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

Associate Editors
Francis X. Diebold, University of Pennsylvania
Joanne M. Garrett, University of North Carolina
Marcello Pagano, Harvard School of Public Health
James L. Powell, UC Berkeley and Princeton University
J. Patrick Royston, Royal Postgraduate Medical School

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.

Copyright Statement. The Stata Technical Bulletin (STB) and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp. The contents of the supporting files (programs, datasets, and help files), may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the STB.

The insertions appearing in the STB may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the STB. Written permission must be obtained from Stata Corporation if you wish to make electronic copies of the insertions.

Users of any of the software, ideas, data, or other materials published in the STB or the supporting files understand that such use is made without warranty of any kind, either by the STB, the author, or Stata Corporation. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the STB is to promote free communication among Stata users.

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

Contents of this issue

<table>
<thead>
<tr>
<th>stata47. lookup now indexes on-line FAQs</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>stata48. Updated reshape</td>
<td>4</td>
</tr>
<tr>
<td>dm49. Some new matrix commands</td>
<td>17</td>
</tr>
<tr>
<td>ip19. Using expressions in Stata commands</td>
<td>20</td>
</tr>
<tr>
<td>sbe17. Discrete time proportional hazards regression</td>
<td>22</td>
</tr>
<tr>
<td>sg72. Newey–West standard errors for probit, logit, and poisson models</td>
<td>32</td>
</tr>
</tbody>
</table>
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 7/96 What does completely determined mean in my logistic regression output?

On-line FAQ at http://www.stata.com/support/faq/stat/
Click on Logistic and probit regressions

The referenced URL http://www.stata.com/support/faq/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 4/97 In ordered probit and logit, what are the cut points?

On-line FAQ at http://www.stata.com/support/faq/stat/
Click on Logistic and probit regressions

FAQ 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/faq/stat/
Click on Logistic and probit regressions

FAQ 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/faq/stat/
Click on Logistic and probit regressions

FAQ 8/95 Is there a nested logit program for Stata?

On-line FAQ at http://www.stata.com/support/faq/stat/
Click on Logistic and probit regressions

FAQ 7/96 What does completely determined mean in my logistic regression output?

On-line FAQ at http://www.stata.com/support/faq/stat/
Click on Logistic and probit regressions

FAQ 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/faq/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?
Online 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?
Online 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 cr41) can be used to evaluate mortality prediction models

STB-35  sg53  . Logistic regression: standardized coeff. & 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  sg2  . Plotting predicted values from linear & logistic reg. 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.

<table>
<thead>
<tr>
<th>stata48</th>
<th>Updated reshape</th>
</tr>
</thead>
</table>

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 atw1(chars)` and both are seldom specified.

**Advanced syntax**

```
reshape i varlist
```

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

```
reshape xij fvarnames , atw1(chars) 
```

```
reshape xi [varlist]
```

```
reshape query
```

```
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:

<table>
<thead>
<tr>
<th>id</th>
<th>sex</th>
<th>inc0</th>
<th>inc01</th>
<th>inc02</th>
<th>inc03</th>
<th>inc04</th>
<th>inc05</th>
<th>inc06</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5000</td>
<td>5500</td>
<td>6000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2000</td>
<td>2200</td>
<td>3300</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3000</td>
<td>2000</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`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,

```
+-----------------+   +-----------------+
| . 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 \texttt{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 \texttt{year} when the data is in the long form. There is no \( j \) variable in the wide form. Instead, the \texttt{inc} variable is suffixed with the values of \( j \), forming \texttt{inc80}, \texttt{inc81}, and \texttt{inc82}.

That leaves only the variable \texttt{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 \texttt{sex} is not specified, \texttt{sex} is assumed to be constant within \( i \) and \texttt{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  we80  we81  we82
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 we, 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)  ->  \texttt{year}
\( x_{ij} \) variables:
  inc80 inc81 inc82  ->  inc
  we80 we81 we82  ->  \texttt{ue}
```

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

```
.list
id  year  sex  inc  we
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 we, i(id) j(year)
```

*note: j = 80 81 82*

**Data**

```matlab
<table>
<thead>
<tr>
<th></th>
<th>wide</th>
<th><code>-&gt;</code></th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of obs.</td>
<td>9</td>
<td><code>-&gt;</code></td>
<td>3</td>
</tr>
<tr>
<td>Number of variables</td>
<td>5</td>
<td><code>-&gt;</code></td>
<td>8</td>
</tr>
<tr>
<td>j variable (3 values)</td>
<td>year</td>
<td><code>-&gt;</code></td>
<td>(dropped)</td>
</tr>
<tr>
<td>xij variables:</td>
<td>inc</td>
<td><code>-&gt;</code></td>
<td>inc80 inc81 inc82</td>
</tr>
<tr>
<td></td>
<td>we</td>
<td><code>-&gt;</code></td>
<td>we80 we81 we82</td>
</tr>
</tbody>
</table>
```

we would be right back to our original data,

```
. list
```

```plaintext
<table>
<thead>
<tr>
<th>id</th>
<th>inc80</th>
<th>we80</th>
<th>inc81</th>
<th>we81</th>
<th>inc82</th>
<th>we82</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>5000</td>
<td></td>
<td>5500</td>
<td>1</td>
<td>6000</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2000</td>
<td>1</td>
<td>2200</td>
<td>0</td>
<td>3300</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3000</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>1000</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

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
```

```plaintext
<table>
<thead>
<tr>
<th>id</th>
<th>sex</th>
<th>inc80</th>
<th>inc81</th>
<th>inc82</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5000</td>
<td>5500</td>
<td>6000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2000</td>
<td>2200</td>
<td>3300</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3000</td>
<td>2000</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2400</td>
<td>2500</td>
<td>2400</td>
<td></td>
</tr>
</tbody>
</table>
```

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

*note: j = 80 81 82*

```
. list
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>sex</th>
<th>inc80</th>
<th>inc81</th>
<th>inc82</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5000</td>
<td>5500</td>
<td>6000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2000</td>
<td>2200</td>
<td>3300</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3000</td>
<td>2000</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2400</td>
<td>2500</td>
<td>2400</td>
<td></td>
</tr>
</tbody>
</table>
```

```
. 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  |
  3  |
(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
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, \texttt{reshape error} will assist in finding the problem:

\begin{verbatim}
    . reshape wide inc, i(id) j(year)
    (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:
    \begin{verbatim}
    id year
    9. 3 82
    10. 3 82
    \end{verbatim}
    (data now sorted by \(id\) \(year\))
\end{verbatim}

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

\begin{verbatim}
    . list
    \begin{verbatim}
    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 0
    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
    \end{verbatim}
\end{verbatim}

What if we forget to specify \(ue\), a variable that varies within \(i\)?

\begin{verbatim}
    . 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.
    \texttt{r(9)};
\end{verbatim}

In this case \texttt{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 \texttt{reshape wide inc, i(id) j(year)}. We could type \texttt{reshape error} if the problem were not obvious.

In summary, there are three cases in which \texttt{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 | use80 | use81 | use82 |
    +---------+---+-------+-------+-------+-------+-------+-------+
    | 1. | 0 | 5000 | 5500 | 6000 | 0 | 1 | 0 |
    | 2. | 1 | 2000 | 2200 | 3300 | 1 | 0 | 0 |
    | 3. | 0 | 3000 | 3000 | 1000 | 0 | 0 | 1 |
. reshape long inc, i(id) j(year)
    (note: j = 80 81 82)
```

Data

```
| +---------+---+-------+-------+-------+-------+-------+-------+-------+---------|
| Number of obs: 3 | -> | 9 |
| Number of variables: 8 | -> | 7 |
| j variable (3 values): | -> | year |
| rij variables: inc80 inc81 inc82 | -> | inc |
```

 reshape does not complain but here is the result:

```
. list
    +---------+---+-------+-------+-------+-------+-------+-------+-------+
    | id | year | sex | use80 | use81 | use82 | inc |
    +---------+---+-------+-------+-------+-------+-------+
    | 1. | 80 | 0 | 0 | 1 | 0 | 5000 |
    | 2. | 81 | 0 | 0 | 1 | 0 | 5500 |
    | 3. | 82 | 0 | 0 | 1 | 0 | 6000 |
    | 4. | 80 | 1 | 1 | 0 | 0 | 2000 |
    | 5. | 81 | 1 | 1 | 0 | 0 | 2200 |
    | 6. | 82 | 1 | 1 | 0 | 0 | 3300 |
    | 7. | 80 | 0 | 0 | 0 | 1 | 3000 |
    | 8. | 81 | 0 | 0 | 0 | 1 | 2000 |
    | 9. | 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 incue, 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 \texttt{reshape} about them by typing out the full \texttt{reshape} command, although no real damage will be done if you forget. If you are converting from long to wide, \texttt{reshape} itself will catch your error and refuse to do the conversion. If you are converting from wide to long, \texttt{reshape} will convert the data, but the result will be surprising (remember what happened when we forget to mention variable \texttt{ue} and ended up with \texttt{ue80}, \texttt{ue81}, and \texttt{ue82} in our long data). You can ‘\texttt{reshape long}’ to undo it and then try again.

**Missing variables**

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

\begin{verbatim}
   . list
   +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
   | id | sex | inc80 | inc81 | inc82 | ue80 | ue82 |
   +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
   | 1  | 1   | 0    | 5000  | 5500  | 6000 | 0    |
   | 2  | 1   | 2000 | 2300  | 3300  | 1    | 0    |
   | 3  | 3   | 3000 | 2000  | 1000  | 0    | 1    |
   +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+

Note that variable \texttt{ue81} is missing. We can still type

\begin{verbatim}
   . reshape long inc ue, i(id) j(year)
   (note: \texttt{j = 80 81 82})
   (note: \texttt{ue81 not found})
   \end{verbatim}

Data	wide $\rightarrow$ long

| Number of obs. | 3 $\rightarrow$ 9 |
| Number of variables | 7 $\rightarrow$ 5 |
| \texttt{j variable (3 values)} | $\rightarrow$ \texttt{year} |
| \texttt{xij variables} | \texttt{inc80 inc81 inc82} $\rightarrow$ \texttt{inc} |
| | \texttt{ue80 ue81 ue82} $\rightarrow$ \texttt{ue} |

The result is

\begin{verbatim}
   . list, nodisplay
   +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
   | id | year | sex | inc | ue |
   +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
   | 1  | 80   | 0   | 5000 | 0  |
   | 2  | 81   | 0   | 5500 | 0  |
   | 3  | 82   | 0   | 6000 | 0  |
   | 4  | 80   | 1   | 2000 | 1  |
   | 5  | 81   | 1   | 2200 | 0  |
   | 6  | 82   | 1   | 3300 | 0  |
   | 7  | 80   | 0   | 3000 | 0  |
   | 8  | 81   | 0   | 2000 | 0  |
   | 9  | 82   | 0   | 1000 | 1  |
   +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+

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

**Advanced issues with basic syntax: i()**

The syntax is

\begin{verbatim}
   reshape ..., i(\texttt{i\_variable}) ... or reshape ..., i(\texttt{i\_variables}) ...
\end{verbatim}

Typically \texttt{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

\begin{verbatim}
   reshape ..., i(id)
   reshape ..., i(hid pid)
\end{verbatim}

**Advanced issues with basic syntax: j()**

The syntax is

\begin{verbatim}
   reshape ..., ... j(\texttt{j\_variable}) ... or reshape ..., ... j(\texttt{j\_variable j\_values}) ...
\end{verbatim}
The second syntax defines the \textit{j} variable and its values. When you leave the values unspecified, \texttt{reshape} works them out for itself. \texttt{reshape} never makes a mistake when the data is in long form—when you type \texttt{reshape wide}. The values can easily be obtained by tabulating the existing \textit{j} variable.

\texttt{reshape} can make a mistake when your data is in the wide form—when you type \texttt{reshape long}—if you have named your variables poorly. Pretend you have the variables \texttt{inc/8/0}, \texttt{inc/8/1}, and \texttt{inc/8/2}, recording income in each of the indicated years, and you have a variable named \texttt{inc/2}, which is not income but when the area was reincorporated. You type

\begin{verbatim}
   . reshape long inc, i(id) j(year)
\end{verbatim}

\texttt{reshape} would see the variables \texttt{inc/2}, \texttt{inc/8/0}, \texttt{inc/8/1}, and \texttt{inc/8/2} and decide that there are four groups: \textit{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 \texttt{inc} followed by a number; \texttt{reinc} would be a good name.

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

\begin{verbatim}
   . reshape long inc, i(id) j(year 80-82)
\end{verbatim}

or

\begin{verbatim}
   . reshape long inc, i(id) j(year 80 81 82)
\end{verbatim}

The dash notation for ranges can be mixed with individual numbers. \texttt{reshape} would understand 80 82–87 89 91–95.

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

**Advanced issues with basic syntax: X\_ij**

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

\begin{verbatim}
   . list
      id  sex  inc/8/0  inc/8/1  inc/8/2  ue/8/0  ue/8/1  ue/8/2
    1.  1  0   5000   5000   6000    0   1   0
    2.  2  1  2000  2200  3300    1   0   0
    3.  3  0  3000  2000 1000    0   0   1
\end{verbatim}

you would type

\begin{verbatim}
   . reshape long inc\@r  ue\@r, i(id) j(year)
   (note:  j = 80 81 82)
\end{verbatim}

\begin{verbatim}
   \begin{tabular}{lrr}
   Data & wide \rightarrow long \\
   \hline
   Number of obs. & 3 \rightarrow 9 \\
   Number of variables & 8 \rightarrow 5 \\
   \textit{j} variable (3 values) \rightarrow \text{year} \\
   \text{rij} variables: inc\@r  inc\@1r  inc\@2r \rightarrow \text{incr} \\
   & ue\@0  ue\@1  ue\@2 \rightarrow \text{ue} \\
   \hline
   \end{tabular}
\end{verbatim}

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

\begin{verbatim}
   . reshape long inc\@r  ue\@r, i(id) j(year)
\end{verbatim}

Either way, the result of the \texttt{reshape long} will be

\begin{verbatim}
   . 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
\end{verbatim}
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 5000
   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
   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

Data

<table>
<thead>
<tr>
<th></th>
<th>wide</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of obs.</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Number of variables</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>j variable (2 values)</td>
<td>sex</td>
<td>(dropped)</td>
</tr>
<tr>
<td>xij variables:</td>
<td>inc</td>
<td>incf incm</td>
</tr>
</tbody>
</table>

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

. list
<table>
<thead>
<tr>
<th>id</th>
<th>kids</th>
<th>incmale</th>
<th>incfem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>2</td>
<td>2000</td>
</tr>
<tr>
<td>3</td>
<td>103</td>
<td>1</td>
<td>3000</td>
</tr>
</tbody>
</table>

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

. list
<table>
<thead>
<tr>
<th>id</th>
<th>sex</th>
<th>kids</th>
<th>inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fem</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>2</td>
<td>male</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>3</td>
<td>fem</td>
<td>2</td>
<td>2000</td>
</tr>
<tr>
<td>4</td>
<td>male</td>
<td>2</td>
<td>2000</td>
</tr>
<tr>
<td>5</td>
<td>fem</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>6</td>
<td>male</td>
<td>1</td>
<td>3000</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>id</th>
<th>kids</th>
<th>minc</th>
<th>finc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>2</td>
<td>2000</td>
</tr>
<tr>
<td>3</td>
<td>103</td>
<td>1</td>
<td>3000</td>
</tr>
</tbody>
</table>

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, umen, uenef, and aegm, aegf. To make the data long, the person types

. reshape long inc uage, 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, uen0, uen1, and aeg0, aeg1 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

<table>
<thead>
<tr>
<th>hid</th>
<th>sex</th>
<th>year</th>
<th>inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>107</td>
<td>m</td>
<td>90</td>
</tr>
<tr>
<td>2.</td>
<td>107</td>
<td>f</td>
<td>90</td>
</tr>
<tr>
<td>3.</td>
<td>107</td>
<td>m</td>
<td>91</td>
</tr>
<tr>
<td>4.</td>
<td>107</td>
<td>f</td>
<td>91</td>
</tr>
</tbody>
</table>

or in the long-year wide-sex form

<table>
<thead>
<tr>
<th>hid</th>
<th>year</th>
<th>finc</th>
<th>minc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>107</td>
<td>90</td>
<td>3200</td>
</tr>
<tr>
<td>2.</td>
<td>107</td>
<td>91</td>
<td>4700</td>
</tr>
</tbody>
</table>

or in the wide-year long-sex form

<table>
<thead>
<tr>
<th>hid</th>
<th>sex</th>
<th>inc90</th>
<th>inc91</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>107</td>
<td>f</td>
<td>3200</td>
</tr>
<tr>
<td>2.</td>
<td>107</td>
<td>m</td>
<td>4500</td>
</tr>
</tbody>
</table>

or the wide-wide form

<table>
<thead>
<tr>
<th>hid</th>
<th>finc</th>
<th>minc</th>
<th>finc</th>
<th>minc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>107</td>
<td>3200</td>
<td>4700</td>
<td>4600</td>
</tr>
</tbody>
</table>

`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

<table>
<thead>
<tr>
<th>gid</th>
<th>year</th>
<th>rs1</th>
<th>rs2</th>
<th>rs3</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>26</td>
<td>90</td>
<td>57</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>2.</td>
<td>26</td>
<td>91</td>
<td>52</td>
<td>33</td>
<td>32</td>
</tr>
</tbody>
</table>

into

<table>
<thead>
<tr>
<th>gid</th>
<th>visit</th>
<th>rs90</th>
<th>rs91</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>26</td>
<td>1</td>
<td>57</td>
<td>52</td>
</tr>
<tr>
<td>2.</td>
<td>26</td>
<td>2</td>
<td>28</td>
<td>33</td>
</tr>
<tr>
<td>3.</td>
<td>26</td>
<td>3</td>
<td>30</td>
<td>32</td>
</tr>
</tbody>
</table>

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
you type a sequence of \texttt{reshape} commands. The initial commands describe the data and the last performs the conversion:

\begin{verbatim}
  . reshape i id
  . reshape j year
  . reshape xij inc
  . reshape long
\end{verbatim}

\texttt{reshape i} corresponds to \texttt{i()} in the basic syntax.  
\texttt{reshape j} corresponds to \texttt{j()} in the basic syntax.  
\texttt{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: 

\texttt{reshape xi \textit{varlist}}

In the basic syntax, all unspecified variables are assumed to be constant within \texttt{i}. The advanced syntax works the same way unless you specify the \texttt{reshape xi} command. \texttt{reshape xi} names the constant-within-\texttt{i} variables. If you specify \texttt{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 \texttt{inc80}, \texttt{inc81}, \texttt{inc82}, \texttt{sex}, \texttt{age}, and \texttt{age2} and that you no longer want the \texttt{age2} variable. You could specify

\begin{verbatim}
  . reshape xi sex age
\end{verbatim}

or you could

\begin{verbatim}
  . drop age2
  . reshape xi
\end{verbatim}

and leave \texttt{reshape xi} unspecified. \texttt{reshape xi} does have one minor advantage: It saves \texttt{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 \texttt{reshape query}—equivalent to typing \texttt{reshape} by itself. \texttt{reshape query} presents a report on what has been defined:

\begin{verbatim}
. reshape (data is wide)
+---------------------------------------------------------------+
| Iij | Command/contents |
+---------------------------------------------------------------+
| Subscript i,j definitions: |
| group id variable(s) | reshape i hid |
| and its range | reshape j year |
| 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.
\end{verbatim}

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

Finally, \texttt{reshape clear} clears the definitions. Remember that \texttt{reshape} definitions are stored with the dataset when you save it. \texttt{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

\[ \text{reshape cons} \ varname \ [\varname \ldots] \]
\[ \text{reshape groups} \ groupvar \ [#[-#] \ [#[-#] \ldots]] \]
\[ \text{reshape vars} \ varname \ [\varname \ldots] \]
\[ \text{reshape query} \]
\[ \text{reshape wide} \]
\[ \text{reshape long} \]

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:

<table>
<thead>
<tr>
<th>old syntax</th>
<th>new syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>reshape cons ...</td>
<td>reshape i ...</td>
</tr>
<tr>
<td>reshape groups ...</td>
<td>reshape j ...</td>
</tr>
<tr>
<td>reshape vars ...</td>
<td>reshape xij ...</td>
</tr>
<tr>
<td>reshape query</td>
<td>reshape query</td>
</tr>
<tr>
<td>reshape wide</td>
<td>reshape wide</td>
</tr>
<tr>
<td>reshape long</td>
<td>reshape long</td>
</tr>
</tbody>
</table>

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

\[ \text{reshape i} \ \text{identification variables} \]
\[ \text{reshape xi} \ \text{constant within group variables} \]

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:

- \[ .\text{dta}[\text{Resver}] \] v. 1 if old-syntax commands were used
- \[ .\text{dta}[\text{Resw}] \] v. 2 if new-syntax commands were used
- \[ .\text{dta}[\text{Resi}] \] i variable name(s)
- \[ .\text{dta}[\text{Resj}] \] j variable name
- \[ .\text{dta}[\text{Resji}] \] j values if specified
- \[ .\text{dta}[\text{Resxij}] \] X_i variable name(s)
- \[ .\text{dta}[\text{Resxii}] \] X_i variable name(s) if specified
- \[ .\text{dta}[\text{Resxstv}] \] atvl() value if specified
- \[ .\text{dta}[\text{Resxstr}] \] 1 if option string specified; otherwise 0.

References

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 [, ginv(name_matrix) rank(name_scalar) tol(#) display format(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 [, cond(name_scalar) display format(fmt)]
```

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

```
matrank matname [ , 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.

```
matorth matname [ , orth(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.

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

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 \(lo, lo+1, \ldots, hi\)

If no distribution is specified, \texttt{matrand} defaults to the U(0,1) distribution.

**Options for matrix-oriented programs**

\texttt{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.

\texttt{format(fmt)} specifies a format (e.g., \texttt{%.10f,%.10g}) to display matrix results.

\texttt{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 \(A = USV^T\), where \(U\) and \(V\) are (column) orthonormal and \(S\) is diagonal with positive entries. We heavily lean on Stata’s singular value decomposition command
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

\[
tol = \max(\text{singular values of } A) \times \text{size}(A) \times \text{eps}
\]

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

\[
c(A) = \frac{\max(\text{singular values of } A)}{\min(\text{singular values of } A)}
\]

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 \( V/S \times U \), 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(10)` (random matrix \( B \), iid \( N(0,1) \))
- `matginv A` (display Moore–Penrose inverse)
- `matginv A, ginv(A)` (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` 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.

**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 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)
. varorth x1 x2 x3 x4       // (Gram-Schmidt orthogonalization of x1-x4)
. for 1/4, l(num) : gen D@ = inc' @   // (polynomials in inc)
. varorth D*                 // (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


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 ]
```

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)
. expr cmd _Expr0 weight _Expr1
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>74</td>
<td>6.165.257</td>
<td>29.49.46</td>
<td>3291</td>
<td>15906</td>
</tr>
<tr>
<td>_Expr0</td>
<td>74</td>
<td>8.640633</td>
<td>3.921059</td>
<td>8.0.09847</td>
<td>9.674452</td>
</tr>
<tr>
<td>weight</td>
<td>74</td>
<td>3019.459</td>
<td>777.1936</td>
<td>1760</td>
<td>4840</td>
</tr>
<tr>
<td>_Expr1</td>
<td>74</td>
<td>7.978751</td>
<td>2.663436</td>
<td>7.473069</td>
<td>8.49467</td>
</tr>
</tbody>
</table>

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)
```

```
Source | SS | df | MS
-------|----|----|----
Model  | 1619.29662 | 3 | 539.765539
Residual | 824.362842 | 70 | 11.7737549
-------|----|----|----
Adj R-squared | 0.6627 |
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 | -.0493323 | 1.523503 | -0.038 | 0.978 | -.3.084046 | .2986595 |
| weight | -.0055666 | .0099105 | 6.908 | 0.000 | -0.089464 | .0036722 |
| foreign | -.1.627894 | 1.356207 | 1.000 | 0.000 | -.4.332454 | .1.07285 |
| _cons | 41.977704 | 11.02068 | 3.809 | 0.000 | 19.9.99658 | 63.9.9575 |

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
```
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

---

**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 $i = 1, \ldots, n$, 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

\[
\lambda_t = \lambda_0(t) \exp(X_{t}^{\prime} \beta)
\]
where \( \lambda_0(t) \) is the baseline hazard function, \( \beta \) is a vector of parameters to be estimated, and \( X_{it} \) is a vector of covariates summarizing observed differences between individuals at \( t \). Let \( X_{it} \) 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

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

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

\[
S(t; X_i) = \exp \{- \exp[X_i\beta + \log(H_t)]\}
\]  

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

The probability of exit in the \( j \)th interval for person \( i \) is

\[
\Pr\{T \in [a_{j-1}, a_j)\} = S(a_{j-1}; X_{ij-1}) - S(a_j; X_{ij})
\]

and the survivor function at the start of the \( j \)th interval is

\[
\Pr\{T \geq a_{j-1}\} = S(a_{j-1}; X_{ij-1})
\]

The hazard of exit in the \( j \)th interval is thus given by

\[
h_j(X_{it}) \equiv \Pr\{T \in [a_{j-1}, a_j) | T \geq a_{j-1}\} = 1 - \frac{S(a_j; X_{ij})}{S(a_{j-1}; X_{ij-1})}
\]

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

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

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 \([t_{j-1}, t_j)\). 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

\[
\log L(\beta, \delta) = \sum_{i=1}^{n} \left\{ c_i \log[S(t_i - 1; X_{it,-1}) - S(t_i; X_{it})] - (1 - c_i) \log S(t_i; X_{it}) \right\}
\]

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

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

where the discrete-time hazard in the \( j \)th 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, $\gamma_j$, 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 $\gamma_j$ 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))

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

where $\varepsilon_t$ is a gamma-distributed random variate with unit mean and variance $\sigma^2 \equiv \nu$, 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

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

where

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

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

and $\theta(j)$ is a function describing duration dependence in the hazard rate, including the nonparametric baseline hazard specification ($\gamma_j$). 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 $\nu \to 0$.

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) 
    [lnvar(#) ef(# #) level(#) nolog trace nocons]
```


Options

lnvar(0#) specifies the value for ln(v) which is used as the starting value in the maximization. The default is -1 (i.e. v ≈ 0.37).
eform reports coefficients transformed to relative risk format, i.e. exp(β) 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 X0iβ.

Saved results include the global macros set by ml post, plus S1 which contains the Model 2 log likelihood value at maximum, and S2 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 ti rows for each person i = 1,..., 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,...,ti.

dead(deadvar) summarizes censoring status during each interval at risk, and corresponds to the indicator variable yit described earlier. If c1 = 0, deadvar = 0 for all j = 1,2,...,ti; if c1 = 1, deadvar =0 for all j = 1,2,...,ti - 1, and deadvar = 1 for j = ti.

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.

```
    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).

(1) PGM hazard model without unobserved heterogeneity

Iteration 1: deviance = 298.3504
Iteration 2: deviance = 237.2425
Iteration 3: deviance = 224.1963
Iteration 4: deviance = 222.8673
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

Bemoulli distribution, cloglog link

|     | Coef. | Std. Err. | z    | P>|z|    | [95% Conf. Interval] |
|-----|-------|-----------|------|--------|---------------------|
| logd | .6407233 | .2448109 | 2.615 | .009   | .1604526            |
| age  | -.188907 | .4125618 | -5.306 | .000   | -2.997676            |
| _cons| -.119345 | .0360335 | 3.231 | .001   | .0469356            |

Log likelihood = -111.26371
Cf. Log likelihood for intercept-only model (Model 0) = -128.86467
Chi-squared statistic for Model (1) vs. Model (0) = 36.201924
Prob. > chi2(3) = 1.104e-7

(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
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.

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 $\log d$, 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 t/0
No/. of failures /= /3/1 chi/2/(/2/) /= /3/3/./1/8
Time at risk /= /7/4/4 Prob > chi/2 /= /0/./0/0/0/0
```

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 0.00013425
46.  1 0.00013425
70.  2 0.0007871
112. 3 0.0007924
149. 4 0.0010667
151. 4 0.0010667
196. 5 0.002276
201. 5 0.002276
253. 6 0.0026013
255. 6 0.0026013
301. 7 0.0036083
306. 8 0.0044666
307. 8 0.0044666
335. 8 0.0044666
380. 10 0.001891
404. 11 0.0041499
429. 11 0.0041499
433. 12 0.0056331
435. 12 0.0056331
476. 13 0.0035069
526. 15 0.0038132
552. 16 0.0043043
559. 17 0.0047359
639. 22 0.0018966
641. 22 0.0018966
652. 23 0.0029848
662. 23 0.0029848
672. 24 0.0018966
680. 25 0.0018966
705. 28 0.0028593
734. 33 0.0043787
```

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/-/3/9/, ltype/(numeric/): gen byte d/@ /= seqvar/=/= /@

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

`. tab seqvar dead

<table>
<thead>
<tr>
<th>seqvar</th>
<th>dead</th>
<th>0</th>
<th>1</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46</td>
<td>2</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>1</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>1</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>42</td>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>2</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>38</td>
<td>2</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>36</td>
<td>1</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>3</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>32</td>
<td>0</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>1</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>27</td>
<td>2</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>2</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>23</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>23</td>
<td>0</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>22</td>
<td>1</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>1</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>16</td>
<td>0</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>13</td>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>11</td>
<td>2</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>10</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>713</td>
<td>31</td>
<td>744</td>
<td></td>
</tr>
</tbody>
</table>

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-d85 d10-d13 d15-d17 d22-d25 d26 d33 drug age
> if (seqvar>=1 & seqvar<8) | (seqvar>=10 & seqvar<13)
> | (seqvar>=15 & seqvar<17) | (seqvar>=22 & seqvar<25)
> | seqvar==26 | seqvar==33
> i(id) s(seqvar) d(dend) nocons

(1) PGM hazard model without unobserved heterogeneity
Iteration 1: deviance = 255.2345
Iteration 2: deviance = 206.7409
Iteration 3: deviance = 196.2560
Iteration 4: deviance = 193.6940
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 X^2 = 590.1132
Deviance = 193.5943

Dispersion = 1.072923
Dispersion = 0.3651927

Bernoulli distribution, cloglog link

| hazard | Conf. Std. Err. z P>|z| [95% Conf. Interval] |
|--------|-----------------|--------|---------|-----------------|
| d1     | -10.80181       | 3.943248 | -2.739  | 0.006 -18.53043 | -0.37185 |
| d2     | -11.30775       | 3.896149 | -2.902  | 0.004 -18.94406 | -0.367436 |
| d3     | -11.24518       | 3.875214 | -2.902  | 0.004 -18.84046 | -0.36498 |
| d4     | -10.35178       | 3.703692 | -2.795  | 0.005 -17.61088 | -0.39376 |
| d5     | -9.691745       | 3.604349 | -2.851  | 0.004 -16.68032 | -0.313217 |
| d6     | -9.78045       | 3.85348 | -2.290  | 0.024 -16.41971 | -0.34939 |
| d7     | -10.47269       | 3.437018 | -3.034  | 0.002 -17.16412 | -0.35927 |
| d8     | -9.193944       | 3.20664 | -2.794  | 0.006 -15.64358 | -0.374441 |
| d9     | -10.01127       | 3.344376 | -2.993  | 0.003 -16.66112 | -0.36511 |
| d10    | -9.193311       | 3.235603 | -2.841  | 0.004 -15.53324 | -0.392766 |
| d11    | -8.620771       | 3.177868 | -2.776  | 0.006 -15.04632 | -0.387324 |
| d12    | -9.834313       | 3.224125 | -2.672  | 0.007 -15.98858 | -0.312687 |
| d13    | -9.145481       | 3.191924 | -2.865  | 0.004 -15.40154 | -0.389424 |

Log likelihood (-0.5*Deviance) = -96.79714
Chi-squared statistic for Model (1) vs. Model (0) = 47.545131
Prob > chi2(22) = 0.0012454

(2) PGM hazard model with gamma distributed unobserved heterogeneity

Iteration 0: Log Likelihood = -97.7371
Concave function encountered

Iteration 1: Log Likelihood = -97.178695
Iteration 2: Log Likelihood = -96.672111
Iteration 3: Log Likelihood = -96.666692
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

| hazard | Conf. Std. Err. z P>|z| [95%Conf.Interval] |
|--------|-----------------|--------|---------|-----------------|
| d1     | -10.80181       | 3.943248 | -2.739  | 0.006 -18.53043 | -0.37185 |
| d2     | -11.30775       | 3.896149 | -2.902  | 0.004 -18.94406 | -0.367436 |
| d3     | -11.24518       | 3.875214 | -2.902  | 0.004 -18.84046 | -0.36498 |
| d4     | -10.35178       | 3.703692 | -2.795  | 0.005 -17.61088 | -0.39376 |
| d5     | -9.691745       | 3.604349 | -2.851  | 0.004 -16.68032 | -0.313217 |
| d6     | -9.78045       | 3.85348 | -2.290  | 0.024 -16.41971 | -0.34939 |
| d7     | -10.47269       | 3.437018 | -3.034  | 0.002 -17.16412 | -0.35927 |
| d8     | -9.193944       | 3.20664 | -2.794  | 0.006 -15.64358 | -0.374441 |
| d9     | -10.01127       | 3.344376 | -2.993  | 0.003 -16.66112 | -0.36511 |
| d10    | -9.193311       | 3.235603 | -2.841  | 0.004 -15.53324 | -0.392766 |
| d11    | -8.620771       | 3.177868 | -2.776  | 0.006 -15.04632 | -0.387324 |
| d12    | -9.834313       | 3.224125 | -2.672  | 0.007 -15.98858 | -0.312687 |
| d13    | -9.145481       | 3.191924 | -2.865  | 0.004 -15.40154 | -0.389424 |
but allowed to vary between these intervals, one would simply
specify a baseline hazard which is constant within six month intervals up to durations of 30 months and constant thereafter,
flexibility in the duration dependence specification, but may help estimation precision when there are few spell endings per duration
model with a piece-wise constant baseline hazard function is an example of a compromise model which allows some nonparametric
compared to the Cox model.

\[
\begin{align*}
\text{lnvar/0/} = \text{exp(ln/varg/)} = 0.592799526; \text{ Std. Err.} = 0.069253711; z = 8.8886902 \\
\text{Likelihood ratio statistic for testing models (1) vs (2) = 12.069361} \\
\text{Prob. test statistic > chisq(1) = 0.00456991}
\end{align*}
\]

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

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

and use the command

\[
\text{pgmhaz dur1-dur6 drug age, i(id) s(seqvar) d(dead) nocons}
\]

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**

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

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 \text{trace} option useful for diagnosing problems. Setting different starting values for the logarithm of the gamma variance with the \text{lnvar/0/} option may also be helpful.

A warning. Because of the particular ordered sequence person-month structure of the data, the \text{if} option should be used with great care (and the \text{in} option should probably never be used). An \text{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 \( s \)th 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


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 \( y_t = m_t(x_t, \beta) \) where \( m_t \) is the conditional mean of \( y_t \) given the covariates \( x_t \). We also introduce a parametric family of weighting functions to be used in weighted nonlinear least squares (WNLS) estimation such that the weighting function \( h_t(x_t, \gamma) \) is strictly positive. The motivation of the weighting function is that for some value of \( \gamma \), the weight function is proportional to the conditional variance of \( y_t \) given the covariates \( x_t \) and the WNLS estimator \( \beta \) is then given by the value of \( \beta \) minimizing

\[
\frac{1}{T} \sum_{t=1}^{T} (y_t - m_t(x_t, \beta))^2 / h_t(x_t, \gamma)
\]

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

\[
\hat{A} = \frac{1}{T} \sum_{t=1}^{T} g_t^2
\]

\[
\hat{B} = \hat{\Omega}_0 + \sum_{j=1}^{G} \omega(j) \left\{ \hat{\Omega}_j + \hat{\Omega}_j' \right\}
\]

\[
\hat{\Omega}_j = \frac{1}{T} \sum_{t=j+1}^{T} \tilde{s}_t \tilde{s}_{t-j}
\]

\[
\omega(j) = 1 - j / (G + 1)
\]

where \( g_t \) is the weighted gradient (evaluated at \( \hat{\beta} \)), \( \tilde{s}_t \) 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 $\hat{B}$ 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) 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 [1] 26.4 Specifying the width of confidence intervals.

`t(varname)` 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

| Coef. Std. Err. t P>|t| [95% Conf. Interval] Newey-West
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>1.823366  .7808755  2.335  0.022  .2663446  3.380387</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>2.067054  7.4306977  0.281  0.780 -12.74184  16.91595</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>displ</td>
<td>2.47907  1129.602  0.219  0.827 -2004.454  2506.269</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_cons</td>
<td>2.47907  1129.602  0.219  0.827 -2004.454  2506.269</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

. 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

| Coef. Std. Err. t P>|t| [95% Conf. Interval] Newey-West
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>1.823366  .7808755  2.335  0.022  .2663446  3.380387</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>2.067054  7.4306977  0.281  0.780 -12.74184  16.91595</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>displ</td>
<td>2.47907  1129.602  0.219  0.827 -2004.454  2506.269</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_cons</td>
<td>2.47907  1129.602  0.219  0.827 -2004.454  2506.269</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

. regress price weight displ, robust
```
Regression with robust standard errors

|         | Coef. | Std. Err. | t  | P>|t| | [95% Conf. Interval] |
|---------|-------|-----------|----|------|---------------------|
| price   | 1.823366 | .7808755 | 2.335 | 0.022 | [.2065466 | 3.380387 |
| disp    | 2.067064 | 7.436967 | 0.281 | 0.780 | -12.7454 | 16.91596 |
| cons    | 247.9207 | 112.6002 | 2.192 | 0.033 | -200.484 | 696.326 |

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
```

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
```

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:
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)

<table>
<thead>
<tr>
<th>agecat</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>20.00</td>
<td>20.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>20.00</td>
<td>40.00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>20.00</td>
<td>60.00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>20.00</td>
<td>80.00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>20.00</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Total | 10 | 100.00
```

```
. poisson deaths smokes a2-a5, exposure(pyears) irr
Iteration 0: Log likelihood = -34.622705
Iteration 1: Log likelihood = -33.600993
Iteration 2: Log likelihood = -33.600993
Poisson regression, normalized by pyears
Number of obs = 10
Goodness-of-fit chi2(4) = 12.132
Model chi2(5) = 92.935
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 | .1850936 | 3.032 | 0.000 | 1.156954 | 1.79492 |
| a2     | 4.410583 | .860192   | 5.069 | 0.000 | 3.009011 | 6.464966|
| a3     | 13.8392  | 2.842837  | 14.301| 0.000 | 9.65325  | 19.8389 |
| a4     | 28.5176  | 5.298797  | 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
chisq( 5) = 70.5945
Prob > chisq = 0.0000

| deaths | Robust IRR | Std. Err. | z     | P>|z| | 95% Conf. Interval |
|--------|------------|-----------|-------|------|-------------------|
| smokes | 1.425518   | .1865466  | 3.034 | 0.002 | 1.133758 | 1.792361|
| a2     | 4.410583   | .925529   | 7.007 | 0.000 | 2.923335 | 6.654468|
| a3     | 13.8392    | 2.76666   | 13.174| 0.000 | 9.361538 | 20.45583|
| a4     | 28.5176    | 5.69211   | 16.786| 0.000 | 19.26394 | 42.17015|
| a5     | 40.4512    | 9.105199  | 16.438| 0.000 | 26.02158 | 62.88241|
```

References


STB categories and insert codes

Inserts in the STB are presently categorized as follows:

General Categories:
- an announcements
- cc communications & letters
- dm data management
- dt datasets
- gr graphics
- in instruction
- ip instruction on programming
- os operating system, hardware, & interprogram communication
- qs questions and suggestions
- tt teaching
- zz not elsewhere classified

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

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.

Company: Applied Statistics & Systems Consultants
Address: P.O. Box 1169
Nazerath-Ellit 17100, Israel
Phone: +92 66554254
Fax: +92 66554254
Email: sasconsl@actcom.co.il
Countries served: Israel

Company: Dittrich & Partner Consulting
Address: Prinzenstrasse 2
D-42697 Solingen
Germany
Phone: +49 212-3390 200
Fax: +49 212-3390 295
Email: evhall@dpc.de
Countries served: Austria, Germany, Italy

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

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

Company: SmiT Consult
Address: Postbox 220
5150 AE Drunen
Netherlands
Phone: +31 416-378 125
Fax: +31 416-378 385
Email: j.a.c.m.smit@smitcon.nl
Countries served: Netherlands

Company: Survey Design & Analysis Services
Address: 249 Eramosa Road West
Moorooduc VIC 3933
Australia
Phone: +61 3 59788329
Fax: +61 3 59788623
Email: rosier@survey-design.com.au
Countries served: Australia

Company: Timberlake Consultants
Address: Praceta do Comércio,
N°13–9º Dto. Quinta Grande
2720 Alfragide Portugal
Phone: +351 (01) 4719337
Email: timberlake.co@mail.telepac.pt
Countries served: Portugal