect if this strategy is followed. This is when the data contain a person contributing $s > 1$ intervals to the analysis who "dies" in the sth interval, and there are no "deaths" observed for any person in the sample during any of the duration intervals prior to s. This situation is likely to be rare.)

Acknowledgments

This work forms part of the scientific program of the ESRC Research Centre on Micro-Social Change, supported by the UK Economic and Social Research Council and the University of Essex. I owe many thanks to Espen Bratberg, Andy Dickerson, John Ermisch, Gene Fisher, Andy Henley, Wilbert van der Klaauw, and most especially Bill Sribney and Mark Stewart, for comments and advice.

References

Allison, P. D. 1982. Discrete-time methods for the analysis of event histories. In *Sociological Methodology*, ed. S. Leinhardt, 61–97. San Francisco: Jossey–Bass Publishers.

Dolton, P. and W. van der Klaauw 1995. Leaving teaching in the UK: a duration analysis. *Economic Journal* 105(429): 431–444.

Jenkins, S. P. 1995. Easy estimation methods for discrete-time duration models. *Oxford Bulletin of Economics and Statistics* 57(1): 129–138.

Lancaster, T. 1990. *The Econometric Analysis of Transition Data, Econometric Society Monograph No. 17*. Cambridge: Cambridge University Press.

Meyer, B. D. 1990. Unemployment insurance and unemployment spells. *Econometrica* 58(4): 757–782.

Prentice, R. and L. Gloeckler. 1978. Regression analysis of grouped survival data with application to breast cancer data. *Biometrics* 34: 57–67.

Stewart, M. B. 1996. Heterogeneity specification in unemployment duration models. unpublished paper, Department of Economics, University of Warwick, Coventry, UK.

sg72	Newey–West standard errors for probit, logit, and poisson models

James W. Hardin, Stata Corporation, stata@stata.com

This article discusses the calculation of standard errors that are robust to heteroscedasticity and serial correlation for probit, logit, and poisson regression models.

In order to calculate any statistic dealing with serial correlation, one must have a variable that identifies the time at which the individual observations were collected. This is to ensure that the observations are placed in the correct order and that the time steps between observations are constant so that the lag-dependent calculations are done correctly.

Methods and formulas

In this article, we focus on a model which may be written as $y_t = m_t(\mathbf{x}_t, \beta)$ where m_t is the conditional mean of y_t given the covariates $\mathbf{x}_t$. We also introduce a parametric family of weighting functions to be used in weighted nonlinear least squares (WNLS) estimation such that the weighting function $h_t(\mathbf{x}_t, \gamma)$ is strictly positive. The motivation of the weighting function is that for some value of γ, the weight function is proportional to the conditional variance of y_t given the covariates $\mathbf{x}_t$ and the WNLS estimator $\widehat{\beta}$ is then given by the value of β minimizing

$$\sum_{t=1}^{T} (y_t - m_t(\mathbf{x}_t - \beta))^2 / h_t(\mathbf{x}_t, \widehat{\gamma})$$

such that under the hypothesis that the conditional mean function is correct, the estimator is asymptotically normal with variance of the form $\mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}$. We can then estimate the variance using

$$\begin{aligned}
\widehat{\mathbf{A}} &= \frac{1}{T} \sum_{t=1}^{T} \widehat{g}_t^2 \\
\widehat{\mathbf{B}} &= \widehat{\Omega}_0 + \sum_{j=1}^{G} \omega(j) \left\{ \widehat{\Omega}_j + \widehat{\Omega}'_j \right\} \\
\widehat{\Omega}_j &= \frac{1}{T} \sum_{i=j+1}^{T} \widehat{s}_t \widehat{s}_{t-j} \\
\omega(j) &= 1 - j/(G+1)
\end{aligned}$$

where $\widehat{g}_t$ is the weighted gradient (evaluated at $\widehat{\beta}$), $\widehat{s}_t$ is the weighted residual, and G is the maximum lag to consider (the bandwidth). Note that under the hypothesis of independent observations ($G = 0$), we would have the usual White estimate of variance (c.f. the example of linear regression). Newey and West (1987) show that $\widehat{\mathbf{B}}$ is positive semi-definite. Other extensions of this approach that use a different weight function $\omega(j)$ and bandwidth G are not included in this command.

For details on the theoretical derivation of these estimators, the interested reader should consult Wooldridge (1991) or Hamilton (1994), especially chapter 10.

Syntax

The syntax for `nwest` is

`nwest` *command varlist* [`if` *exp*] [`in` *range*] [, `lag(`*#*`)` `level(`*#*`)` `t(`*varname*$_t$`)` *opts*]

where *command* is one of `regress`, `logit`, `probit`, or `poisson`.

Note that `nwest regress` will produce the same results as using Stata's `newey` command.

Options

`lag(`*#*`)` specifies the maximum lag to consider in calculating the standard errors. The default is zero.

`level(`*#*`)` specifies the confidence level, in percent, for confidence intervals. The default is `level(95)` or as set by `set level`; see [U] **26.4 Specifying the width of confidence intervals**.

`t(`*varname*$_t$`)` needs to be specified only if a nonzero `lag()` is also specified; `t()` specifies the variable that contains the time at which each observation was recorded. `t()` can also be omitted if it has been specified on a previous run; `nwest` will remember the previous identity of `t()`.

opts are options supported by the individual command (e.g., `irr` for `poisson`).

Example: linear regression models

We use the automobile dataset to illustrate that the `nwest` command will produce the same results as the `newey` command. We also show that considering the Newey–West estimate with lag zero is equivalent to the robust standard error provided by the `regress` command.

```
. nwest regress price weight displ

Regression with Newey-West standard errors          Number of obs  =        74
maximum lag : 0                                     F(  2,    71)  =     14.44
                                                    Prob > F       =    0.0000

------------------------------------------------------------------------------
         |             Newey-West
   price |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  weight |   1.823366   .7808755      2.335   0.022       .2663446    3.380387
   displ |   2.087054   7.436967      0.281   0.780      -12.74184    16.91595
   _cons |    247.907   1129.602      0.219   0.827      -2004.454    2500.269
------------------------------------------------------------------------------

. newey price weight displ, lag(0)

Regression with Newey-West standard errors          Number of obs  =        74
maximum lag : 0                                     F(  2,    71)  =     14.44
                                                    Prob > F       =    0.0000

------------------------------------------------------------------------------
         |             Newey-West
   price |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  weight |   1.823366   .7808755      2.335   0.022       .2663446    3.380387
   displ |   2.087054   7.436967      0.281   0.780      -12.74184    16.91595
   _cons |    247.907   1129.602      0.219   0.827      -2004.454    2500.269
------------------------------------------------------------------------------

. regress price weight displ, robust
```

```
Regression with robust standard errors                 Number of obs =      74
                                                       F(  2,    71) =   14.44
                                                       Prob > F      =  0.0000
                                                       R-squared     =  0.2909
                                                       Root MSE      =  2518.4

------------------------------------------------------------------------------
         |               Robust
   price |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  weight |   1.823366   .7808755      2.335   0.022       .2663446    3.380387
   displ |   2.087054   7.436967      0.281   0.780      -12.74184    16.91595
   _cons |    247.907   1129.602      0.219   0.827      -2004.454    2500.269
------------------------------------------------------------------------------
```

Example: probit and logit models

In this example, we use data on the unionization of women. We first present the results of a probit analysis not addressing possible heteroscedasticity or serial correlation.

```
. probit union age grade not_smsa south southXt

Iteration 0:  Log Likelihood =-10465.715
Iteration 1:  Log Likelihood =-10252.844
Iteration 2:  Log Likelihood =-10252.282
Iteration 3:  Log Likelihood =-10252.282

Probit Estimates                                        Number of obs =  19224
                                                        chi2(5)       = 426.87
                                                        Prob > chi2   = 0.0000
Log Likelihood = -10252.282                             Pseudo R2     = 0.0204

------------------------------------------------------------------------------
   union |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
     age |   .0058862   .0018991      3.099   0.002       .0021639    .0096084
   grade |   .0188945   .0042971      4.397   0.000       .0104724    .0273167
not_smsa |  -.1474075   .0233975     -6.300   0.000      -.1932658   -.1015493
   south |  -.3319289    .285638     -1.162   0.245       -.891769    .2279113
 southXt |  -.0002766   .0035351     -0.078   0.938      -.0072052    .0066521
   _cons |  -.9784501   .0793062    -12.338   0.000      -1.133887   -.8230127
------------------------------------------------------------------------------
```

As there are repeated observations over time, we suspect that there will be a problem with serial correlation which should be addressed (up to lag 2).

```
. nwest probit union age grade not_smsa south southXt, lag(2) t(time)

Probit with Newey-West standard errors              Number of obs  =     19224
maximum lag : 2                                     chi2(  5)      =    227.09
                                                    Prob > chi2    =    0.0000

------------------------------------------------------------------------------
         |             Newey-West
   union |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
     age |   .0058862   .0024516      2.401   0.016       .0010811    .0106912
   grade |   .0188945   .0063701      2.966   0.003       .0064093    .0313798
not_smsa |  -.1474075   .0318795     -4.624   0.000      -.2098902   -.0849249
   south |  -.3319289   .3573839     -0.929   0.353      -1.032389    .3685308
 southXt |  -.0002766   .0044246     -0.063   0.950      -.0089486    .0083954
   _cons |  -.9784501   .1103485     -8.867   0.000      -1.194729   -.7621709
------------------------------------------------------------------------------
```

Note that probit and logit models may be run the same way using the logit keyword instead of the probit keyword.

Example: poisson models

In this example, we use the data listed in the *Stata Reference Manual*:

```
. list

        agecat     smokes     deaths     pyears
  1.         1          1         32      52407
  2.         2          1        104      43248
  3.         3          1        206      28612
  4.         4          1        186      12663
  5.         5          1        102       5317
```

```
  6.          1          0          2      18790
  7.          2          0         12      10673
  8.          3          0         28       5710
  9.          4          0         28       2585
 10.          5          0         31       1462
```

We would like to obtain the robust standard errors but the `poisson` command does not have a `robust` option. We also can not obtain robust standard errors from the `glm` command as it does not have the `robust` option either.

```
. tab agecat, gen(a)
     agecat |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2       20.00       20.00
          2 |          2       20.00       40.00
          3 |          2       20.00       60.00
          4 |          2       20.00       80.00
          5 |          2       20.00      100.00
------------+-----------------------------------
      Total |         10      100.00
. poisson deaths smokes a2-a5, exposure(pyears) irr
Iteration 0: Log Likelihood = -34.022705
Iteration 1: Log Likelihood = -33.60083
Iteration 2: Log Likelihood = -33.600098
Poisson regression, normalized by pyears            Number of obs    =      10
Goodness-of-fit chi2(4)     =    12.132             Model chi2(5)    = 922.935
Prob > chi2                 =    0.0164             Prob > chi2      =  0.0000
Log Likelihood              =   -33.600             Pseudo R2        =  0.9321
------------------------------------------------------------------------------
  deaths |        IRR   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  smokes |   1.425518   .1530638      3.302   0.001       1.154984    1.759421
      a2 |   4.410583   .8605195      7.606   0.000       3.009011    6.464996
      a3 |    13.8392   2.542637     14.301   0.000       9.654325    19.83809
      a4 |   28.51678   5.269876     18.130   0.000       19.85177    40.96395
      a5 |    40.4512    7.77551     19.249   0.000       27.75325    58.95885
------------------------------------------------------------------------------
. nwest poisson deaths smokes a2-a5, exposure(pyears) irr
Poisson with Newey-West standard errors             Number of obs  =        10
maximum lag : 0                                     chi2(  5)      =   7089.45
                                                    Prob > chi2    =    0.0000
------------------------------------------------------------------------------
         |             Robust
  deaths |        IRR   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  smokes |   1.425518   .1665546      3.034   0.002       1.133758    1.792361
      a2 |   4.410583   .9255229      7.072   0.000       2.923335    6.654468
      a3 |    13.8392   2.760089     13.174   0.000       9.361538    20.45853
      a4 |   28.51678    5.69211     16.786   0.000       19.28394    42.17015
      a5 |    40.4512   9.105199     16.438   0.000       26.02158    62.88241
------------------------------------------------------------------------------
```

References

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

Newey, W. K. and K. D. West. 1987. A simple positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. *Econometrica* 55: 703–708.

Wooldridge, J. M. 1991. On the application of robust, regression-based diagnostics to models of conditional means and conditional variances. *Journal of Econometrics* 47: 5–46.

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	*qs*	questions and suggestions
dt	datasets	*tt*	teaching
gr	graphics	*zz*	not elsewhere classified
in	instruction		

Statistical Categories:

sbe	biostatistics & epidemiology	*ssa*	survival analysis
sed	exploratory data analysis	*ssi*	simulation & random numbers
sg	general statistics	*sss*	social science & psychometrics
smv	multivariate analysis	*sts*	time-series, econometrics
snp	nonparametric methods	*svy*	survey sampling
sqc	quality control	*sxd*	experimental design
sqv	analysis of qualitative variables	*szz*	not elsewhere classified
srd	robust methods & statistical diagnostics		

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

International Stata Distributors

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