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

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

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

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

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

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

| an70. | Fall NetCourse schedule announced | 2 |
| dm66.1. | Stata 6 version of recoding variables using grouped values | 3 |
| dm68. | Display of variables in blocks | 3 |
| dm69. | Further new matrix commands | 5 |
| dm70. | Extensions to generate, extended | 9 |
| gr38. | Enhancement to the hilite command | 17 |
| ip28. | Automatically sorting by subgroup | 20 |
| sbe29. | Generalized linear models: extensions to the binomial family | 21 |
| sg112.1. | Multivariable fractional polynomials: update | 25 |
| sg112.1. | Nonlinear regression models involving power or exponential functions of covariates: update | 26 |
| sg113. | Tabulation of modes | 26 |
| sg114. | rglm - Robust variance estimates for generalized linear models | 27 |
| stata53. | censored option added to sts graph command | 34 |
| sxd1.1. | Update to random allocation of treatments to blocks | 36 |
We have announced our latest NetCourse schedule:

**NetCourse 101. An Introduction to Stata**
- 6 weeks (4 lectures)
- Course dates: August 20 through October 1
- Deadline for enrollment: August 16
- Cost: $85
- Course leaders: Mark Esman, Allen McDowell, and Kyle Willman
- Prerequisites: Stata 6, installed and working
- Schedule:
  - Lecture 1: August 20
  - Lecture 2: August 27
  - One-week break: September 2 through September 8
  - Lecture 3: September 10
  - Lecture 4: September 17
  - Closing discussion
  - Course ends: October 1

**NetCourse 101. An Introduction to Stata**
- 6 weeks (4 lectures)
- Course dates: September 17 through October 29
- Deadline for enrollment: September 13
- Cost: $85
- Course leaders: Mark Esman, Jeremy Wernow, and Kyle Willman
- Prerequisites: Stata 6, installed and working
- Schedule:
  - Lecture 1: September 17
  - Lecture 2: September 24
  - One-week break: September 30 through October 6
  - Lecture 3: October 8
  - Lecture 4: October 15
  - Closing discussion
  - Course ends: October 29

**NetCourse 151. Introduction to Stata programming**
- 6 weeks (4 lectures)
- Course dates: October 1 through November 12
- Deadline for enrollment: September 27
- Cost: $100
- Course leaders: David Drukker, Ken Higbee, and Allen McDowell
- Prerequisites: Stata 6, installed and working
- Schedule:
  - Lecture 1: October 1
  - Lecture 2: October 8
  - One-week break: October 14 through October 20
  - Lecture 3: October 22
  - Lecture 4: October 29
  - Closing discussion
  - Course ends: November 12

More information, including an outline of each course, can be obtained by
2. Clicking on the Headline Fall NetCourse schedule announced.

Email stata@stata.com for enrollment forms.
NetCourses are courses offered over the Internet via email then run about 6 weeks. Every Friday a “lecture” is emailed to the course participants. After reading the lecture, participants email questions and comments back to the Course Leaders. These emailed questions are remailed to all course participants by the NetCourse software. Course leaders respond to the questions and comments on Tuesday and Thursday. The other participants are encouraged to amplify or otherwise respond to the questions and comments as well. The next lecture is then emailed on Friday and process repeats.

The courses are designed to take roughly 3 hours per week.

All courses have been updated to Stata 6.

**dm66.1**  
**Stata 6 version of recoding variables using grouped values**

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

On the diskette accompanying this issue is a Stata 6 version of *cut* (see Clayton and Hills, 1999).

**Reference**


**dm68**  
**Display of variables in blocks**

Jeroen Weesie, Utrecht University, Netherlands, j.weesie@fss.uu.nl

Stata’s *list* command to display the values of variables provides two display styles. In the concise *tabular* (*nodisplay*) style, the default with “few” variables, *list* displays a “matrix” or “table” of values, optionally preceded with an observation number, and with the variable names in a header line above the table. If the output does not fit into the output device, Stata wraps each of the lines of the output. In the *display* style, the default with “many” variables, *list* lists the values of the variables per case in three columns, preceded with the variable name. The command *listblk* described in this insert provides an alternative style that lists as many variables as fit onto the output device for all cases, wrapping the variables rather than the observations.

The three styles are most easily shown and compared via an example, using Stata’s automobile data. If no style is specified, *list* selects the style it thinks is most appropriate.

```
    . list make price weight length mpg rep78 trunk displ foreign in 1/5
    Observation 1
          make        price     weight      length   mpg  rep78    trunk     displ  foreign
          AMC Concord     4099     2980          106      11       121     Domestic
          Observation 2
          make        price     weight      length   mpg  rep78    trunk     displ  foreign
          AMC Pacer      4749     3350           17      11       258     Domestic
          Observation 3
          make        price     weight      length   mpg  rep78    trunk     displ  foreign
          AMC Spirit     3799     2640           165      12       121     Domestic
          Observation 4
          make       Buick Centur. price     3250           106      20       196     Domestic
          length      mpg  rep78    trunk     displ  foreign
          Observation 5
          make       Buick Electr. price     4080           222      15       350     Domestic
          length      mpg  rep78    trunk     displ  foreign
```

This display style is quite liberal with output space, and lists the variable names for each observation. The *nodisplay* option produces more concise output.

```
    . list make price weight length mpg rep78 trunk displ foreign in 1/5, nodisplay
```
While more concise, it has become almost unreadable as well. The command `listblk` provides an alternative display style that may be a reasonable compromise between conciseness and readability.

```
. listblk make price weight length mpg rep/7/8 trunk displ foreign in /1///5
```

Note that `listblk` repeats the observation numbers so that it becomes possible to link related lines in the output. In cases with a meaningful case identifier, it may be more natural to repeat this variable in each table. This is accomplished with the option `repeat(#),` where `#` is the number of leading variables to be repeated.

```
. listblk make price weight length mpg rep/7/8 trunk displ foreign in /1///5, noobs rep(1)
```

Syntax
```
listblk [varlist] [if exp] [in range] [repeat(#) width(#) nolabel noobs]
```

Description
`listblk` displays the values of variables in `varlist` for selected cases in an alternative “blocks-of-variables” format. If no `varlist` is specified, the values of all the variables are displayed.

Options
`repeat(#)` specifies the number of leading variables repeated at the beginning of each block. The default is 0.
`width(#)` specifies the display linesize. By default, `listblk` uses the current setting of `display linesize`.
`nolabel` causes the numeric codes rather than label values to be displayed.
`noobs` suppresses printing of the observation numbers.

Acknowledgment
This project was supported by grant PGS 50-370 of the Netherlands Organization for Scientific Research.
Further new matrix commands

Syntax

matcfa argname1 argname2
matcfm argname1 argname2
matchk argname
matcname matrix1 matrix2
matdelrc matrix [ , row(rowexp) col(colexp) ]
matewd matrix1 matrix2 matrix3 [ , format(fmt) ]
matewm matrix1 matrix2 matrix3 [ , format(fmt) ]
matewop matrix1 matrix2 matrix3 , operator(op) [ , format(fmt) ]
matgop column_vector row_vector matrix , operator(op) [ , format(fmt) ]
matadw matrix1 matrix2
matmps matrix1 scalar matrix2 or matmps scalar matrix1 matrix2
matpow matrix new_matrix [ , format(fmt) power(#) iterate(#) tolerance(#) ]

Description

matcfa checks that

matrix new_matrix = argname1 + argname2

would execute correctly; that is, argname1 and argname2 name matrices with the same dimensions. matcfa is likely to be most useful within matrix programs.

matcfm checks that

matrix new_matrix = argname1 /* argname2

would execute correctly; that is, either both arguments are matrices such that the number of columns of argname1 equals the number of rows of argname2, or one of the two arguments is a scalar and the other is a matrix. If not, an error message will be issued. matcfm is likely to be most useful within matrix programs.

matchk checks that argname names an existing matrix. If not, an error message will be issued. matchk is likely to be most useful within matrix programs.

matcname gives matrix1 the row and column names of matrix2, provided that the two matrices have the same dimensions. If not, an error message will be issued.

Given a matrix, matdelrc deletes a specified row, or a specified column, or both. matdelrc will not delete (i.e. annihilate) entire row vectors or entire column vectors.

Given matrices A and B of the same order, matewd calculates and displays the matrix C having typical element the ratio

\[ c_{ij} = a_{ij} / b_{ij} \]

provided that no \( b_{ij} \) is equal to 0. C may overwrite A or B. A and B may be the same matrix.

Given matrices A and B of the same order, matewm calculates and displays the matrix C having typical element the product

\[ c_{ij} = a_{ij} \times b_{ij} \]

C may overwrite A or B. A and B may be the same matrix.

Given matrices A and B of the same order and a user-supplied binary operator \( op \), matewop calculates and displays the matrix C having typical element

\[ c_{ij} = a_{ij} \ op \ b_{ij} \]

provided that no \( c_{ij} \) would be missing. C may overwrite A or B. A and B may be the same matrix.
Given a column vector \( \mathbf{a} \) and a row vector \( \mathbf{b} \) and a user-supplied binary operator \( \text{op} \), \text{matgop} calculates and displays the matrix \( \mathbf{C} \) having typical element

\[
c_{ij} = a_i \text{ op } b_j
\]

provided that no \( c_{ij} \) would be missing.

\text{matmad} calculates for matrices \( \mathbf{A} \) and \( \mathbf{B} \) of the same order the maximum absolute difference between elements

\[
\max_{i,j} |a_{ij} - b_{ij}|
\]

\text{matmad} is likely to be most useful for checking the convergence of iterative matrix calculations within programs. The result is saved as \( r(\text{mad}) \).

\text{matmps} calculates the sum of a scalar and a matrix and places it in a second matrix. For a scalar \( x \) and a matrix \( \mathbf{A} \) the second matrix \( \mathbf{B} \) has typical element

\[
b_{ij} = x + a_{ij},
\]

the order of the arguments being immaterial. \text{matrix2} may overwrite \text{matrix1}.

Given a square matrix \( \mathbf{A} \) and power \( p \), \text{matpow} calculates and displays \( \mathbf{A}^p \), the \( p \)th power of \( \mathbf{A} \). The result is placed in a second matrix.

**Options**

\text{format}(fmt) controls the format with which the resulting matrix is printed. The default is \text{format}(%9.3f).

**Options unique to \text{matdelrc}**

\( \text{col}(\text{colexp}) \) specifies the column number. \( \text{colexp} \) should be or evaluate to an integer between 1 and the number of columns.

\( \text{row}(\text{rowexp}) \) specifies the row number. \( \text{rowexp} \) should be or evaluate to an integer between 1 and the number of rows.

**Option unique to \text{matewop} and \text{matgop}**

\( \text{operator}(\text{op}) \) specifies a binary operator and is a required option.

**Options unique to \text{matpow}**

\( \text{iterate}(\#) \) specifies the maximum number of iterations to attempt when powering until convergence. The default is 100.

\( \text{power}(\#) \) specifies the power. If not specified, the power is taken as effectively infinite: that is, powering is repeated until convergence (or until the limit imposed by \( \text{iterate}() \)).

\( \text{tolerance}(\#) \) is a technical option indicating the criterion for convergence. This is the largest acceptable absolute difference between each matrix element and that element on the previous iteration. The default is \( 1e-6 = 10^{-6} \).

**Remarks**

This set of programs is a suite: all should be installed together.

\text{matcfa}, \text{matcfm}, \text{matchk} and, to a lesser extent, \text{matcname} are essentially utilities for programmers that automate basic checking and management tasks. \text{matcname} may also be of use interactively.

\text{matewd}, \text{matewm}, \text{matewop}, \text{matgop} and \text{matmps} implement various matrix operations which go typically beyond mainstream matrix algebra and reflect a more general view of matrices, as implemented in various array-oriented languages such as APL.

\text{matmad} calculates one of many measures of the distance between two matrices. Another is \text{mreldif}(): see [U] 17.8.2 Matrix functions returning scalars. \text{matmad} is used within \text{matpow} to test for convergence.

An example of the use of \text{matpow} is powering a matrix of transition probabilities. In the examples below, data from Jeffers (1978) are used that give transition probabilities over 20 years between different vegetation types in a raised mire: bog, \textit{Calluna} (ling), woodland and grazed.

See also Weesie (1997) for another set of matrix programs complementing Stata’s built-in commands.
Examples

We create a three by three matrix:

```
. mat A = (1,2,3\4,5,6\7,8,9)
. mat li A

A[3,3]
c1 c2 c3
r1  1  2  3
r2  4  5  6
r3  7  8  9
```

Delete its second row:

```
. matdelrc A, r(2)
. mat li A

A[2,3]
c1 c2 c3
r1  1  2  3
r3  7  8  9
```

Now delete the second column of the new matrix:

```
. matdelrc A, c(2)
. mat li A

A[2,2]
c1 c3
r1  1  3
r3  7  9
```

Create a second matrix:

```
. mat B = (1,3\7,9)
. mat li B

B[2,2]
c1  c2
r1  1  3
r2  7  9
```

Element-wise division:

```
. matewd A B C

symmetric C[2,2]
c1  c2
r1  1.00
r2  1.00  1.00
```

Element-wise multiplication:

```
. matewm A B C

C[2,2]
c1  c2
r1  1.00  9.00
r2  49.00  81.00
```

Element-wise comparison:

```
. matewop A B C, o(>)

symmetric C[2,2]
c1  c2
r1  0.00
r2  0.00  0.00
```

Now do the outer product of a vector with itself transposed:

```
. mat int = (1,2,3,4,5,6,7,8,9,10,11,12)
. mat li int

int[12,12]
c1  c2  c3  c4  c5  c6  c7  c8  c9  c10  c11  c12
r1  1  2  3  4  5  6  7  8  9  10  11  12
```

```
. mat intt = int'
. matgop intt int multtab, o(*)
```
symmetric multtab[12,12]

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>c4</th>
<th>c5</th>
<th>c6</th>
<th>c7</th>
<th>c8</th>
<th>c9</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td>2.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td>3.00</td>
<td>6.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td>4.00</td>
<td>8.00</td>
<td>12.00</td>
<td>16.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td>5.00</td>
<td>10.00</td>
<td>15.00</td>
<td>20.00</td>
<td>25.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r6</td>
<td>6.00</td>
<td>12.00</td>
<td>18.00</td>
<td>24.00</td>
<td>30.00</td>
<td>36.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r7</td>
<td>7.00</td>
<td>14.00</td>
<td>21.00</td>
<td>28.00</td>
<td>35.00</td>
<td>42.00</td>
<td>49.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r8</td>
<td>8.00</td>
<td>16.00</td>
<td>24.00</td>
<td>32.00</td>
<td>40.00</td>
<td>48.00</td>
<td>56.00</td>
<td>64.00</td>
<td></td>
</tr>
<tr>
<td>r9</td>
<td>9.00</td>
<td>18.00</td>
<td>27.00</td>
<td>36.00</td>
<td>45.00</td>
<td>54.00</td>
<td>63.00</td>
<td>72.00</td>
<td>81.00</td>
</tr>
<tr>
<td>r10</td>
<td>10.00</td>
<td>20.00</td>
<td>30.00</td>
<td>40.00</td>
<td>50.00</td>
<td>60.00</td>
<td>70.00</td>
<td>80.00</td>
<td>90.00</td>
</tr>
<tr>
<td>r11</td>
<td>11.00</td>
<td>22.00</td>
<td>33.00</td>
<td>44.00</td>
<td>55.00</td>
<td>66.00</td>
<td>77.00</td>
<td>88.00</td>
<td>99.00</td>
</tr>
<tr>
<td>r12</td>
<td>12.00</td>
<td>24.00</td>
<td>36.00</td>
<td>48.00</td>
<td>60.00</td>
<td>72.00</td>
<td>84.00</td>
<td>96.00</td>
<td>108.00</td>
</tr>
</tbody>
</table>

Finally, we form the transition matrix and raise it to successively higher powers:

```
. mat P = (.65,.29,.06, 0, .03,.33,.07,.02, .08,.69,.03)
. mat rownames P = Bog Calluna Woodland Grazed
. mat colnames P = Bog Calluna Woodland Grazed
. mat li P
```

```
P[4,4]  
Bog  Calluna  Woodland  Grazed
Bog  .65  .29  .06  0
Calluna  .3  .33  .3  .07
Woodland  0  .28  .69  .03
Grazed  0   .4  .2  .4
```

```
P2[4,4]  
Bog  Calluna  Woodland  Grazed
Bog  .510  .301  .167  .022
Calluna  .294  .308  .338  .060
Woodland  .084  .298  .566  .052
Grazed  .120  .348  .338  .194
```

```
P3[4,4]  
Bog  Calluna  Woodland  Grazed
Bog  .421  .303  .241  .036
Calluna  .363  .306  .355  .056
Woodland  .144  .302  .495  .059
Grazed  .182  .322  .384  .112
```

```
P4[4,4]  
Bog  Calluna  Woodland  Grazed
Bog  .365  .304  .289  .042
Calluna  .276  .305  .365  .054
Woodland  .184  .304  .433  .059
Grazed  .215  .311  .395  .079
```

```
Pinf[4,4]  
Bog  Calluna  Woodland  Grazed
Bog  .361  .304  .380  .055
Calluna  .361  .304  .380  .055
Woodland  .361  .304  .380  .055
Grazed  .361  .304  .380  .055
```

Acknowledgments

William Gould sketched the main idea of `matdelrc` and made very helpful comments on previous versions.
References

Syntax

\texttt{egen \ [type] \ newvar = \ fcn(stuff) \ [if \ exp] \ [in \ range] \ , \ options}

Description

This insert describes 24 additional functions for \texttt{egen}.

Help for these extra functions has been placed in a file \texttt{egenodd.hlp}. Therefore, to obtain on-line help, issue either \texttt{help egenodd} or \texttt{whelp egenodd}.

\texttt{egen} creates \texttt{newvar} of the optionally specified storage type equal to \texttt{fcn(stuff)}. Depending on \texttt{fcn()}, \texttt{stuff} refers to an expression, a \texttt{varlist}, or a \texttt{numlist}, and the options are similarly function-dependent.

Note that \texttt{egen} may change the sort order of your data.

The new functions

\texttt{any(varname)}, \texttt{values(integer numlist)} is \texttt{varname} if \texttt{varname} is equal to any of the integer values in a supplied \texttt{numlist}, and missing otherwise. See also \texttt{eqany(varlist)} and \texttt{neqany(varlist)}.

\texttt{atan2(sinevar, cosinevar)}[, radians] supplies arctangent of \texttt{sinevar/cosinevar} as an angle between 0 and 360 degrees, or optionally between 0 and 2\pi radians.

\texttt{concat(varlist)}[, format(fmt)] \texttt{decode maxlength(#)} \texttt{punct(pchars)} concatenates \texttt{varlist} to produce a string variable. Values of string variables are unchanged. Values of numeric variables are converted to string as is or converted using a format under option \texttt{format(fmt)} or decoded under option \texttt{decode}, in which case \texttt{maxlength()} may also be used to control the maximum length of label used. By default, variables are added end-to-end: \texttt{punct(pchars)} may be used to specify punctuation, such as a space, \texttt{punct(" ")}, or a comma, \texttt{punct(,)}.

\texttt{eqany(varlist)}, \texttt{values(integer numlist)} is 1 if any of the variables in \texttt{varlist} is equal to any of the integer values in a supplied \texttt{numlist}, and 0 otherwise. See also \texttt{any(varname)} and \texttt{neqany(varlist)}.

\texttt{head(strvar)}[, punct(pchars) trim] gives the first word of string variable \texttt{strvar}. Given \texttt{pchars}, by default a single space " ", the head is whatever precedes the first occurrence of \texttt{pchars}, or the whole of the string if it does not occur. \texttt{head()} applied to "frog, toad" is "frog" and to "frog" is "frog". \texttt{head()} applied to "frog, toad" is similarly "frog" with \texttt{pchars}.

The trim option trims any leading or trailing spaces. See also \texttt{last(strvar)} and \texttt{tail(strvar)}.

\texttt{kurt(varname)}[, by(byvarlist)] returns the kurtosis of \texttt{varname}.

\texttt{last(strvar)}[, punct(pchars) trim] gives the last 'word' of string variable \texttt{strvar}. Given \texttt{pchars}, by default a single space " ", the last word is whatever follows the last occurrence of \texttt{pchars}, or the whole of the string if it does not occur. \texttt{last()} applied to "frog, toad, newt" is "newt" and to "frog" is "frog". \texttt{last()} applied to "frog, toad" is similarly "toad" with \texttt{pchars}.

The trim option trims any leading or trailing spaces. See also \texttt{head(strvar)} and \texttt{tail(strvar)}.

\texttt{lgrouplevels(varname)}[, missing] returns integers from 1 and higher according to the distinct groups of \texttt{varname} in sorted order. Integers will be labeled with the values of \texttt{varname}, or its value labels if such exist. This is useful as an alternative to \texttt{group()} when labels are needed as well as the bare integer codes. \texttt{missing} indicates that missing values in \texttt{varname} are to be treated like any other value when assigning groups, instead of missing values being assigned to the group missing.

\texttt{mad(exp)}[, by(byvarlist)] returns the median absolute deviation from the median of \texttt{exp}.

\texttt{mdev(exp)}[, by(byvarlist)] returns the mean absolute deviation from the mean of \texttt{exp}.
mode(varname) [, minmode unique missing by(byvarlist) ] produces the mode for varname, which may be numeric or string. The mode is the value occurring most frequently. If two or more modes exist, the mode produced will be the highest such value (largest numerically or last alphabetically), except that \text{minmode} specifies use of the lowest such value and \text{unique} specifies that only unique modes may be produced. Missing values are excluded from determination of the mode unless \text{missing} is specified. Even so, the value of the mode is recorded for observations for which the values of varname are missing unless explicitly excluded, e.g., by \text{if varname < .} or if \text{varname != ""}. by \text{byvarlist} specifies that determination is to be carried out separately for distinct groups defined by \text{byvarlist}.

\text{neqany(varlist), values(integer numlist)} contains for each observation the number of variables in varlist for which values are equal to any of the integer values in a supplied numlist. See also \text{any(varname)} and \text{eqany(varlist)}.

\text{pc(exp) [ , by(byvarlist) ]} returns \text{exp} scaled to be a percent of total, between 0 and 100. See also \text{prop(exp)}.

\text{pp(varname) [ , by(byvarlist) a(#) ]} sorts varname smallest to largest and computes the corresponding plotting position \((i-a)/(n-2a+1)\) for \(i = 1, \ldots, n\) (largest) and constant \(a\). The default \(a = 0.5\) yields \((i-0.5)/n\), while \(a = 0\) yields \(i/(n+1)\).

\text{prop(exp) [ , by(byvarlist) ]} returns \text{exp} scaled to be a proportion of total, between 0 and 1. See also \text{pc(exp)}.

\text{rev(varname) [ , by(byvarlist) ]} returns the reverse of varname, that is, varname[\text{N}]) is exchanged with varname[1], and so forth.

\text{rindex(strvar), substr(string) returns the index of the last (rightmost) occurrence of string in the string variable strvar.}

\text{rmed(varlist) returns the median across variables for each observation. (The number of variables must not exceed the number of observations.)}

\text{rotate(varname) [ , start(#) max(#) ]} rotates a set of integers 1, \ldots, max. Suppose we have months 1, \ldots, 12 and we wish to map 7 to 1, 8 to 2, \ldots, 12 to 6, 1 to 7, \ldots, 6 to 12. This would be achieved by start(7) max(12).

\text{seq() [ , from(#) to(#) block(#) by(byvarlist) ]} returns integer sequences. Values start from \text{from} (default 1) and increase to \text{to} (default the maximum number of values) in blocks (default size 1). If \text{to} is less than the maximum number, sequences restart at \text{from}. Numbering may also be separate within groups defined by \text{byvarlist}, or decreasing if \text{to} is less than \text{from}. Sequences depend on the sort order of observations, following three rules: (1) observations excluded by \text{if} or \text{in} are not counted, (2) observations are sorted by \text{byvarlist}, if specified, (3) otherwise, the order is that when called. Note that no \text{stuff} is specified.

\text{skew(varname) [ , by(byvarlist) ]} returns the skewness of varname.

\text{sub(strvar), find(findstr) [ replace(replacestr) all word ]} replaces occurrences of \text{findstr} by \text{replacestr} in the string variable \text{strvar}. By default only the first such occurrence in each string value is acted upon. \text{all} specifies that all occurrences in each string value are to be acted upon. If \text{replacestr} is not specified, it is taken to be empty, that is, \text{findstr} is deleted. \text{word} specifies that only occurrences of \text{findstr} that are complete words are to be acted upon.

\text{tag(varlist) [ , missing ]} tags just one observation in each distinct group defined by \text{varlist}. When all observations in a group have the same value for a summary variable calculated for the group, it will be sufficient to use just one such value for many purposes. The result will be 1 or 0, according to whether the observation is tagged, and never missing: hence if \text{tag} is the variable produced by \text{egen tag = tag(varlist)} the idiom \text{if tag} is always safe. \text{missing} specifies that missing values of varlist may be included.

\text{tail(strvar) [ , punct(pchars) trim ]} gives the remainder of string variable \text{strvar}. Given \text{pchars}, by default a single space " ", the tail is whatever follows the first occurrence of \text{pchars}, which will be the empty string " " if it does not occur. \text{tail()} applied to "frogs toad" is "toad" and to "frogs" is "." . \text{tail()} applied to "frogs toad" is similarly "toad" with \text{punct(,)}. The \text{trim} option trims any leading or trailing spaces. See also \text{head(strvar)} and \text{last(strvar)}.

\text{Remarks}

The official Stata command \text{egen} (see \text{[R] egen}) is a driver for a set of functions, each defined by a separate program in an ado-file. The principle is that a command \text{egen newvar = fcn(stuff)} embodies a call to \text{egen} function \text{fcn}, itself defined by program \text{.gcn}. Hence the functions of \text{egen} are defined by programs written in the Stata language, in contrast to the functions used by \text{generate}, which are all part of the Stata executable (see \text{[U] 16 Functions and expressions}). Hence Stata programmers may add \text{egen} functions to those supplied by official Stata. Examples previously published in the STB are the \text{rmiss2()} function of Goldstein (1995) and the \text{cut()} function of Clayton and Hills (1999). The \text{egen} functions in official Stata supply the interested programmer with useful templates to modify: often only a few lines in your program need differ from an already written program.
The advantage of an \texttt{egen} function is convenience for interactive use. In contrast, it is not usually a good idea to issue \texttt{egen} commands within Stata programs. If this is done, \texttt{egen} spends much of its time doing housekeeping work which should typically be unnecessary within a Stata program. A program that uses the bare minimum of commands will thus be faster than the same program with a corresponding call to \texttt{egen}. However, this is a counsel of perfection; for occasional use, or if you are a novice Stata programmer, the loss of machine time in using \texttt{egen} may be much smaller than the time you might take to modify a program so that it does not use \texttt{egen}.

More positively, it is worth flagging a change made to \texttt{egen} in Stata 6. \texttt{egen} may now produce string variables as results. Note, however, that it remains true that the default variable type produced by \texttt{egen} is that defined by \texttt{set type} (see \cite{R:generate}). In turn, by default this is \texttt{float}.

Users of \texttt{generate} will know that it is essential when generating a string variable to specify a string data type. If this habit is ingrained, you need not unlearn it when using the string functions included in this insert. However, these functions implement what is intended as a smart approach. They always generate the string variable of the smallest string type that will do the job required. You might specify a string type that is too small, just right or too big; you might specify any data type whatever, including even a numeric data type; or you might not specify any data type. In all these cases, the functions that produce string results are written on the basis that Stata can work out the best string data type required, so that whatever you might specify will be ignored. What happens is that within each \texttt{egen} program, a \texttt{str1} variable is first generated, and then replaced by the result required. Stata automatically promotes the variable to whatever string type is needed.

\section*{Explanations and examples}

The \texttt{egen} functions discussed here fall into various classes.

\subsection*{Functions for string variables}

\begin{itemize}
  \item \texttt{concat(varlist)}
  \end{itemize}

Concatenation of string variables is already provided in Stata. In context, Stata understands the addition symbol $+$ as specifying concatenation, adding strings end to end. \texttt{"soft" + "ware"} produces \texttt{"software"} and, given string variables \texttt{s1} and \texttt{s2}, \texttt{s1 + s2} indicates their concatenation.

The complications that may arise in practice include (1) wanting to concatenate the string versions of numeric variables and (2) wanting to concatenate variables, together with some separator such as a space or a comma. Given numeric variables \texttt{n1} and \texttt{n2}

\begin{verbatim}
  . gen str1 newstr = ""
  . replace newstr = s1 + string(n1) + string(n2) + s2
\end{verbatim}

shows how numeric values may be converted to their string equivalents before concatenation and

\begin{verbatim}
  . replace newstr = s1 + " " + s2 + " " + s3
\end{verbatim}

shows how spaces may be added in between variables. Here, as often happens, it is supposed that we would rather let Stata work out the particular string data type required. That is, we first \texttt{generate} a variable of type \texttt{str1}, the most compact string type; then the \texttt{replace} command automatically leads to promotion of the variable to the appropriate data type.

If all this is already possible, why then introduce \texttt{concat()'?} \texttt{concat()} allows you to do everything in one line in a very concise manner.

\begin{verbatim}
  . egen newstr = concat(s1 n1 n2 s2)
\end{verbatim}

carries with it an implicit instruction to convert numeric values to their string equivalents, and the appropriate string data type is worked out within \texttt{concat()} by Stata's automatic promotion. Moreover,

\begin{verbatim}
  . egen newstr = concat(s1 s2 s3), p("")
\end{verbatim}

specifies that spaces are to be used as separators. (The default is no separation of concatenated strings.)

As an example of punctuation other than a space, consider

\begin{verbatim}
  . egen fullname = concat(surname forename), p(", ")
\end{verbatim}

Non-integer numerical values can cause difficulties, but

\begin{verbatim}
  . egen newstr = concat(n1 n2), format(%9.3f) p(" ")
\end{verbatim}

specifies the use of format \texttt{\%9.3f}. In other words, this is equivalent to
The main subtlety is that these functions are literal, so that the tail of "frog toad newt", in which two spaces follow "frog", includes the second of those spaces, and is thus " toad newt". Therefore you may prefer to use the trim() option to trim the result of any leading and/or trailing spaces, producing in this instance "toad newt".

The punct(pchars) option may be used to specify separators other than spaces. The general definitions of head(), tail() and last() are therefore in terms of whatever separator has been specified, that is, relative to the first or last occurrence of the separator in the string value. Thus with punct(,;) and the string "Darwin, Charles Robert" the head is "Darwin" and the tail and the last are both " Charles Robert". Note again the leading space in this example, which may be trimmed with trim(). The punctuation (here the comma,) is discarded, just as it is with a single space.

pchars, the argument of punct(), will usually, but not necessarily, be a single character. If two or more characters are specified, then these must occur together; punct(;;) would mean that words are separated by a colon followed by a semi-colon (that is ;;;). It is not implied, in particular, that the colon and semi-colon are alternatives: for that the user must modify the programs presented here or resort to first principles using tokenize (see [R] tokenize).

With personal names head() or last() might be applied to extract surnames if strings were like "Darwin, Charles Robert" or "Charles Robert Darwin" with the surname coming first or last. What then happens with surnames like "von Neumann" or "de la Mare"? "von Neumann, John" is no problem, if the comma is specified as a separator, but last() does not contain enough intelligence to handle "Walter de la Mare" properly. For that, the best advice is to use programs specially written for person name extraction, such as extrname (Gould 1993).

rindex(strvar)

The existing function index() finds the position of the first occurrence of a specified substring within a string. Thus index("Stata", "a") returns 3, because the first occurrence of "a" in "Stata" starts at that position. Similarly index("Stata", "at") is also 3.

rindex() (read right index if you wish) returns the position of the last occurrence of a substring. Thus if "Stata" were the value of a string variable, then egen rindex = rindex(varname), sub(a) would return 5 in that case.

As with index(), if a substring does not occur within the string value, then rindex() returns 0.

Note, however, that rindex() is not a perfect analogue of index(). index() can be applied to both variables and
individual strings. \texttt{rindex()} may only be applied to string variables.

- \texttt{sub(strvar)}

  \texttt{sub()} substitutes occurrences of one substring with another. An important special case, which is in fact the default, is that the replacement substring is empty: that is, the substring is deleted.

  \texttt{sub()} applies the extended macro function \texttt{subinstr} (see \citep{Stata:2013} \texttt{21.3.6 Extended macro functions}) to each observation in turn. As a consequence it may be rather slow.

  If we wished to delete commas from a string variable, we would use

  \begin{verbatim}
  . egen nocomma = sub(myvar), f(,)
  \end{verbatim}

  while if we wished to replace them with spaces, we would have

  \begin{verbatim}
  . egen nocomma = sub(myvar), f(,) r(" ")
  \end{verbatim}

  Note the use of \texttt{all} to specify that all commas, not just the first found in each string value, should be replaced.

  An important restriction in many problems is to work only on complete words. If we wished to replace \texttt{"man"} with \texttt{"male"}, then the option \texttt{word} would prevent \texttt{"woman"} becoming \texttt{"womale"}.

Functions for categorical and integer variables

- \texttt{any(varname)}, \texttt{eqany(varlist)} and \texttt{neqany(varlist)}

  \texttt{any()}, \texttt{eqany()} and \texttt{neqany()} are for categorical or other variables taking integer values. If we define a subset of values specified by an integer \texttt{numlist} (see \citep{Stata:2013} \texttt{14.1.8 numlist}), then \texttt{any()} extracts the subset, leaving every other value missing, \texttt{eqany()} defines an indicator variable (1 if in subset, 0 otherwise), and \texttt{neqany()} counts occurrences of the subset across a set of variables. Therefore, with just one variable \texttt{eqany(varname)} and \texttt{neqany(varname)} are equivalent.

  With the auto data supplied with official Stata, we can generate a variable containing the high values of \texttt{rep78} and a variable indicating whether \texttt{rep78} has a high value:

  \begin{verbatim}
  . egen hirep = any(rep78), v(3/5)
  . egen ishirep = eqany(rep78), v(3/5)
  \end{verbatim}

  In this case, it is easy to produce the same results with official Stata commands:

  \begin{verbatim}
  . gen hirep = rep78 if rep78 == 3 | rep78 == 4 | rep78 == 5
  . gen byte ishirep = rep78 == 3 | rep78 == 4 | rep78 == 5
  \end{verbatim}

  but as the specification becomes more complicated, or involves several variables, the \texttt{egen} functions here may be more convenient.

- \texttt{lgroup(varname)}

  The existing \texttt{egen} function \texttt{group()} maps the distinct groups of a \texttt{varlist} to a categorical variable that takes on integer values from 1 to the number of groups. The order of the groups is that of the sort order of \texttt{varlist}. The \texttt{varlist} may be of numeric variables, string variables, or a mixture. The resulting variable can be useful for many purposes, including stepping through the distinct groups in an easy and systematic manner and tidying up an untidy ordering. Suppose the actual (and arbitrary) codes present in the data are 1, 2, 4 and 7, but we desire equally spaced numbers, as when the codes will be values on one axis of a graph. \texttt{group()} will map these to 1, 2, 3 and 4.

  The resulting variable does not have value labels. Therefore the values from 1 upwards carry no indication of meaning. Interpretation requires comparison with the original \texttt{varlist}.

  \texttt{lgroup()} produces a categorical variable in the same manner as \texttt{group()}, but with value labels. These value labels are either the actual values of \texttt{varname}, or any value labels of \texttt{varname}, if they exist. The values of \texttt{varname} could be as long as those of a single \texttt{str80} variable, yet value labels may be no longer than 80 characters. Thus \texttt{lgroup()} is restricted to taking a single variable.

  The \texttt{missing} option behaves in the same way as that of \texttt{group()}. The default is that missing maps to missing. If missing values are to be treated like any other, and given an integer code, then specify \texttt{missing}.

- \texttt{rotate(varname)}

  The idea behind \texttt{rotate()} is perhaps best explained by a specific example. A rainfall time series with a strong seasonal component is collected for each month over several years for a station in the Northern hemisphere. The months are coded 1 (January) through 12 (December). The scale, however, is clearly circular, and January follows December just as February follows January. A plot of rainfall against month shows a pattern of a dry summer (driest around July) and a wet winter (wettest
around January). But the conventional coding means that on this plot the interesting wet season is split between the left-hand and right-hand edges of the graph. It would be better to have the month axis extend from (say) July to June. Such a rotation would reunite the two parts of the wet season, and is accomplished by

```stata
egen newmonth = rotate(month), st(7) max(12)
```

indicating that the new start is 7 and the scale wraps around after the maximum value of 12.

A special feature of `rotate()` is that any value labels associated with the variable to be rotated will be rotated to match. Thus `newmonth` has July coded as 1, in contrast to `month` which had July coded as 7. If `month` had value label "July" associated with value 7, then `newmonth` will have the same value label associated with value 1.

- **seq()**

  `seq()` creates a new variable containing one or more sequences of integers. It is principally useful for quick creation of observation identifiers or automatic numbering of levels of factors or categorical variables. `seq()` is based on the separate command `seq` (Cox 1997), but one notable detail has been changed, as noted at the end of this section.

  In the simplest case

  ```stata
egen a = seq()
```

  is just equivalent to the common idiom

  ```stata
gen a = _n
```

  `a` may also be obtained from

  ```stata
range a 1 \_N
```

  (the actual value of \_N may also be used).

  In more complicated cases, `seq()` with option calls is equivalent to nimble fingerwork with those versatile functions `int` and `mod`.

  ```stata
egen b = seq(), b(2)
```

  produces integers in blocks of 2, while

  ```stata
egen c = seq(), t(6)
```

  restarts the sequence after 6 is reached.

  ```stata
egen d = seq(), f(10) t(12)
```

  shows that sequences may start with integers other than 1, and

  ```stata
egen e = seq(), f(3) t(1)
```

  shows that they may decrease.

Suppose we have 12 observations in memory. The results of these commands are shown by

```stata
list a b c d e
```

and are

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>

All these sequences could have been generated in one line with `generate` and use of `int` and `mod` functions. The variables `b` through `e` are obtainable by
In general, the sequences produced depend on the sort order of observations, following three rules:

1: observations excluded by if or in are not counted.
2: observations are sorted by byvarlist, if specified.
3: otherwise, the order is that when called.

Note that seq (Cox 1997) did not use Rule 3. The consequence was that the result of applying seq was not guaranteed identical from application to application whenever sorting was required, even with identical data, because of the indeterminacy of sorting. That is, if we sort (say) integer values, it is sufficient that all the 1s are together and followed by all the 2s. But there is no guarantee that the order of the 1s, as defined by any other variables, will be identical from sort to sort.

The existing egen function filll() offers an alternative to seq() (see [R] egen). In essence, filll() requires a minimal example that indicates the kind of sequence required, whereas seq() requires the rule to be specified through options. There are sequences that filll() can produce that seq() cannot, and vice versa. filll() cannot be combined with if or in, in contrast to seq().

Functions for statistical or quantitative analysis

- pp (varname)
  pp() calculates plotting positions. These are so called because of their role in distributional diagnostic plots (see [R] diagplots). If data are ranked so that \( x_{(1)} \ldots x_{(n)} \) are in ascending order, a general form that includes essentially all plotting positions used in practice is \( (i - a)/(n - 2a + 1) \) for \( i = 1, \ldots, n \) and some constant \( a \). This provides, in essence, an estimate of the proportion of data less than or equal to \( x_{(i)} \). Popular choices for \( a \) are 0.5, suggested by Hazen, and wired into the official Stata command quantile, and 0, suggested by Weibull and Gumbel, and wired into the official Stata commands pnorm, qnorm, pchi, and qchi.

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

  The purpose therefore of pp() is to provide Stata users with a general tool for choosing plotting positions, making it easier to produce customized distributional diagnostic plots, especially for probability distributions not allowed for in official Stata.

  The option by() allows calculation of plotting positions to proceed separately for different groups, as in

  \[ .
  \text{egen pmpg = pp(mpg), by(rep78) a(0)}
  \]

- mode (varname)

  The mode is the most common value of a dataset. This idea can be applied to numeric and string variables alike. It is perhaps most useful for categorical variables (whether defined by integers or strings) or for other integer-valued values, but mode() can be applied to variables of any type. Nevertheless, the modes of continuous (or nearly continuous) variables are perhaps better estimated either from inspection of a graph of a frequency distribution or from the results of some density estimation (see [R] kdensity).

  A key question is what to do if two or more values are equally common. The somewhat arbitrary default of mode() is that the highest such value will be reported. Highest means highest in sort order, whether numeric for numeric variables, or alphabetic for string variables. The opposite convention, to report the lowest, is obtained by the minmode option. A more stringent convention that only unique modes be reported is obtained by the unique option. With that stringent convention, there might not be a unique mode, in which case the result produced will be a missing value.
Missing values need special attention. It is very possible that missing (whether the period \( . \) for numeric variables or the empty string "" for string variables) is the most common value in a variable. However, missing values are by default excluded from determination of modes. If you wish to include them, use the `missing` option.

In contrast, `egen mode = mode(varname)` allows the generation of nonmissing modes for observations for which `varname` is missing. This allows use of the mode as one simple means of imputation for categorical variables. If it is desired that the mode is missing whenever `varname` is missing, that is readily achieved by specifying `if varname < .` or `if varname != ""` or, most generally, `if !missing(varname)`.

- `rmed(varlist)`

  `rmed()` resembles, for example, the existing `egen` function `rmean()`, but it produces medians across variables rather than means. The procedure used to calculate the median depends on placing all the values in a single observation (row of the data) into a temporary variable (column of the data). Therefore it is essential that the number of variables in the data be no greater than the number of observations. (If it is greater, then increasing the number of observations may be a way forward, assuming that sufficient memory is available. See [R] `obs`.)

- `mad(exp)` and `mdev(exp)`

  `mad()` and `mdev()` produce alternative measures of spread. The median absolute deviation from the median and even the mean deviation will both be more resistant than the standard deviation to heavy tails or outliers, in particular from distributions with heavier tails than the normal or Gaussian. The first measure was named the MAD by Andrews et al. in 1972, but known already to K.F. Gauss in 1816, according to Hampel et al. (1986). For further historical and statistical details, see David (1998).

- `skew(varname)` and `kurt(varname)`

  `skew()` and `kurt()` put the results of skewness and kurtosis calculations from `summarize` (see [R] `summarize`) into new variables.

  Most commonly with such functions, we might wish to analyze variations in spread or shape measures between groups, so that typically the `by()` option will be used to produce values for distinct groups. But then the same MAD or skewness, for example, applies to every value in each group and the variable contains redundant information. In these circumstances, many tasks need use only one such value from each group.

- `tag(varlist)`

  The answer to that need is `tag()`. `tag()` tags just one observation in each distinct group with 1 and all others with 0. Missing is never produced as a result, even for observations excluded by an `if` or `in` condition. This allows the user to be safe with idioms such as `if tag`, (`if tag` is a contraction of `if tag != 0`, which is always the same as `if tag == 1`, which is turn is always the same as `if tag`, when applied to the results of `tag()`.)

  As an illustration, suppose we want a plot of group kurtosis versus group skewness. 100,000 values are divided into 100 groups.

    . `egen skew = skew(myvar)`, by(group)
    . `egen kurt = kurt(myvar)`, by(group)
    . `egen tag = tag(group)`
    . `graph kurt skew if tag`

  This sequence of commands ensures that the graph is based on 100 data points, not 100,000 data points, each 1,000 of which is identical.

  Evidently the same approach can be used with any other calculations of group summaries, whether with `egen` functions discussed here or with other Stata commands or programs.

  A programming detail is that since groups might be as small as 1 in number, there are two possible approaches, to tag the first or the last in each group. `tag()` tags the first. This should be immaterial, but just in case it is important to you, note that which value is taken as first may not be identical, even with the same data, from application to application, because of the indeterminancy associated with sorting.

- `atan2(sinevar cosinevar)`

  Official Stata already has an `atan()` function (see [U] 16.3.1 Mathematical functions). `atan()` takes a single argument and returns a result in radians, between \(-\pi/2\) and \(\pi/2\), namely a range of half the circle. `atan2()` takes two variables as arguments, which we may call a sine variable and a cosine variable, as in

    . `egen atan = atan2(sinevar cosinevar)`
It returns a result that is by default in degrees and between 0° and 360°. The option `radians` specifies a result in radians.

The choice of default range, 0° to 360° not −180° to 180°, and of default units, degrees not radians, arises from conventions in the statistical analysis of circular data, where these are the standard ways of expressing both data and results (see, for example, Fisher 1993).

- `pc(exp)` and `prop(exp)`
  - `pc()` and `prop()` produce percents (which sum to 100) and proportions (which sum to 1).

- `rev(varname)`
  - `rev()` is applicable to any variable, but perhaps most likely to be used with some quantitative variable which we want to reverse for some reason, making the first last, and *vice versa*.

### Acknowledgments

Several members of Statalist contributed by posting problems or commenting on previous versions of various programs.

### References


---

**Gr38 Enhancement to the hilite Command**

Jeroen Weesie, Utrecht University, Netherlands, j.weesie@fss.uu.nl

For the visualization of more-than-two-dimensional data on a two-dimensional computer display, I find crosstabs in which points that satisfy some condition are marked (highlighted) quite useful. Stata provides the `hilite` command for this purpose. In my work, I feel often bothered that `hilite` is able to hilite a single expression only. Trying to get some understanding of three- or higher-dimensional structure is already hard enough for me to want to glance at different plots obtained by subsequent calls of `hilite`; I rather want to look at all these plots at the same time. Of course, printing is sometimes an option. The `forgraph` command (see Weesie 1999) is also sometimes of use.

Frustration with existing constraints often being the source of innovation, I wrote an extension to `hilite`, called `hilite2`, that reduces some of my frustrations; the frustrations that remain may well induce many future submissions to the *Stata Technical Bulletin*. I hope that some readers may feel the same, that is, a minor relief after glancing through this command. As far as I know, `hilite2` is backward compatible with `hilite`.

### Syntax

```
hilite2 yvar xvar [if] [in range], { hilite(exp-list) | hivar(varlist) } [miss { overlay | matrix } margin(#) symbol(str) nolabel saving(filename) title(str) bsize(#) graph_options ]
```

---

**Gr38 Enhancement to the hilite Command**

Jeroen Weesie, Utrecht University, Netherlands, j.weesie@fss.uu.nl

For the visualization of more-than-two-dimensional data on a two-dimensional computer display, I find crosstabs in which points that satisfy some condition are marked (highlighted) quite useful. Stata provides the `hilite` command for this purpose. In my work, I feel often bothered that `hilite` is able to hilite a single expression only. Trying to get some understanding of three- or higher-dimensional structure is already hard enough for me to want to glance at different plots obtained by subsequent calls of `hilite`; I rather want to look at all these plots at the same time. Of course, printing is sometimes an option. The `forgraph` command (see Weesie 1999) is also sometimes of use.

Frustration with existing constraints often being the source of innovation, I wrote an extension to `hilite`, called `hilite2`, that reduces some of my frustrations; the frustrations that remain may well induce many future submissions to the *Stata Technical Bulletin*. I hope that some readers may feel the same, that is, a minor relief after glancing through this command. As far as I know, `hilite2` is backward compatible with `hilite`.

### Syntax

```
hilite2 yvar xvar [if] [in range], { hilite(exp-list) | hivar(varlist) } [miss { overlay | matrix } margin(#) symbol(str) nolabel saving(filename) title(str) bsize(#) graph_options ]
```
Options

`hilite(exp-list)` specifies a list of expressions, separated by blanks, to be highlighted. The expressions should not contain embedded spaces.

`hivar(varlist)` specifies a `varlist` so that observations with the same values for variables in `varlist` are highlighted. The variables may be numeric or string-typed.

`miss` specifies that observations with missing values of the `hivar-varlist` should be treated as a separate hilite group.

`overlay` specifies that the highlights for all expressions should be plotted in a single ‘overlay’ plot. In an overlay plot, at most 6 expressions should be implied by `hilite` or `hivar`.

`matrix` specifies that a matrix plot is produced of separate hilite plots for each explicit or implicit expression. For readability, vertical and horizontal labels are only displayed in the left-most and down-most plots. At most, 49 expressions should be implied by `hilite` or `hivar`.

`margin(#)` specifies the margin for a matrix-style plot.

`symbol(str)` specifies the symbols used to highlight expressions (see `symbols()` in online help for `graph`). If a matrix plot is produced, `str` should contain at most 2 characters. In an overlay plot, the number of characters in `str` should equal the number of expressions to be highlighted.

`nolabel` specifies that the titles describing the highlighting expressions for `hivar`-induced `hilite` groups ignore value labels.

This may be useful if the resulting expressions are too long.

`saving(filename)` specifies the overlay plot or the matrix plot should be saved in a file named `filename`.

`title(str)` specifies the title displayed in the combined matrix plot.

`bsize(#)` specifies the plot size of the title describing the `hilite` expression.

`graph_options` are any of the options allowed with `graph, twoway`; see online help for `graph`. In the case of a matrix plot, these options are applied to each of the two-way scattergrams, not to the combined plot. Even `by()` is permitted but usually leads to an ugly plot.

Examples

First, `hilite2` allows highlighting of multiple sets of points, identified by a sequence of expressions. Expressions should be separated by white space, and thus should not contain embedded blanks. For example,

```
.hilite2 price mpg, hilite(rep78==1) rep78==2 rep78==3 rep78==4 rep78>4
```

produces Figure 1. (As a technical aside, `hilite2` parses off the option `hilite` using `parsoptp` (see Weesie 1997) and hence expressions may contain parentheses.) I would prefer to actually identify the two sets of points in one graph. This is accomplished with the `overlay` option:

```
.hilite2 price mpg, hilite(rep78==1 rep78==2 rep78==3 rep78>4) overlay border xlabel ylabel
```
Note that `hilite2` allows the specification of the normal options for `twoway` plots. Overlay plots are only possible with a restricted number of marker symbols; Stata currently allows up to six highlighted sets. Note that Stata will display identifying labels above the figure for at most 4 different sets.

Typing a sequence of explicit equations is tedious and often leads to errors. Often, as in the case above, the expressions are very similar. Thus, I added an option `hivar` that accepts a `varlist` and makes an overlay or matrix plot in which groups are highlighted that have the same values for the variables in `varlist`. Two examples

```
. hilite2 price mpg, hivar(rep78)
. hilite2 price mpg, hivar(rep78 foreign) border bsize(300) title(A big plot)
```

are shown in Figures 3 and 4.
Note that value labels are used whenever defined to enhance the readability of the plots.

References

I used a version of *by* since I started to use Stata, because I never liked to “manually” sort variables. In addition, it was hard to explain to my colleagues and to students of my applied statistics classes why a modern program such as Stata would not sort itself. Recently on the discussion list, Bill Gould described his reason why Stata does not sort automatically: one requires the data to be sorted within the groups formed by the values of a *varlist*. This point is well taken. In an earlier version of *by*, I dealt with this problem by *only* sorting if the data were not already “sufficiently” sorted, that is, sorted on the specified variables, and, possibly, on additional variables.

Bill suggested the syntax used above in order to specify sorting on *varlist2* within groups formed by *varlist1*. I like this new syntax as it expresses semantics much better than the current implicit approach with *by* and my original *by*. I therefore modified my code of *by*.

References

**sbe29 Generalized linear models: extensions to the binomial family**

James Hardin, Texas A&M University, jhardin@stat.tamu.edu
Mario Cleves, Stata Corporation, mcleves@stata.com

In this article we present a new command for fitting generalized linear models with binary outcomes. The *binreg* command is an extension to the *glm* command that provides some features not available with *glm*. The *binreg* command offers one new link function (log complement) and provides proper range checking for other link functions (identity and log) so that one may obtain estimates with those links.

Using this new command, one may fit a generalized linear model with the binomial family and one of the four links identity, logistic, log, and log complement. Exponentiated coefficients are presented by default for the logistic, log, and log complement links, but the raw coefficients may be obtained using the *coeff* option. The identity link provides only the raw coefficients.

**Syntax**

```
binreg depvar [varlist] [weight] [if exp] [in range] [, noconstant scale(x2|dev|#) lnoffset(varname)
          disp(#) coeff [or|rr|hr|rd] level(#) iterate(#) lltol(#) in(varname|#)
          nolog(#) ]
```

aweights, fweights, and iweights are allowed, see [U] 18.1.6 weight.
*binreg* shares the features of all estimation commands.

**Syntax for predict**

```
predict [type] newvarname [if exp] [in range] [, statistic ] [noffset]
```

where *statistic* is

- **mu** predicted mean of $\hat{y} = g^{-1}(x\hat{b})$ (the default)
- **xb** linear prediction
- **stdp** standard error of the linear prediction; SE($x_jb$)
- **deviance** deviance residual
- **pearson** Pearson residual

These statistics are available both in and out of sample; type *predict* ... *if e(sample)* ... if wanted only for the estimation sample.

**Description**

*binreg* fits generalized linear models for the binomial family. It estimates odds ratios, risk ratios, health ratios and risk differences. The available links are
<table>
<thead>
<tr>
<th>Option</th>
<th>Implied link</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>logit</td>
<td>Odds ratios = ( \exp(\beta) )</td>
</tr>
<tr>
<td>rr</td>
<td>log</td>
<td>Risk ratios = ( \exp(\beta) )</td>
</tr>
<tr>
<td>hr</td>
<td>log complement</td>
<td>Health ratios = ( \exp(\beta) )</td>
</tr>
<tr>
<td>rd</td>
<td>identity</td>
<td>Risk differences = ( \beta )</td>
</tr>
</tbody>
</table>

Note that estimates of odds, risk and health ratios are obtained by exponentiating the appropriate coefficients. The option or produces the same results as Stata’s logistic command and or coeff yields the same results as the logit command. When no link is specified/implied, or is assumed (logistic link is implied).

**Options**

noconstant specifies that the linear predictor has no intercept term, thus forcing it through the origin on the scale defined by the link function.

scale(x2)dev(#) overrides the default scale parameter. By default, scale(1) is assumed for discrete distributions (binomial, Poisson, negative binomial) and scale(x2) for continuous distributions (Gaussian, gamma, inverse Gaussian).

scale(x2) specifies the scale parameter be set to the Pearson chi-squared (or generalized chi-squared) statistic divided by the residual degrees of freedom.

scale(dev) sets the scale parameter to the deviance divided by the residual degrees of freedom. This provides an alternative to scale(x2) for continuous distributions and over- or under-dispersed discrete distributions.

scale(#) sets the scale parameter to #.

\[ \ln|\text{offset}(\text{varname})| \] specifies an offset to be added to the linear predictor. offset() specifies the values directly: \( g(E(y)) = xB + \text{varname} \). lnoffset() specifies exponentiated values: \( g(E(y)) = xB + \ln(\text{varname}) \).

disp(#) multiplies the variance of \( y \) by # and divides the deviance by #. The resulting distributions are members of the quasi-likelihood family.

coeff displays the nonexponentiated coefficients and corresponding standard errors and confidence intervals. This has no effect when the rd option is specified as it is always presenting the nonexponentiated coefficients.

or requests the logit link and results in odds ratios if coeff is not specified.

rr requests the log link and results in risk ratios if coeff is not specified.

hr requests the log complement link and results in health ratios if coeff is not specified.

rd requests the identity link and results in risk differences if coeff is not specified.

level(#) specifies the confidence level, in percent, for confidence intervals of the coefficients.

iterate(#) specifies the maximum number of iterations allowed in estimating the model; iterate(50) is the default.

ltol(#) specifies the convergence criterion for the change in deviance between iterations; ltol(1e-6) is the default.

init(varname) specifies varname containing an initial estimate for the mean of depvar. This can be useful if you encounter convergence difficulties.

n(varname|#) specifies either a constant integer to use as the denominator for the binomial family, or a variable which holds the denominator for each observation. This is useful for grouped data; see [R] glm for a complete description.

nolog(#) suppresses the iteration log.

**Options for predict**

mu the default, requests the predicted value of \( y \); \( \hat{y} = g^{-1}(xB) \).

xb calculates the linear prediction.

stdp requests the standard error of the linear predictor.

deviance requests the deviance residuals.

pearson requests Pearson residuals.

nooffset is relevant only if you specified offset() or lnoffset() for glm. It modifies the calculations made by predict so that they ignore the offset variable; the linear prediction is treated as \( x_b \) rather than \( x_b + \text{offset}_j \).
Remarks

Wacholder (1986) suggests methods for estimating risks ratios and risk differences from prospective binomial data. These estimates are obtained by selecting the proper link functions in the generalized linear model framework.

Let \( \pi_i \) be the probability of success for the \( i \) observation, \( i = 1, \ldots, N \) and \( X \beta \) the linear predictor. Then the link function relates the covariates of each observation to its respective probability through the linear predictor.

In logistic regression the logit link is used

\[
\ln \left( \frac{\pi}{1 - \pi} \right) = X \beta
\]

The regression coefficient \( \beta_k \) represents the change in the logarithm of the odds associated with a one unit change in the value of \( X_k \) covariate, thus, \( \exp(\beta_k) \) is the ratio of the odds associated with a change of one unit in \( X_k \).

For risk differences, the identity link \( \pi = X \beta \) is used. The regression coefficient \( \beta_k \) represents the risk difference associated with a change of one unit in \( X_k \). When using the identity link it is possible to obtain fitted probabilities outside of the interval \((0, 1)\). As suggested by Wacholder, at each iteration fitted probabilities are checked for range conditions (and put back in range if necessary). For example, if the identity link results in a fitted probability that is smaller than \( 1e^-4 \), the probability is replaced with \( 1e^-4 \) before the link function is calculated.

A similar adjustment is made for the logarithmic link which is used for estimating the risk ratio, \( \ln(\pi) = X \beta \) where \( \exp(\beta_k) \) is the risk ratio associated with a change of one unit in \( X_k \), and for the log complement link used to estimate the probability of no disease or health, where \( \exp(\beta_k) \) represents the “health ratio” associated with a change of one unit in \( X_k \).

Example

Wacholder (1986) presents an example utilizing data from Wright et al. (1983) of an investigation of the relationship between alcohol consumption and the risk of a low birth weight baby. Covariates examined included whether the mother smoked (yes or no), mother’s social class (three levels) and drinking frequency (light, moderate or heavy). The data for the 18 possible categories determined by the covariates is illustrated below.

Let’s first describe the data and list a few observations.

```
   . list, noobs

     cat  d   n  alc   smo  soc
    1    1   84   3    1    1
    2    5   79   2    1    1
    3   11  169   1    1    1
    4    6   26   3    2    1
    5    3   13   2    2    1
    6    1   26   1    2    1
    7    4   22   3    1    2
    8    3   25   2    1    2
   10    4   17   3    2    2
   11    2   17   2    2    2
   12    6   36   1    2    2
   13    0   14   3    1    3
   14    1   18   2    1    3
   15   12   91   1    1    3
   16    7   19   3    2    3
   17    2   18   2    2    3
   18    8   70   1    2    3
```

Each observation corresponds to one of the 18 covariate structures. The number of low birth babies out of \( n \) in each category is given by the variable \( d \).

We will begin by estimating risk ratios:

(Example continued on next page)
By default, the program outputs the risk ratios (the exponentiated regression coefficients) estimated by the model. We can see that the risk ratio comparing heavy drinkers with light drinkers after adjusting for smoking and social class is \( \exp(0.6801017) = 1.9740785 \). That is, mothers who drink heavily during their pregnancy have approximately twice the risk of delivering low weight babies than mothers who are light drinkers.

The nonexponentiated coefficients can be obtained via the `coeff` option.

Risk differences are obtained using the `rd` option:
The risk difference between the heavy drinkers and the light drinkers is simply the value of the coefficient for $\text{Ialc}_3 = 0.0801291$. Note that risk differences are obtained directly from the coefficients estimated using the identity link, thus the `coeff` option has no effect in this case.

Health ratios are obtained using the `hr` option. The health ratios (exponentiated coefficients for the log-complement link), are reported directly.

To see the nonexponentiated coefficients we can specify the `coeff` option.

Saved Results

`binreg` saves the same results in `e()` as `glm`. See `[R] glm` for a listing.

References


A small bug in one of the support routines to `mfracpol` (see Royston and Ambler, 1999) has been found and corrected.

Reference

sg112.1 Nonlinear regression models involving power or exponential functions of covariates: update

Patrick Royston, Imperial College School of Medicine, UK, proyston@rpms.ac.uk
Gareth Ambler, Imperial College School of Medicine, UK, gambler@rpms.ac.uk

The support routine frac_ext.ado was inadvertently left out of the distribution for boxtid (see Royston and Ambler, 1999). This caused an error in the init option. The diskette accompanying this issue of the STB has the complete distribution for boxtid.

Reference

sg113 Tabulation of modes

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

Syntax

\[ \text{modes} \ varname \ [\text{weight}] \ [\text{if}] \ [\text{in}] \ [\text{range}] \ , \ \text{min}(#) \ ] \]

Description

\text{modes} tabulates the mode(s) of \text{varname}, that is, the value(s) of \text{varname} that occur most frequently. \text{varname} may be numeric or string.

\text{fweights} and \text{aweights} are allowed.

Options

\text{min}(#) \ specifies that all values with a frequency of \# or more should be shown.

Remarks

The mode may be defined strictly as the most common value of a variable, meaning the value with the highest frequency. Modes may be defined more broadly as common values, as is implied by terms such as bimodal or multimodal. In practice precise work with any broad definition requires a specification, possibly arbitrary, of the minimum acceptable frequency. Even with the strict definition several modes may be identified whenever two or more values occur with the highest observed frequency. Hence the problem of reporting modes is often a problem of tabulating several commonly observed values.

Information on modes is supplied as part of the output of \text{tabulate} (see [R] \text{tabulate}). However, with many data sets such output may be so long that a more specialized tool aimed at tabulating only the most common values may be useful.

\text{modes} by itself tabulates the value or values with the highest frequency. \text{modes} with the \text{min()} option tabulates that value or values with at least the specified minimum frequency.

\text{modes} is most obviously useful with a discrete or categorical variable. Indeed, it may be applied to both numeric and string variables. Continuous variables may need to be placed in bins or classes first; otherwise with arbitrarily precise measurement each value approaches uniqueness. Alternatively, with a continuous (or nearly continuous) variable it may be much more helpful to inspect a graph of the frequency distribution when looking for modes, such as a histogram or a dotplot or a spike plot (see [G] \text{graph}). A graph is also likely to be much better for showing local modes; which may be defined as values more common than values just higher or lower, even if they are not especially common, and for showing other fine structure in the frequency distribution. Some form of density estimation (see [R] \text{kdensity}) may also be useful.

Example

```
. use auto
   (1978 Automobile Data)
. modes rep/76
Mode of rep/76

                   | Frequency
-------------|---------
Repair           |         
Record           |         
1976             | 3        
1978             | 30       
```

. modes rep78, min(10)  
Modes of rep78  
-------------------  
Repair |     
Record |     
1978   | Frequency  
-------------------  
3 | 30  
4 | 18  
5 | 11  
-------------------  

RGLM - Robust variance estimates for generalized linear models  
Roger Newson, Imperial College School of Medicine, London, UK, r.newson@ic.ac.uk  

Syntax  
rglm [varlist] [weight] [if exp] [in range], cluster(varname) mspec tdist minus(#) glm-options  

fweights, iweights and aweights are allowed; see [U] 18.1.6 weight.  

Description  
rglm fits generalized linear models and calculates a Huber (sandwich) estimate of the variance–covariance matrix of estimates. It can be used alone or called without arguments after a previous call to glm. As with other “robust” commands, the units may be considered to fall into clusters.  

Options  
cluster(varname) specifies the variable which defines sampling clusters.  
mspec specifies that full Huber variances be used. These are robust to misspecification of conditional means. If mspec is absent, semi-Huber variances are calculated, robust to variance misspecification caused by overdispersion, underdispersion, heteroscedasticity and clustering, but assuming that conditional means are specified correctly by the model. (Except in the case of canonical link functions, where the semi-Huber variance is the full Huber variance. See Section 2.5 of McCullagh and Nelder (1989).)  
tdist specifies that p values and confidence intervals are to be calculated assuming estimates to have a t distribution with M − p degrees of freedom, where p is the number of model parameters, and M is the number of clusters if cluster is specified, or the number of observations (or sum of frequency weights) if cluster is not specified.  
minus(#) specifies the minus parameter to pass to rrobust, to apply a finite-sample adjustment to the Huber covariance matrix.  
If absent (or negative), it is reset to p (the number of model parameters).  

glm-options are any of the options available for glm; see [R] glm.  
If a varlist is supplied, then all glm options are allowed. If not, then the only glm options allowed are level and eform, and cluster, mspec, tdist and minus are ignored.  

Methods and Formulas  
In a generalized linear model (GLM), we attempt to predict an (n × 1) outcome variate Y using an (n × p) matrix X of predictor variates, where n is the number of observations and p is the number of parameters. These parameters form a (p × 1) vector β. The (n × 1) vector η = Xβ, known as the linear predictor, is used to predict Y, which is assumed in the model to have a conditional expectation equal to the (n × 1) vector μ. The vectors η and μ are assumed in the model to have a relation of the form ηi = g(μi), where g(·) is a monotonically increasing function, referred to as the link function. The conditional variance of yi, given X, is assumed in the model to be proportional to a variance function V(μi). The choice of link function g(·) and variance function V(·) distinguishes one GLM from another. We will assume frequency weights (fweights) wi and non-frequency weights (iweights) wi, both defaulting to ones if not specified. (In fact, only one kind of weight may be specified, but that is a very minor defect of Stata, not a mathematical requirement.)  
The fitting of a GLM involves finding values of β which give a zero value simultaneously for the p sums of scores \( \sum_{i=1}^{n} f_i \psi_{ij} \), for j from 1 to p. The (n × p) matrix Ψ is defined such that \( \psi_{ij} = w_i x_{ij} S_i \), where the S_i are in turn defined by
\[ S_i = S(y_i, \eta_i) = \frac{d\mu_i}{d\eta_i} |V(\mu_i)|^{-1} (\mu_i - y_i) \]

In the model, \( S_i \) is interpreted as the derivative, with respect to \( \eta_i \), of the \( i \)th squared deviance residual, which is proportional to the conditional log likelihood of \( y_i \) given the row matrix \( X_i \). \( \psi_{ij} \) is the corresponding derivative with respect to \( \beta_j \). (See [R] \texttt{glm} or McCullagh and Nelder 1989.)

The derivative of the \( j \)th sum of scores with respect to the \( k \)th parameter \( \beta_k \) is equal to \( \sum_{i=1}^{n} f_i w_i x_i j x_i k H_i \), where \( H_i \) is the \( i \)th Hessian function

\[
H_i = \frac{dS_i}{d\eta_i} = [V(\mu_i)]^{-1} \left( \frac{d\mu_i}{d\eta_i} \right)^2 + (\mu_i - y_i) \frac{d}{d\eta_i} \left\{ [V(\mu_i)]^{-1} \frac{d\mu_i}{d\eta_i} \right\}
\]

To estimate the dispersion matrix of the parameters \( \beta_j \), we proceed as follows, using the principles of Huber (1967). Define the \((p \times p)\) matrix \( D = \sum_{i=1}^{n} f_i w_i H_i X_i^T X_i \). The variance expression depends on whether or not clusters are specified. If there are no clusters, then the estimated dispersion matrix is

\[
\frac{\sum_{i=1}^{n} f_i}{\sum_{i=1}^{n} f_i - k_{\text{min}}} (\Psi D^{-1})^T F (\Psi D^{-1})
\]

where \( k_{\text{min}} \) is the value given by the \texttt{min} option, and \( F \) is the \((n \times n)\) diagonal matrix of the frequency weights, \( f_i \). If there are clusters, we denote by \( M \) the number of these clusters and define \( \Psi^* \) as the \((M \times p)\) matrix with one row per cluster, equal to the sum of the rows of \( \Psi \) corresponding to observations in that cluster, and estimate the dispersion matrix as

\[
\frac{n}{n - k_{\text{min}}} (\Psi^* D^{-1})^T (\Psi^* D^{-1})
\]

(It does not make sense to have both clusters and \texttt{fweights}, because \( f_i > 1 \) implies that the \( i \)th observation represents multiple clusters.)

To calculate the Hessian in the general case by (2), we must know the variance function with its first derivative, and the inverse link function with its first two derivatives. The available variance functions have names corresponding to distributional families, whose variances are proportional to the respective functions, and their formula and derivatives are as follows:

<table>
<thead>
<tr>
<th>Family name</th>
<th>( V(\mu) )</th>
<th>( dV(\mu)/d\mu )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian (normal)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Gamma</td>
<td>( \mu^2 )</td>
<td>2\mu</td>
</tr>
<tr>
<td>Inverse Gaussian</td>
<td>( \mu^3 )</td>
<td>3\mu^2</td>
</tr>
<tr>
<td>Bernoulli</td>
<td>( \mu (1 - \mu) )</td>
<td>1 - 2\mu</td>
</tr>
<tr>
<td>Poisson</td>
<td>( \mu )</td>
<td>( k \mu )</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>( \mu + k\mu^2 )</td>
<td>1 + 2k\mu</td>
</tr>
</tbody>
</table>

The case of fitting a binomial model with totals \( m_i \) to the \( y_i \) is handled by \texttt{rglm} as equivalent to fitting a Bernoulli model to the proportions \( y_i/m_i \) and multiplying the \texttt{fweights} by the \( m_i \). (That is to say, we substitute \( y_i/m_i \) for \( y_i \), and \( w_i m_i \) for \( w_i \), in the formula above.) In the case of the negative binomial distribution, the shape parameter \( k \) is defined according to the conventions of the Stata manuals and the innards of \texttt{glm ado}, in which \( k \) is the reciprocal of the parameter of the same name defined in McCullagh and Nelder (1989). I do not know how this confusing state of affairs came about.
The available forms for a link function $\eta = g(\mu)$ also have names. The following table gives their formula and inverses, with their first and second derivatives. The derivatives are expressed in a computationally convenient form. In the case of the probit link, $\Phi(\cdot)$ is the standard Gaussian cumulative distribution function, and $\phi(\cdot)$ is its derivative, the standard Gaussian probability density function.

<table>
<thead>
<tr>
<th>Link function</th>
<th>$g(\mu)$</th>
<th>$g^{-1}(\eta)$</th>
<th>$d\mu/d\eta$</th>
<th>$d^2\mu/d\eta^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>$\mu$</td>
<td>$\eta$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Log</td>
<td>$\ln \mu$</td>
<td>$e^\eta$</td>
<td>$\mu$</td>
<td>$\mu$</td>
</tr>
<tr>
<td>Logit</td>
<td>$\ln[\mu/(1-\mu)]$</td>
<td>$e^{\eta}/(1+e^\eta)$</td>
<td>$(\mu - 1)\mu (1 - 2\mu)$</td>
<td>$(\mu - 1)log(1 - \mu)$ + $\mu(1 - \mu)(1 - 2\mu)$</td>
</tr>
<tr>
<td>Probit</td>
<td>$\Phi^{-1}(\mu)$</td>
<td>$\Phi(\eta)$</td>
<td>$\mu$</td>
<td>$-\eta\phi(\eta)$</td>
</tr>
<tr>
<td>Complementary log-log</td>
<td>$\ln[- \ln(1 - \mu)]$</td>
<td>$1 - e^{-e^\eta}$</td>
<td>$(\mu - 1)\log(1 - \mu)$</td>
<td>$[1 + \log(1 - \mu)]d\mu/d\eta$</td>
</tr>
<tr>
<td>Odds power $q$</td>
<td>$\mu/(1 - \mu)^q$</td>
<td>$\eta^{1/q}/(1 + \eta^{1/q})$</td>
<td>$1/q\mu^{1-q}(1 - \mu)^{1+q}$</td>
<td>$q(1 - q)\mu^{1-q}(1 - 2\mu - q)d\mu/d\eta$</td>
</tr>
<tr>
<td>Power $q$</td>
<td>$\mu^q$</td>
<td>$\eta^{1/q}$</td>
<td>$q^{-1}\mu^{1-q}$</td>
<td>$(1 - q)q^{-1}\mu^{1-q}d\mu/d\eta$</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>$\ln[k\mu/(k\mu + 1)]$</td>
<td>$k^{-1}e^{\eta}/(1 - e^\eta)$</td>
<td>$\mu + k\mu^2$</td>
<td>$(1 + 2k\mu)d\mu/d\eta$</td>
</tr>
</tbody>
</table>

The negative binomial link function defined here is the correct version, consistent with the notation of the Stata manuals and with glm.ado. (The definition in [R] glm is a misprint.)

The calculation of the Hessian is greatly simplified if we can ignore the second term of the second line of (2), in which case we have

$$H_i = [V(\mu_i)]^{-1} \left( \frac{d\mu_i}{d\eta_i} \right)^2$$

(5)

and we need only know the variance function and the first derivative of the inverse link. This equality holds, in the expectation, if the model is indeed a correct specification of the conditional mean of $Y$ given $X$, so that $E(\mu_i, y_i) = 0$ for each individual $i$. It also holds if the link function is the canonical link for the variance function. In this case, the variance function is proportional to the first derivative of the inverse link, and their ratio is a constant function of $\eta$, so the second term of the second line in (2) is zero. (See Section 2.5 of McCullagh and Nelder 1989.) The variances calculated using the formula (5) are known as semi-Huber variances, whereas the variances calculated using formula (2) are known as full Huber variances. The semi-Huber variances (given by default) are robust to heteroscedasticity, overdispersion, underdispersion and clustering. The full Huber variances (obtained by the mspec option) are robust to all of these, and also to mis-specification of the conditional expectation. So, for instance, if we are fitting the parameters of a straight line, using a link function non-canonical for the chosen variance function, and the true relationship is slightly curved, then the parameters are estimates of the straight line giving the best fit to that curve, and the full Huber variances are consistent estimators of the true variance, in the population from which the rows of $X$ and $Y$ are jointly sampled. (I have not had time to do much research on how important the difference between semi-Huber and full Huber variances is in practice.)

**Example 1**

I often use rglm for carrying out unequal-variance $t$ tests on logs, using the eform option to get confidence intervals (CIs) for the two group geometric means and their ratio. For instance, in the case of the auto data, we might decide (after looking at stem-and-leaf plots) that mpg (miles per gallon) was distributed lognormally rather than normally. The calculation of the geometric means and their ratio is carried out by Stata as follows:

. * Geometric averages and their ratio *
. gen logmpg=log(mpg)
. gen byte us/=foreign
. * Stem and leaf plots *
. stem mpg
We find that foreign cars traveled at a geometric average of 23.96 mpg, whereas US cars traveled at a geometric average of 19.30 mpg. The foreign/US ratio was 1.24 (95% CI, 1.09 to 1.41), so foreign cars, on average, were 9% to 41% more efficient than US cars.

Example 2

We might also do a probit analysis to find a way of guessing whether a car is foreign, based on knowledge of its fuel efficiency and weight. The probit link is non-canonical for the Bernoulli variance function, so the full Huber variance will in general be different from the semi-Huber variance. Here, the analysis is carried out in three ways: using glm, using rglm with semi-Huber variances, and using rglm with full Huber variances. The results are as follows:
Example 3

This example is based on Stata’s housing data. Here, the data points are states of the USA, and we want to predict median rent from pcturban (percent urban) and hsngval (median housing value). This example compares the output from rglm, tdist with those from regress and regress, robust. Note that the two robust methods produce the same result (as they should), but the nonrobust method gives the same estimates and very different CIs.
Validation

A program as comprehensive as rglm requires more validation than three examples. Accordingly, an intensive validation was carried out, using the auto data. Rglm was tested using all six available variance functions, with one y variate for each (rep78 for the three discrete families, mpg for the three continuous families). Each family was tested with one canonical and one non-canonical link, except the binomial family, which was tested with its canonical logit link and all the non-canonical links for which the Binomial family is obligatory. (So every family and link was tested, and every family was tested with a canonical and a non-canonical link.) For each combination of family and link, three models were fitted. These had parameters as follows:

**Model 1.** One parameter, corresponding to the grand mean.

**Model 2.** Two groups (US and foreign cars), with parameters corresponding to two group means.

**Model 3.** Two parameters (an intercept and the slope of a quantitative covariate).

The quantitative covariate in Model 3 was always gratio for identity links and weight for non-identity links. (This was done because when mpg and rep78 were plotted against weight and gratio, the relationships involving gratio looked more linear.) The models fitted for each distributional family are summarized below.

<table>
<thead>
<tr>
<th>Family</th>
<th>Y-variates</th>
<th>Canonical Link</th>
<th>Covariate for Canonical Link</th>
<th>Non-canonical Link(s)</th>
<th>Covariate for Non-canonical Link(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gaussian</td>
<td>mpg</td>
<td>identity</td>
<td>gratio</td>
<td>log</td>
<td>weight</td>
</tr>
<tr>
<td>gamma</td>
<td>mpg</td>
<td>power-1</td>
<td>weight</td>
<td>identity</td>
<td>gratio</td>
</tr>
<tr>
<td>igaussian</td>
<td>mpg</td>
<td>power-2</td>
<td>weight</td>
<td>identity</td>
<td>gratio</td>
</tr>
<tr>
<td>binomial</td>
<td>rep78</td>
<td>logit</td>
<td>weight</td>
<td>probit, cloglog, opower 2</td>
<td>weight</td>
</tr>
<tr>
<td>poisson</td>
<td>rep78</td>
<td>log</td>
<td>weight</td>
<td>identity</td>
<td>gratio</td>
</tr>
<tr>
<td>nbinomial</td>
<td>rep78</td>
<td>nbinomial</td>
<td>weight</td>
<td>identity</td>
<td>gratio</td>
</tr>
</tbody>
</table>

For each of the 42 models fitted, the dispersion was estimated in five different ways. These were the orthodox (Nelder) method given as default by glm, semi-Huber and full Huber variances without clustering, and semi-Huber and full Huber variances with clustering by manuf. Each parameter of each model therefore had five alternative standard errors (SEs).
In theory, some of these distinct SEs were expected to be equal. In the case of Model 1, there was no possibility for heteroscedasticity, overdispersion, underdispersion or misspecification (as there is a single constant X variate of ones), so all three unclustered SEs were expected to be equal, and both the clustered SEs were expected to be equal. In Model 2, there was a possibility of heteroscedasticity (because of unequal group variances), and sometimes overdispersion and underdispersion, but no possibility of misspecification (because the predicted value of each individual is its group mean). The semi-Huber SE was therefore expected to be equal to the corresponding full Huber SE in each clustering class, although the orthodox, unclustered Huber and clustered Huber SEs were expected to be different. In Model 3, there was a possibility of heteroscedasticity, overdispersion, underdispersion and misspecification, so all five SEs were expected to be different. Therefore, if rglm is working correctly, then we expect the SEs of Model 1 parameters to fall into two pre-defined equivalence groups (clustered and unclustered), the SEs of Model 2 parameters to fall into three pre-defined equivalence groups (orthodox, clustered Huber and unclustered Huber), and the SEs of Model 3 parameters to fall into five pre-defined equivalence groups of one each. SEs in the same equivalence group should be equal (or different only to the extent compatible with floating point calculation error), whereas SEs for the same model in different equivalence groups should be different.

As it happened, no two SEs in the same equivalence class were different by a ratio of more than 1.0001 (that is to say, the largest SE in an equivalence class was never more than 0.01% greater than the smallest SE in the same equivalence class). There was a lot more variation between equivalence classes for the same parameter of the same model. No two SEs in different equivalence classes for the same parameter of the same model differed by a ratio of less than 1.0020. That is to say, for any two SEs in different equivalence classes for the same parameter of the same model, the larger was always greater than the smaller by more than 0.2%, and usually the variation was much greater.

Figure 1 shows standard errors plotted on a binary log scale for all parameters of all models fitted. In the left-hand plot, the data points are SE equivalence classes (more than one for each parameter of each model), and the largest SE in the equivalence class is plotted against the smallest SE in the equivalence class. Note that all points are on the line of equality. In the right-hand plot, the data points are model parameters (one for each parameter of each model), and the largest SE for the parameter is plotted against the smallest SE calculated for that parameter. Note that the data points are visibly above the line of equality, although usually not so far above it as to indicate that the different SE calculation formulae give results in different binary orders of magnitude. So, unlike Example 3, these ad hoc examples do not truly demonstrate the advantages of Huber variances, although they do demonstrate that the SEs calculated by rglm using different methods are equal when they are supposed to be.

**Acknowledgment**

This program was based on a previous version called hglm, which calculated only semi-Huber variances, and was kindly supplied to the author by David Clayton of MRC in Cambridge, England. The present author cleaned out some bugs, and added the options mspec, tdist and minus.

**References**


**sts graph** has been modified so that tick marks indicating the number of censored observations may be placed on graphs of Kaplan–Meier survivor functions and Nelson–Aalen cumulative hazard functions.

The new option `censored()` does this; all other options remain unchanged.

**Syntax**

`sts graph` now has syntax

```
sts graph [if exp] [in range] [, by(varlist) strata(varlist) adjustfor(varlist) nolabel failure
    ggood na cna level(#) lost enter separate tmin(#) tmax(#) xasis yasis noborder noshow
    noorigin atrisk censored(single | number | multiple) graph_options ]
```

**Options**

The `censored()` option is new. See [R] `sts graph` for a description of the other options.

`censored(single|number|multiple)` specifies that tick marks be placed on the graph to indicate the censored observations.

- `censored(single)` places one tick at each censoring time regardless of the number of censorings at that time.
- `censored(number)` places one tick at each censoring time and displays the number of censorings above the tick.
- `censored(multiple)` places multiple ticks for multiple censorings at the same time. If three observations are censored at time 5, then three ticks are placed around time 5. `censored(multiple)` is intended for use when there are few censored observations; if there are too many, the graph can look bad and in such cases we recommend that `censored(number)` be used.

`censored()` is not allowed with options `lost`, `enter` or `atrisk`.

**Example**

Using the cancer data distributed with Stata, we will plot the Kaplan–Meier estimated survivor function using each of the three censoring options:

```
. use cancer.dta
(Patient Survival in Drug Trial)
. stset studytim, failure(died)
(output omitted)
. sts graph, censored(single) ti("censored(single)")
```

![Kaplan-Meier survival estimate](image-url)
Similarly, the new option can be used when plotting the estimated Nelson–Aalen cumulative hazard function.

```
. sts graph, na censored(single) ti("censored(single)"
```

```
. sts graph, na censored(single) ti("censored(single)"
```

```
. sts graph, na censored(single) ti("censored(single)"
```
Philip Ryan, University of Adelaide, Australia, pryan@medicine.adelaide.edu.au

In Ryan (1998) I published a program \texttt{ralloc} that randomly allocates treatments in blocks. I have updated the program in the following ways:

1. There is a new option \texttt{tr#lab(string)} (where \# can be 1, 2, 3, or 4) allowing user-specified treatment names. The defaults remain A, B, C, and D.

2. There is a new option \texttt{shape(long | wide)} allowing the randomization schedule to be saved in wide or long forms. In long form each observation is a treatment allocation, while in wide form each observation is a block (see the example below). Further, Stata’s \texttt{reshape} parameters are set so that one can easily switch between long and wide forms.

3. As a consequence of the above, the old variable called \texttt{Order} produced by \texttt{ralloc} is replaced by a new \texttt{SeqInBlk} (sequence within block) variable.

4. There are new choices of 1 or 2 in the \texttt{osize(#)} option. Previously, this option, which sets the number of different block sizes, took arguments 3, 4, 5, 6, or 7. This was really because my own work placed a premium on concealing the allocation sequence, rather than minimizing the chance of imbalance. For smaller studies, the latter may be more important, so a constant minimum block size (equal to the number of treatments) can now be specified by \texttt{osize(1)}. The value 2 was included for good measure.
5. Modification of, and addition to, the saved notes.

Note that `ralloc` is still written in Stata 5; the only change required to make it Stata 6 would be to modify the `syntax` statement, but I felt that some people may have used it to generate randomization schedules under Stata 5, still only have Stata 5, and would like the continuing facility to be able to reproduce their schedule, say for auditing purposes.

**Example**

To illustrate the new `shape` option, we have

```
. ralloc blknum blksiz Rx, ns(494) osiz(2) eq ntrat(2) sav(mywide) shape(wide)
>  trl1ab(Placebo) trl2lab(Active)
```

**Frequency of block sizes:**

<table>
<thead>
<tr>
<th>block size</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>81</td>
<td>49.39</td>
<td>49.39</td>
</tr>
<tr>
<td>4</td>
<td>83</td>
<td>50.61</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Total | 164   | 100.00|

Randomization data is saved to `mywide.dta` and is now in memory. Issue the `notes` command to review your protocol.

```
. list in 1/7
```

<table>
<thead>
<tr>
<th>blknum</th>
<th>blksiz</th>
<th>Rx1</th>
<th>Rx2</th>
<th>Rx3</th>
<th>Rx4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td>Placebo</td>
<td>Placebo</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>Active</td>
<td>Placebo</td>
<td>Active</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>Active</td>
<td>Active</td>
<td>Placebo</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>4</td>
<td>Placebo</td>
<td>Placebo</td>
<td>Active</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
<td>Placebo</td>
<td>Active</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>4</td>
<td>Placebo</td>
<td>Active</td>
<td>Placebo</td>
</tr>
</tbody>
</table>

If we now use `reshape` we have

```
. reshape long
```

<table>
<thead>
<tr>
<th>Data</th>
<th><code>j</code> variable (4 values)</th>
<th><code>xij</code> variables:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SeqInBlk</code></td>
<td><code>Rx1 Rx2 ... Rx4</code></td>
<td><code>Rx</code></td>
</tr>
</tbody>
</table>

To illustrate the new `SeqInBlk` variable, we have

```
. sort blknum SeqInBlk
. drop if Rx ==
(163 observations deleted)
. list in 1/10
```

<table>
<thead>
<tr>
<th>blknum</th>
<th>SeqInBlk</th>
<th>blksiz</th>
<th>Rx</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Reference**

STB categories and insert codes

Inserts in the STB are presently categorized as follows:

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

**Statistical Categories:**
- \textit{sbe} biostatistics & epidemiology
- \textit{sed} exploratory data analysis
- \textit{sg} general statistics
- \textit{snv} multivariate analysis
- \textit{snr} nonparametric methods
- \textit{sqc} quality control
- \textit{sqv} analysis of qualitative variables
- \textit{srd} robust methods & statistical diagnostics
- \textit{ssa} survival analysis
- \textit{ssi} simulation & random numbers
- \textit{ssw} social science & psychometrics
- \textit{sts} time-series, econometrics
- \textit{sud} survey sampling
- \textit{svd} experimental design
- \textit{szz} not elsewhere classified

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

**Guidelines for authors**

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

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

A submission to the STB consists of

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

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

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

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

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

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

```
tar -cf - | compress | uuencode xyz.tar.Z > whatever
mail stb@stata.com < whatever
```
International Stata Distributors

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: P.O. Box 1169 17100 NAZERATH-ELLIT</td>
<td>Address: P.O. Box 2222 PRIMROSE 1416</td>
<td>South Africa</td>
</tr>
<tr>
<td>Phone: +972 (0)6 6100101</td>
<td>Phone: +27-11-8286169</td>
<td></td>
</tr>
<tr>
<td>Fax: +972 (0)6 6554254</td>
<td>Fax: +27-11-8221377</td>
<td></td>
</tr>
<tr>
<td>Email: <a href="mailto:ascc@netvision.net.il">ascc@netvision.net.il</a></td>
<td>Email: <a href="mailto:iem@hot.co.za">iem@hot.co.za</a></td>
<td></td>
</tr>
<tr>
<td>Countries served: Israel</td>
<td>Countries served: South Africa, Botswana, Lesotho, Namibia, Mozambique, Swaziland, Zimbabwe</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company: Axon Technology Company Ltd</th>
<th>Company: MercoStat Consultores</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: 9F, No. 259, Sec. 2 Ho-Ping East Road TAIPEI 106</td>
<td>Address: 9 de junio 1389 CP 11400 MONTEVIDEO Uruguay</td>
<td></td>
</tr>
<tr>
<td>Phone: +886-(0)2-27045535</td>
<td>Phone: 598-2-613-7905</td>
<td></td>
</tr>
<tr>
<td>Fax: +886-(0)2-27541785</td>
<td>Fax: Same</td>
<td></td>
</tr>
<tr>
<td>Email: <a href="mailto:hank@axon.axon.com.tw">hank@axon.axon.com.tw</a></td>
<td>Email: <a href="mailto:mercost@adinet.com.uy">mercost@adinet.com.uy</a></td>
<td></td>
</tr>
<tr>
<td>Countries served: Taiwan</td>
<td>Countries served: Uruguay, Argentina, Brazil, Paraguay</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company: Chips Electronics</th>
<th>Company: Metrika Consulting</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: Lokasari Plaza 1st Floor Room 82 Jalan Mangga Besar Raya No. 82 JAKARTA Indonesia</td>
<td>Address: Mosstorpsvagen 48 183 30 Taby STOCKHOLM Sweden</td>
<td></td>
</tr>
<tr>
<td>Phone: 62 - 21 - 600 66 47</td>
<td>Phone: +46-708-163128</td>
<td></td>
</tr>
<tr>
<td>Fax: 62 - 21 - 600 66 47</td>
<td>Fax: +46-8-7924747</td>
<td></td>
</tr>
<tr>
<td>Email: <a href="mailto:puyuh23@indo.net.id">puyuh23@indo.net.id</a></td>
<td>Email: <a href="mailto:sales@metrika.se">sales@metrika.se</a></td>
<td></td>
</tr>
<tr>
<td>Countries served: Indonesia</td>
<td>Countries served: Sweden, Baltic States, Denmark, Finland, Iceland, Norway</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company: Dittrich &amp; Partner Consulting</th>
<th>Company: Ritme Informatique</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: Kieler Strasse 17 5. floor D-42697 Solingen Germany</td>
<td>Address: 34, boulevard Haussmann 75009 Paris France</td>
<td></td>
</tr>
<tr>
<td>Phone: +49 2 12 / 26 066 - 0</td>
<td>Phone: +33 (0)1 42 46 00 42</td>
<td></td>
</tr>
<tr>
<td>Fax: +49 2 12 / 26 066 - 66</td>
<td>Fax: +33 (0)1 42 46 00 33</td>
<td></td>
</tr>
<tr>
<td>Email: <a href="mailto:sales@dpc.de">sales@dpc.de</a></td>
<td>Email: <a href="mailto:info@ritme.com">info@ritme.com</a></td>
<td></td>
</tr>
<tr>
<td>URL: <a href="http://www.dpc.de">http://www.dpc.de</a></td>
<td>URL: <a href="http://www.ritme.com">http://www.ritme.com</a></td>
<td></td>
</tr>
<tr>
<td>Countries served: Germany, Austria, Italy</td>
<td>Countries served: France, Belgium, Luxembourg</td>
<td></td>
</tr>
</tbody>
</table>

(List continued on next page)
# International Stata Distributors

(Continued from previous page)

<table>
<thead>
<tr>
<th>Company: Scientific Solutions S.A.</th>
<th>Company: Timberlake Consulting S.L.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: Avenue du Général Guisan, 5 CH-1009 Pully/Lausanne Switzerland</td>
<td>Address: Calle Mendez Nunez, 1, 3 41011 Sevilla Spain</td>
</tr>
<tr>
<td>Phone: 41 (0)21 711 15 20</td>
<td>Phone: +34 (9) 5 422 0648</td>
</tr>
<tr>
<td>Fax: 41 (0)21 711 15 21</td>
<td>Fax: +34 (9) 5 422 0648</td>
</tr>
<tr>
<td>Email: <a href="mailto:info@scientific-solutions.ch">info@scientific-solutions.ch</a></td>
<td>Email: <a href="mailto:timberlake@zoom.es">timberlake@zoom.es</a></td>
</tr>
<tr>
<td>Countries served: Switzerland</td>
<td>Countries served: Spain</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: Doormanstraat 19 5151 GM Drunen Netherlands</td>
<td>Address: Praceta Raul Brandao, nº1, 1ºE 2720 ALFragide Portugal</td>
</tr>
<tr>
<td>Phone: +31 416-378 125</td>
<td>Phone: +351 (0)1 471 73 47</td>
</tr>
<tr>
<td>Fax: +31 416-378 385</td>
<td>Fax: +351 (0)1 471 73 47</td>
</tr>
<tr>
<td>Email: <a href="mailto:J.A.C.M.Smit@smitcon.nl">J.A.C.M.Smit@smitcon.nl</a></td>
<td>Email: <a href="mailto:timberlake.co@mail.telepac.pt">timberlake.co@mail.telepac.pt</a></td>
</tr>
<tr>
<td>URL: <a href="http://www.smitconsult.nl">http://www.smitconsult.nl</a></td>
<td></td>
</tr>
<tr>
<td>Countries served: Netherlands</td>
<td>Countries served: Portugal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company: Survey Design &amp; Analysis Services P/L</th>
<th>Company: Unidost A.S.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: 249 Eramosa Road West Moorooduc VIC 3933 Australia</td>
<td>Address: Rihtim Cad. Polat Han D:38 Kadikoy 81320 ISTANBUL Turkey</td>
</tr>
<tr>
<td>Phone: +61 (0)3 5978 8329</td>
<td>Phone: +90 (216) 414 19 58</td>
</tr>
<tr>
<td>Fax: +61 (0)3 5978 8623</td>
<td>Fax: +30 (216) 336 89 23</td>
</tr>
<tr>
<td>Email: <a href="mailto:sales@survey-design.com.au">sales@survey-design.com.au</a></td>
<td>Email: <a href="mailto:info@unidost.com">info@unidost.com</a></td>
</tr>
<tr>
<td>Countries served: Australia, New Zealand</td>
<td>Countries served: Turkey</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company: Timberlake Consultants</th>
<th>Company: Vishvas Marketing-Mix Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: Unit B3 Broomsleigh Business Park Worsley Bridge Road LONDON SE26 5BN United Kingdom</td>
<td>Address: C‘O S. D. Wamorkar “Prashant” Vishnu Nagar, Naupada THANE - 400602 India</td>
</tr>
<tr>
<td>Phone: +44 (0)208 697 3377</td>
<td>Phone: +91-251-440087</td>
</tr>
<tr>
<td>Fax: +44 (0)208 697 3388</td>
<td>Fax: +91-22-5378552</td>
</tr>
<tr>
<td>Email: <a href="mailto:info@timberlake.co.uk">info@timberlake.co.uk</a></td>
<td>Email: <a href="mailto:vishvas@vsnl.com">vishvas@vsnl.com</a></td>
</tr>
<tr>
<td>URL: <a href="http://www.timberlake.co.uk">http://www.timberlake.co.uk</a></td>
<td></td>
</tr>
<tr>
<td>Countries served: United Kingdom, Eire</td>
<td>Countries served: India</td>
</tr>
</tbody>
</table>