STATA
TECHNICAL
BULLETIN

March 1995
STB-24

A publication to promote communication among Stata users

Editor

Sean Becketti
Stata Technical Bulletin
8 Wakeman Road
South Salem, New York 10590
914-533-2278
914-533-2902 FAX
stb@stata.com EMAIL

Associate Editors

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

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

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

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

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

dm25	Recoding in steps

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

Recoding variables from one numbering scheme to another is a tedious but necessary part of data preparation. Stata offers several tools to help in this task. The `autocode()` and `recode()` functions ([2] function) and the `recode` command ([5d] recode) can handle very general types of recoding. This insert presents a new command, `srecode`, that makes it easy to do a specialized, but common, type of recoding: recoding in uniform intervals or "steps".

As an example of recoding in steps, imagine that you want to recode the year into a variable that marks decades. For this example, no special command is needed. Just type

```
. generate int decade = 10*int(year/10)
```

Integer division can be used to easily recode in steps that are powers of ten. But what if you want to recode the year into five-year steps, rather than into decades? Then, you can type

```
.  generate int quint = decade + 5*(~mod(year,5))
```

These types of tricks can be extended to handle many cases. However, it is tiresome to try to remember these tricks. Moreover, you may not remember exactly how these tricks work when you reread your data preparation program many months later. And what do you do if you need to recode a variable into steps that are each, say, $\pi/6$ long?

The srecode command

`srecode` simplifies the task of recoding a variable in steps.

`srecode` [*type*] *newvar* `=` *exp* [`if` *exp*] [`in` *range*] [`,` `max`(#) `midpoint` `min`(#) `step`(#)]

recodes the expression and stores the recoded values in *newvar*.

`max(#)` specifies the maximum value of the expression to include in the recoding. By default, all values are used.

`midpoint` specifies that *newvar* should be coded with the midpoint of each step. The default is to use the minimum value on each step.

`min(#)` specifies the minimum value of the first step. By default, the minimum value of the expression is used, but this choice may produce awkward step boundaries.

`step(#)` specifies the width of each step. By default the step size is set to '1'.

Example

We use the automobile data supplied with Stata to illustrate the use of the `srecode` command. Our investigation focuses on the price and fuel efficiency (`mpg`) of the automobiles in this data set.

```
. use auto
(1978 Automobile Data)
. summarize price mpg
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+--------------------------------------------------------
   price |      74    6165.257   2949.496       3291      15906
     mpg |      74     21.2973   5.785503         12         41
```

We want to investigate whether the price of an automobile varies with its fuel efficiency. We have no prior opinion about the form of this relationship. Price might be either increasing or decreasing in fuel efficiency. The relationship might not be monotonic. There might be no relationship. To avoid imposing a functional form at this early stage of the investigation, we recode the price into uniform price categories, each $4,000 wide, with the intention of analyzing the average automobile price in each price category.

```
. srecode int pcat = price, min(0) step(4000)
. tabulate pcat, sum(price)
           |          Summary of Price
      pcat |       Mean   Std. Dev.       Freq.
-----------+------------------------------------
         0 |   3750.9091   246.92973          11
      4000 |   5153.5714   909.28853          49
      8000 |   10220.889   1277.1902           9
     12000 |     14091.2   1152.3173           5
-----------+------------------------------------
     Total |   6165.2568   2949.4959          74
```

By default, `srecode` assigns the minimum value on each step as the value of the recoded variable. Thus, in this example, cars with prices greater than or equal to $0 and less than $4,000 are assigned the code '`0`', cars with prices greater than or equal to $4,000 and less than $8,000 are assigned the code '`4000`', and so on. We used the option '`min(0)`' to force the steps to break on round numbers. Otherwise, `srecode` would have started the first step at $3,291, the minimum price in the data.

We note that the $4,000–$8,000 step contains the majority of the observations. To investigate the potential relationship between price and fuel efficiency more closely, we use `srecode` to break price into smaller groupings in this range.

```
. srecode int pcat2 = price if pcat>=4000 & pcat<8000, min(4000) step(1000)
(25 missing values generated)
. generate int newpcat = cond(pcat2!=.,pcat2,pcat)
```

We confirm that `newpcat` has been formed correctly.

```
. list price pcat pcat2 newpcat in f/15
       price      pcat     pcat2   newpcat
  1.    4099      4000      4000      4000
  2.    4749      4000      4000      4000
  3.    3799         0         .         0
  4.    4816      4000      4000      4000
  5.    7827      4000      7000      7000
  6.    5788      4000      5000      5000
  7.    4453      4000      4000      4000
  8.    5189      4000      5000      5000
  9.   10372      8000         .      8000
 10.    4082      4000      4000      4000
 11.   11385      8000         .      8000
 12.   14500     12000         .     12000
 13.   15906     12000         .     12000
 14.    3299         0         .         0
 15.    5705      4000      5000      5000
```

Finally, we examine the mean fuel efficiency in each price category.

```
. oneway mpg newpcat, tabulate
           |       Summary of Mileage (mpg)
   newpcat |       Mean   Std. Dev.       Freq.
-----------+------------------------------------
         0 |  26.181818   5.1733583          11
      4000 |       22.5   4.9497475          26
      5000 |  20.571429   6.6066899          14
      6000 |  20.428571   4.9617585           7
      7000 |         19   5.6568542           2
      8000 |  17.666667           4           9
     12000 |         15   3.4641016           5
-----------+------------------------------------
     Total |  21.297297   5.7855032          74

                       Analysis of Variance
    Source              SS         df      MS            F     Prob > F
------------------------------------------------------------------------
Between groups      640.180239      6   106.696706      3.96     0.0019
 Within groups      1803.27922     67   26.9146152
------------------------------------------------------------------------
    Total           2443.45946     73   33.4720474

Bartlett's test for equal variances:  chi2(6) =   3.5321  Prob>chi2 = 0.740
```

dm26	Labeling graphs with date formats

Alan Riley, Stata Corporation, FAX 409-696-4601, EMAIL stata@stata.com

This insert describes `datelab`, an ado-file that makes it easy to construct axis labels for date variables.

Stata's new date formats ([4] dates) are a big step forward in Stata's handling of date information. Stata has supported an elapsed date encoding of dates for several years. Natural date values that would normally be stored in two or three separate variables (e.g., `year month`) can be encoded as an elapsed date—defined as the number of days since January 1, 1960—and stored in a single variable. This approach makes it easy to perform many commonly needed date calculations.

Until the release of Stata 4.0, the elapsed date encoding suffered from an important drawback: the encoded values were difficult to read. The introduction of date formats has solved this problem. Elapsed dates now can be displayed in a variety of convenient and readable forms.

Dates are frequently used to label the x-axis of graphs, and Stata's new date formats improve the display of dates on graphs. However, Stata's axis labeling options are not "date-aware". By default, these options choose a handful of "nice" numbers to label. (Nice numbers typically end in a zero or a five.) These options choose nice elapsed date values, which typically do not correspond to nice dates.

This problem can be illustrated with a simple example. We have a data set that contains monthly observations on two variables: an elapsed date and the average daily yield on the 1-year constant maturity Treasury (CMT). The sample begins in January 1986 and ends in December 1995. If we let Stata automatically label the date axis, we get the following result:

```
. use cmt, clear
(Created 08:06:07, 11 Jan 1995)
. describe
Contains data from cmt.dta
  Obs:   108 (max= 30485)                    Created 08:06:07, 11 Jan 1995
 Vars:     2 (max=    99)                    22 Feb 1995 05:10
Width:     6 (max=   200)
  1. date         int    %dm_y               Date
  2. cmt1         float  %9.3f               1-year CMT yield
Sorted by:  date
. list in f
        date       cmt1
  1.   Jan 86      7.729
. list in l
        date       cmt1
108.   Dec 94      7.135
. graph cmt1 date, c(l) s(.) ylabel(3,4,5,6,7,8,9) rlabel(3,4,5,6,7,8,9) xlabel
(graph appears, see Figure 1)
```

While this graph provides a clear record of the Federal Reserve's actions to tighten monetary policy in 1994, the date values labeled by Stata are not the ones we normally would have chosen. For example, Stata labeled May 1987. It turns out that the elapsed date value for May 19, 1987 is a nice number, even though May 1987 is not a nice date:

```
. display = mdy(5,19,1987)
10000
```

To label this graph more sensibly, we could `display` the elapsed dates for the dates we wish to label, jot down the values, then type them explicitly in the `xlabel()` option. This approach is inconvenient. `datelab` provides a convenient alternative to this approach.

`datelab` is an interactive ado-file that calculates and stores elapsed dates. You specify dates (in m/d/y format), and Stata stores the elapsed date values corresponding to these dates in global macros called `date1`, `date2`, and so on. When you are finished storing elapsed date values, type "`end`" and `datelab` returns control of the session to you. The listing below shows how `datelab` can be used to improve the appearance of our graph.

```
. datelab
date 1 ? . 1/1/1986
date 2 ? . 1/1/1988
date 3 ? . 1/1/1990
date 4 ? . 1/1/1992
date 5 ? . 1/1/1994
date 6 ? . end
. graph cmt1 date, c(l) s(.) ylabel(3,4,5,6,7,8,9) rlabel(3,4,5,6,7,8,9)
                   xlabel($date1,$date2,$date3,$date4,$date5)
(graph appears, see Figure 1)
```

`datelab` is a relatively simple program. As a consequence, it is easy to modify `datelab` to accept dates in a new format if the default month/day/year format is inconvenient for you. The entire program is listed below:

```
*! datelab -- calculate date values for axis labels
*! version 1.0     Alan Riley, StataCorp     STB24: dm26
program define datelab
        version 4.0
        local i 0
        global date0
        while "${date`i´}"!="end" {
                local i = `i´ + 1
                di "date `i´ ? " _request(date`i´)
                if "${date`i´}"!="end" {
                        global date`i´ = date("${date`i´}","mdy")
                }
        }
end
```

To change the input format for dates, change the "`mdy`" string in the `date()` function to the format you prefer ([2] functions).

Figures

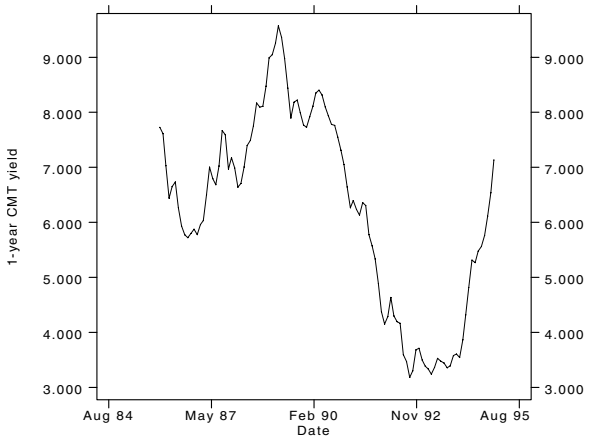

Figure 1

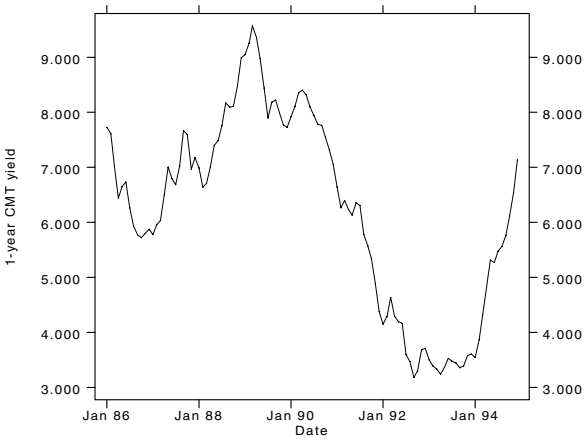

Figure 2

dm27	An improved collapse, with weights

William Gould, Stata Corporation, FAX 409-696-4601, EMAIL stata@stata.com

The syntax of `coll2` is

`coll2` *clist* [*weight*] [`if` *exp*] [`in` *range*] [`, by(`*varlist*`) cw`]

where *clist* is

[(*tspec*)] *varlist* [[(*tspec*)] ...]
[(*tspec*)] *target_var*=*varname* [*target_var*=*varname* ...] [[(*tspec*)] ...]

or any combination of the *varlist* and *target_var* forms, and *tspec* is

`mean`	means	`median`	medians
`sd`	standard deviations	`p1`	1st percentile
`sum`	sums	`p2`	2nd percentile
`rawsum`	sums ignoring optionally specified weight	...	3rd–49th percentiles
`count`	no. of nonmissing obs.	`p50`	50th percentile (same as `median`)
`max`	maximums	...	51st–97th percentiles
`min`	minimums	`p98`	98th percentile
		`p99`	99th percentile

`aweights` and `fweights` are allowed.

Description

`coll2` is an alternative to `collapse`; see [5d] collapse. It converts the data in memory into a data set of means, sums, medians, etc. `coll2` has better syntax than `collapse` and allows weights.

Options

`by(`*varlist*`)` specifies the groups over which the means, etc., are to be calculated. If not specified, the resulting data set will contain one observation. If specified, *varlist* may refer to either string or numeric variables.

`cw` specifies casewise deletion. If not specified, all observations possible are used for each calculated statistic.

Remarks

`coll2` takes the data in memory and creates a new data set from it, the new data set containing summary statistics of the original data. For instance, you might have individual-level data from the Census in which each observation is a person. `coll2` could take that data and create a one-observation data set containing the means of age, education, and income. More usefully, it could take the data and create a 50-observation data set containing means of age, education, and income for each state.

`coll2`'s syntax diagram makes using it appear more complicated than it is. To obtain a data set of means by state—assuming the original data contained the variables `age`, `educ`, `income`, `state`, and perhaps more—one need only type

```
. coll2 age educ income, by(state)
```

Variables `age`, `educ`, and `income` must be numeric (you are, after all, requesting the mean), but variable `state` could be numeric—taking on, say, values 1–50—or it could be a string—taking on values Alabama–Wyoming.

In any case, '`coll2 age educ income, by(state)`' calculates means because `coll2` assumes you want means if you do not specify otherwise. To make this explicit, one could type

```
. coll2 (mean) age educ income, by(state)
```

The parenthesized `(mean)` out front specifies the statistic to be calculated. Thus, if you wanted medians rather than means, you could type

```
. coll2 (median) age educ income, by(state)
```

`coll2` is not limited to calculation of only one type of statistic at a time. Typing

```
. coll2 (mean) age educ (median) income, by(state)
```

would return a data set containing the means of age and education but the median of income. What if one wanted both the average and median of income?

```
. coll2 (mean) age educ income (median) income, by(state)
error:
    income = (mean) income
    income = (p 50) income
name conflict
r(198);
```

`coll2` says no because you are requesting the new data set contain a variable named `income` that contains both the mean and median (`p 50` stands for 50th percentile). You can, however, specify the name of the variable to contain the statistic (called *target_var* in the syntax diagram).

```
. coll2 (mean) age educ income (median) medinc=income, by(state)
```

The new data set will contain variables named age, educ, income—containing means—and medinc—containing the median of income.

Finally, note that when you do not specify the *target_var*, the notation allows a varlist. If you wanted means of a lot of variables and medians of a few, you could type

```
. coll2 (mean) age-income (median) medinc=income medage=age, by(state)
```

Note that age-income refers to all the variables age through income (as shown by describe; see [5d] describe). Similarly, you can request many statistics on the same variable:

```
. coll2 (mean) age-income (median) medage=age inc50=income
                          (p25) inc25=income (p75) inc75=income, by(state)
```

The functions p25 and p75 return the 25th and 75th percentiles. (Function p50 is identical to median, so we could have typed (p50) rather than (median) if we preferred.)

Variable-wise vs. casewise deletion

In our original data, let us assume we have 25,000 persons for whom age is recorded but only 15,000 for whom we have information on income. That is, income contains missing values for 10,000 observations. Obviously, when we calculate income statistics, we can use only the 15,000 persons for whom income is known, but what about age? Is coll2 to use all 25,000 persons or just the 15,000 for whom income is also known?

By default, coll2 uses all the data it can, meaning the mean of age will be calculated using 25,000 observations. (This is how the existing collapse command works, too.) coll2's cw option, however, tells it to use casewise deletion; if a variable contains missing value for any variable being used, the observation is to be discarded in making all the calculations. In our Census example, specifying cw would change the calculation of the mean of age to using only the 15,000 observations for which income is also nonmissing.

```
. coll2 (mean) age-income (median) medage=age inc50=income
                          (p25) inc25=income (p75) inc75=income, by(state) cw
```

Weights

coll2 understands weights. Rather than starting with individual-level data and aggregating to state level, let's assume we started with state-level data and were aggregating to the region level. Let's also assume our original data set contained pop, the population of each state. It would then make sense to type

```
. coll2 (mean) age-income (median) medage=age inc50=income (p25) inc25=income
        (p75) inc75=income [weight=pop], by(region)
```

coll2 understands two kinds of weights—aweights and fweights—and defaults to aweights. Which you use makes no difference except for the functions sd (standard deviation), sum (sum of a variable), and count (the count of observations).

sd returns the bias-corrected standard deviation, which is based on the factor $\sqrt{N/(N-1)}$, N being the number of observations. For analytically weighted data, N is obtained as the physical number of observations in the data set. For frequency weighted data, N is the sum of the weight. Frequency weights are appropriate if the original data contains individual observations and analytic weights are appropriate if the original data contains cell means.

In the case of sum and count, the effect of the aweight–fweight choice is more important. The sum and count functions are defined such that sum/count is equal to the (weighted) mean. This yields the following individual definitions:

count:		
	unweighted:	_N, the number of physical observations
	aweight:	_N, the number of physical observations
	fweight:	$W = \sum w_j$, the sum of the user-specified weight
sum:		
	unweighted:	$\sum x_j$, the sum of the variable
	aweight:	$\sum v_j x_j$; $v_j = (w_j$ normalized to sum to _N)
	fweight:	$\sum w_j x_j$

For instance, consider a case where there are 25 physical observations in the data and a weighting variable that sums to 57. In the unweighted case, the weight is not specified and $N = 25$. In the analytically weighted case, N is still 25; the scale of the weight is irrelevant. In the frequency weighted case, however, $N = 57$, the sum of the weights.

Example 1. Obtain unweighted means and medians of age and income, by state, along with total population.

```
. coll2 (mean) age income (median) medage=age medinc=income (sum) pop, by(state)
```

Example 2. Obtain weighted means and medians of age and income, by state, along with total population. Use frequency weights.

```
. coll2 (mean) age income (median) medage=age medinc=income (count) pop
                [fweight=pop], by(state)
```

Note: Specifying '`(sum) pop`' would not have worked because that would have yielded the `pop`-weighted sum of `pop`. Specifying '`(count) age`' would have worked as well as '`(count) pop`' because `count` merely counts the number of nonmissing observations. The counts here, however, are frequency weighted and so equal the sum of `pop`.

Example 3. Same problem as example 2, but use analytic weights.

```
. coll2 (mean) age income (median) medage=age medinc=income (rawsum) pop
                [aweight=pop], by(state)
```

Note: Specifying '`(count) pop`' would not have worked because, with analytic weights, `count` would count numbers of physical observations. Specifying '`(sum) pop`' would not have worked because `sum` would calculate weighted sums (with a normalized weight). The `rawsum` function, however, ignores the weights and just sums the specified variable. `rawsum` would have worked as the solution to all three examples.

A real example

```
. use c:\stata\census.dta, clear
(1980 Census data by state)
. gen mrate = marriage/pop18p
. coll2 (median) medage mrate (mean) avgmrate=mrate [aw=pop], by(region)
. list
       region     medage      mrate   avgmrate
  1.       NE       31.9   .0108964    .011108
  2.  N Cntrl       29.9   .0133482   .0135694
  3.    South       29.6   .0163545   .0164672
  4.     West       29.9   .0122035   .0174117
. describe
Contains data from c:\stata\census.dta
  Obs:     4 (max=  5067)                    1980 Census data by state
 Vars:     4 (max=    99)
Width:    14 (max=   200)
  1. region       int    %8.0g       cenreg   Census region
  2. medage       float  %9.0g                (p 50) medage
  3. mrate        float  %9.0g                (p 50) mrate
  4. avgmrate     float  %9.0g                (mean) mrate
Sorted by:  region
Note:  Data has changed since last save
```

gr17	Switching graphics windows in Unix

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

The following four simple routines can be useful when working in Unix with `STATAPD` set to `pd.X`.

`win1` and `win2` without any arguments each create a graphic window and make the new window active. The two windows are in nonoverlapping positions on the right of the screen. These commands can be followed on the same line by any Stata command. In particular they are intended to be used together with an ado-file that draws a graph. When used in this way `win1` and `win2` simply make the corresponding window active (providing it exists), so that the new graph (and any future graphs) will appear in it.

`gr1` and `gr2` can be used instead of `graph`. They assume the existence of graphic windows *win1* and *win2* respectively and place the graph in the appropriate window. After plotting, `gr1` and `gr2` "`set graphics on default`".

Example session

```
. gr mpg                              /* plots histogram in default window */
. win1                                /* creates a new window win1 */
. gr                                  /* plots the histogram in win1 */
. win2                                /* creates window win2 */
. gr2 weight length                   /* plots twoway graph in win2 */
. gr1 mpg price                       /* plots twoway graph in win1 */
. gr                                  /* plots previous graph in default window */
. win1 ksm weight length, lowess      /* plots smooth in win1 */
```

Remarks

1. `win1` and `win2` leave the graphics set to the window with the same name, whereas `gr1` and `gr2` return the graphics to the default window.
2. `gr1` and `gr2` do not `capture` the graph command. Thus if there is an error in the graph command, graphics will *not* be set to default.

ip6.2 | Storing matrices as variables

William M. Sribney, Stata Corporation, FAX 409-696-4601, EMAIL stata@stata.com

`svmat` takes a matrix and stores its columns as new variables. It is the reverse of the `mkmat` command which creates a matrix from existing variables. `matname` renames the rows and columns of a matrix. After using `mkmat` to make a matrix, `matname` can be used to correctly name the rows and columns for use with commands such as `matrix post` or `matrix score`. `matname` differs from the `matrix rownames` and `matrix colnames` commands in that `matname` expands varlist abbreviations and also allows a restricted range for the rows or columns.

`svmat` [*type*] **A** [, names(col | eqcol | matcol | *string*)]

`matname` **A** *namelist* [, rows(*range*) columns(*range*) explicit]

where **A** is the name of an existing matrix; *type* is a storage type for the new variables; and *namelist* is either (1) a *varlist*, i.e., names of existing variables possibly abbreviated; (2) `_cons` and the names of existing variables possibly abbreviated; or (3) arbitrary names when the `explicit` option is specified.

Options

`names(col | eqcol | matcol | `*string*`)` specifies how the new variables are to be named. `names(col)` uses the column names of the matrix to name the variables. `names(eqcol)` uses the equation names prefixed to the column names. `names(matcol)` uses the matrix name prefixed to the column names. `names(`*string*`)` names the variables *string*1, *string*2, ..., *string* n, where *string* is a user-specified *string* and n is the number of columns of the matrix. If `names()` is not specified, the variables are named **A**1, **A**2, ..., **A**n, where **A** is the name of the matrix. If necessary, names will be truncated to 8 characters; if these names are not unique, an error message will be returned.

`rows(`*range*`)` and `columns(`*range*`)` specify the rows and columns of the matrix to rename. The number of rows or columns specified must be equal to the number of names in *namelist*. If both `rows()` and `columns()` are given, then the specified rows are named *namelist* and the specified columns are also named *namelist*. The range must be given in one of the following forms:

`rows(.)`	renames all the rows;
`rows(2..8)`	renames rows 2 through 8;
`rows(3)`	renames only row 3;
`rows(4...)`	renames row 4 to the last row.

If neither `rows()` nor `columns()` is given, then `rows(.) columns(.)` is the default. That is, the matrix must be square, and both the rows and columns are named *namelist*.

`explicit` suppresses the expansion of varlist abbreviations and omits the verification that the names are those of existing variables. That is, the names in *namelist* are used explicitly and can be any valid row or column names.

Example

Let us get the vector of coefficients from a regression, use `svmat` to save the vector as a new variable, save the data set, load the data set back into memory, use `mkmat` to create a vector from the variable, and finally, use `matname` to rename the columns of the row vector.

```
. quietly regress mpg weight gratio foreign
. matrix b = get(_b)
. matrix list b
b[1,4]
        weight      gratio     foreign       _cons
y1  -.00613903   1.4571134  -2.2216815   36.101353
. matrix c = b'
. svmat double c, name(bvector)
. list bvector1 in 1/5
       bvector1
  1. -.00613903
  2.  1.4571134
  3. -2.2216815
  4.  36.101353
  5.          .
. save example
file example.dta saved
. use example
. mkmat bvector1 if bvector1~=.
. matrix list bvector1
bvector1[4,1]
      bvector1
r1  -.00613903
r2   1.4571134
r3  -2.2216815
r4   36.101353
. matrix d = bvector1'
. matname d wei gr for _cons, c(.)
. matrix list d
d[1,4]
              weight      gratio     foreign       _cons
bvector1  -.00613903   1.4571134  -2.2216815   36.101353
```

sed9	Symmetric nearest neighbor linear smoothers

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

`running` produces a locally weighted least squares fit and may be regarded as an alternative to `ksm`. Unlike `ksm` it does not permit arbitrary weights. However it calculates the whole vector of fitted values in one step rather than one observation at a time. Thus `running` may be used even when the number of observations is very large.

`running` sorts the data by the x-variable and forms local neighborhoods consisting of an (x, y) pair together with the k nearest x-neighbors on either side of x. At the ends of the x-range the neighborhoods will be asymmetric, consisting of k nearest neighbors on one side and the remaining (fewer than k) observations on the other side.

The smoothed value of each point is based on a least squares fit using the observations in the neighborhood of the point. The fitted model has either a constant only (running mean) or a constant and a linear term (running line). The program permits the smooth values to be smoothed again. For the running mean, this is equivalent to smoothing with a different kernel. If the current smooth is based on a kernel f and it is re-smoothed using a kernel g, then the resulting smooth is equivalent to having smoothed the original data with the convolution $f * g$. Since the kernel g is always uniform, one can calculate the resulting kernel. For instance, if f and g are both uniform on $(-k, k)$, then $f * g$ is triangular on $(-2k, 2k)$. There is probably little to be gained from repeating this procedure more than three times.

The span option controls the proportion of observations given nonzero weight in the kernel that produces the final smooth from the original data. The knn option controls the number of observations with nonzero weight in each half of this kernel, i.e., $\texttt{span} = (2\texttt{knn} + 1)/N$.

If there are tied x-values, running replaces the fit in all of them by the mean of the fitted values for each of the tied x's. This prevents the unsightly vertical lines that ksm produces with tied data.

To avoid bias at the ends of the data due to asymmetric neighborhoods, running makes the smoothed values at the ends of the x-range missing if the mean option is chosen.

Syntax

running *yvar* [*xvar*] [if *exp*] [in *range*] [, double gen(*newvar*) nograph knn(*#*)
logit mean repeat(*#*) span(*#*) *graph_options*]

Description

running smooths *yvar* on *xvar*. By default the smoothed version is a running line: a running mean is also available. A graph of *yvar* together with its smooth is plotted against *xvar*, unless suppressed. If *xvar* is not provided, then *yvar* is smoothed against the ordered observations. A new variable containing the smoothed values may be generated.

Options

double doubles the value of repeat. Thus if repeat is not specified, double is equivalent to repeat(2).

gen(*newvar*) creates *newvar* containing the smoothed values of *yvar*. Note that this will be on a logit scale if logit is used.

nograph suppresses the display of the graph.

knn(#) specifies the number of nearest neighbors on each side to be used. The value of knn is stored in S_1. The greater the value, the greater the smoothing. If span is specified, knn is ignored.

logit transforms the smooth and plots the y-axis on a logit scale. The observations are automatically jittered in the vertical scale and are plotted just outside the range of the smoothed curve.

mean specifies running-mean least-squares smoothing; default is running-line.

repeat(#) specifies the number of times the data are to be smoothed. The default is 1. Increasing repeat increases the time taken to calculate the smooth, but should improve the result.

span(#) specifies the span or proportion of the data to be used in the symmetric nearest neighbors. If span is specified, knn is defined to be $(N * \texttt{span} - 1)/2$, where N is the number of observations. If both span and knn are specified, knn is ignored. The span is stored in S_2.

graph_options are any of the options allowed with graph, twoway. The graph option (the default) plots *yvar* followed by its smooth against *xvar*. The default options are s(oi) and c(.l). If there are more than 1000 observations, we advise using s(.i) instead.

Examples

```
. use auto, clear
(1978 Automobile Data)
. running foreign mpg, ylabel xlabel
```

(graph appears, see Figure 1)

```
. running foreign mpg, ylabel(-4,-2,0,2) xlabel logit yline(0) span(.9) repeat(3)
```

(graph appears, see Figure 2)

Methods and Formulae

The default value of knn is defined to be $N^{0.67}$ where N is the number of nonmissing observations. The actual span of the smooth is $r * \texttt{int}(\texttt{knn} * r^{-0.5} + 0.5)$ where r is the value of repeat and int is the function yielding the integer part of its argument.

The data are sorted according to *xvar* and the subscripts refer to the ordered data. The running line smoother calculates the intercept (α_i) and slope (β_i) for the ith ordered data point using the formulae

$$\widehat{\beta}_i = \frac{\frac{1}{n}\sum x_j y_j - \frac{1}{n^2}\sum x_j \sum y_j}{\frac{1}{n}\sum x_j^2 - (\frac{1}{n}\sum x_j)^2}$$

$$\widehat{\alpha}_i = \frac{1}{n}\sum y_j - \frac{\widehat{\beta}_i}{n}\sum x_j$$

where the summations are from $i-k$ to $i+k$ and n is the number of terms in the summation ($= 2k+1$) for $k < i < N-k$. In the tails ($i <= k$ or $i >= N-k$), fewer than k terms are included on the "short" side of i. $k = \texttt{int}(\texttt{knn} * r^{-0.5} + 0.5)$.

For computational stability it is sensible to center the *xvar* and *yvar*, but in order to take advantage of Stata's ability to perform calculations on the entire range of a variable much faster than it can explicitly loop through all the observations, the program does not use a different centering for each i. Thus, for instance, for $k < i < N-k$ one can generate $(\sum x_j^2)/n$ by typing

```
. generate sxx=sum(x*x)
. generate rsxx=(sxx[_n+k] - sxx[_n-k-1])/(2*k+1)
```

Full details of the way in which `running` works on the ends of the data and of its treatment of ties in *xvar* are best obtained by studying the ado-file.

Discussion

Stata provides very few built-in smoothing routines. The `m` and `s` arguments of the `connect` option of the `graph` command are the only ones I know of. Several smoothing routines have appeared in the STB, however, and some of these have been incorporated into Stata as ado-files, most notably `ksm` and `smooth`. For large data sets, `running` is much faster than either `ksm` or `smooth`. The reason for the improved speed over `ksm` is that `ksm` calculates the smooth for one observation at a time. In contrast, `running` calculates the fit for the whole vector at once. The saving can be considerable, as illustrated in Table 1. Each program was used to smooth `y=sin(6*x)+z` against `x`, where `x` is a pseudo uniform random variable on (0,1) and `z` is a pseudo standard normal random variable. The last row of Table 1 gives the slope coefficient from the regression of the logarithm of the execution time on the logarithm of the number of observations. These calculations confirm that `ksm` is order N^2 whereas `running` and `smooth` are order N.

Table 1. Execution times (in seconds) of different smoothers.

Number of obs.	`running`	`ksm, line`	`ksm, lowess`	`smooth, 4253H`	`smooth, 3rssh`
8000	12	4248	7261	211	361
4000	6	1101	1878	117	183
2000	3	299	507	59	91
1000	2	87	145	30	45
coef.	0.9	1.9	1.9	0.9	1.0

Such time comparisons are not completely fair since the `lowess` option of `ksm` and all options of `smooth` are by necessity more computer intensive than the smooth carried out by `running`. I would suggest however that there are very few circumstances in which one would really prefer `ksm` to `running`. In this context it should be noted that the `lowess` option of `ksm` does not perform the robust weighting proposed in Cleveland's 1979 paper.

`smooth` is a very different smoother. First, it does not take an *xvar* argument. In our notation the implicit *xvar* is taken to be `_n`. In other words, *yvar* is implicitly assumed to be a series measured at equally spaced time points. Second, `smooth` is a nonlinear smoother. The use of the median makes the smooth resistant to gross outliers in *yvar*. Note however that the span of `smooth` does not increase with the number of observations. Implicitly, `smooth` is designed for asymptotics in which the length of time over which observations are made increases, but the frequency of observations remains fixed. By contrast `running` implicitly assumes that as the number of observations increases so does their frequency.

The odd-looking formula for choosing the span is designed to produce smoothers that are as similar as possible for different values of `repeat`. The shape of the kernel is complicated by several factors: (i) the ends of the data; (ii) the spacing of the x-values; and (iii) the use of running lines. The comments that follow refer to the special cases of (a) an evenly spaced x-variable, or (b) the kernel of the running mean viewed as a function of the ranks of the x-variable. They do not apply to the ends of the data.

All kernels are centered around the point being estimated. It makes sense then to control the variance of the kernels. As remarked earlier, the kernel resulting from r passes of the smoother is equivalent to the convolution of r uniform kernels. The variance of the kernel is then $r * s^2/12$, where s is the length of support of the uniform kernel. Thus if $s = s_0 * r^{-0.5}$, then the variance of the resulting kernel will be independent of r. Figure 3 illustrates the similarity of kernels for $r = 2$, 3, 4, and 9. This figure was produced by the following Stata commands.

```
. clear
. set obs 999
obs was 0, now 999
. generate x=_n/1000
. generate y=(_n==500)
. running y x, gen(y2) repeat(2) nograph
. running y x, gen(y3) repeat(3) nograph
. running y x, gen(y4) repeat(4) nograph
. running y x, gen(y9) repeat(9) nograph
. graph y2 y3 y4 y9 x if x>.3 & x<.7, symbol(iiii) connect(llll) ylabel
(graph appears, see Figure 3)
```

References

Cleveland, W. S. 1979. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association* 74: 829–836.

Figures

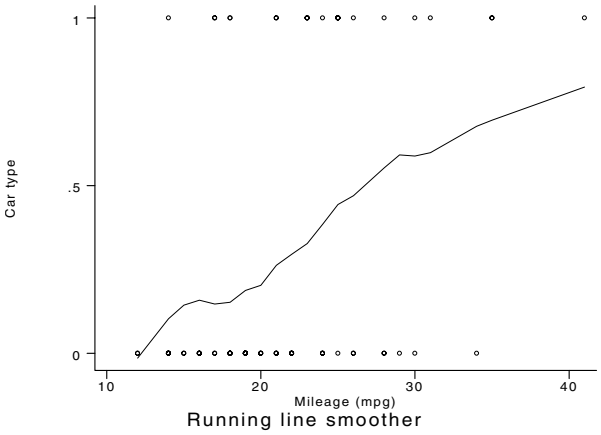

Figure 1

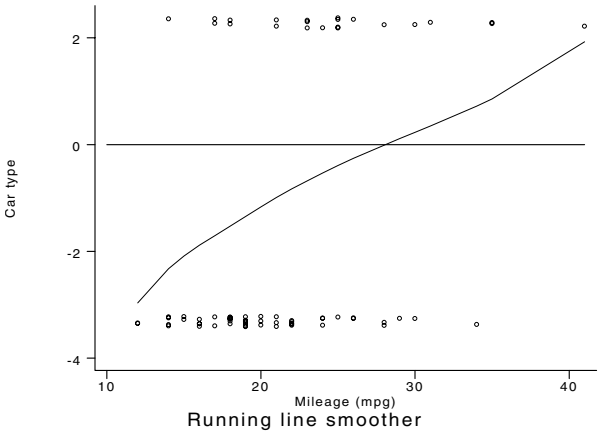

Figure 2

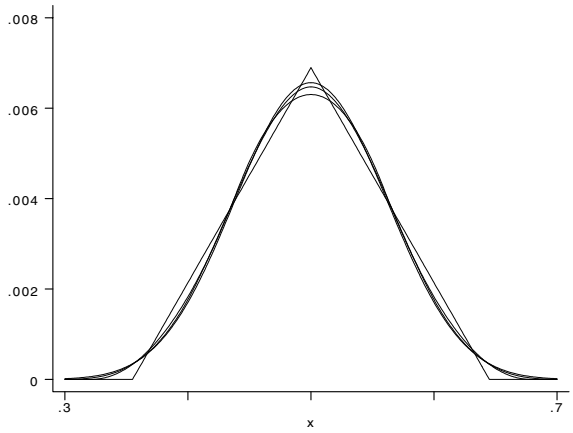

Figure 3

sg21.1	Equivalency testing: Correction

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

I have discovered and corrected a minor bug in the programs described in *sg21* (Goldstein 1994). This bug was unlikely to affect results. Corrected versions of the programs are supplied on this month's distribution diskette.

Reference

Goldstein R. 1994. sg21: Equivalency testing. *Stata Technical Bulletin* 17: 13–18.

sg26.2	Calculating and graphing fractional polynomials

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

In *sg26*, Royston and Altman (1994a, 1994b) introduce fractional polynomials, closed-form functions that can describe a wide array of flexible forms using very few parameters. The set of fractional polynomials contains the conventional polynomials as special cases. In addition, fractional polynomials offer impressive flexibility in low-order models, the ability to avoid such artifacts as waviness and endpoint effects in higher-order models, and the ability to model asymptotes—features that are lacking in the conventional polynomials. As a consequence, fractional polynomials combine many of the advantages of nonlinear models with the ease of application and interpretation of polynomial models.

Royston and Altman illustrate the use of fractional polynomials to model curved regression relationships, and they present several ado-files to estimate regressions containing fractional polynomials. I have used these programs for several months, and I have found them very useful. However, because fractional polynomials are novel, I often find it difficult to mentally convert the estimated coefficients of a fractional polynomial into an image of the fitted curve. Royston and Altman provide a program to graph the most-recently-estimated fractional polynomial fit, but I find I need additional tools. In particular, small changes in coefficients often produce dramatic qualitative changes in the shape of a fractional polynomial, effects that cannot be foreseen just by graphing the current estimate.

This insert presents `fpcurve`, a program that calculates and graphs arbitrary fractional polynomials. The next section briefly reviews Royston and Altman's definition of fractional polynomials. The succeeding section demonstrates the use of `fpcurve`.

Fractional polynomials

A fractional polynomial of degree m is a parametric function of the form

$$\phi_m(X;\beta;p) = \beta_0 H_0(X;p_0) + \beta_1 H_1(X;p_1) + \cdots + \beta_m H_m(X;p_m),$$

where $H_0(X) \equiv 1$, $p_0 \equiv 0$, and

$$H_j = \begin{cases} X^{(p_j)}, & \text{if } p_j \neq p_{j-1}; \\ H_{j-1}(X) \ln X, & \text{if } p_j = p_{j-1}. \end{cases}$$

$X^{(p)}$ denotes the Box–Tidwell transformation:

$$X^{(p)} = \begin{cases} X^p, & \text{if } p \neq 0; \\ \ln X, & \text{if } p = 0. \end{cases}$$

For instance, if $m = 4$ and $p = (0, -\frac{1}{2}, 2, 2, 2)$,

$$\phi(X) = \beta_0 + \beta_1 X^{-\frac{1}{2}} + \beta_2 X^2 + \beta_3 X^2 \ln X + \beta_4 X^2 (\ln X)^2$$

As this definition and example make clear, the powers in a fractional polynomial need not be positive or integers, and powers of $\ln X$ can be interacted with powers of X. In practice, choosing powers from the set $p \in \{-3, -2, -1, -\frac{1}{2}, -\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{2}, 1, 2, 3\}$ appears to be adequate.

Using fpcurve to calculate and graph arbitrary fractional polynomials

The conventional polynomials are the workhorses of regression analysis, and applied statisticians can easily translate the estimated coefficients of a conventional polynomial into a mental image of the estimated regression curve. Fractional polynomials are less familiar to researchers. Moreover, their extreme flexibility makes it difficult to predict their shape just from examining their estimated coefficients.

`fpcurve` calculates and graphs arbitrary fractional polynomials. `fpcurve` can be used to examine the shape of an estimated fractional polynomial. More importantly, `fpcurve` can be used to observe the effects of small (or large) changes in the coefficients of the curve.

The syntax for `fpcurve` is

`fpcurve` [[*newvar*] `=` *exp*] , `coefficients(#[,#[,...]])` `powers(#[,#[,...]])`
[`constant(#)` `nograph` `logpowers(#[,#[,...]])` `obs(#)` `range(#,#)` *graph_options*]

While this syntax diagram may look a bit daunting, `fpcurve` is actually very easy to use. To prove this point, we use `fpcurve` to draw an ordinary quadratic function.

```
. describe
Contains data
  Obs:     0 (max= 30488)
 Vars:     0 (max=    99)
Width:     0 (max=   200)
Sorted by:
. fpcurve, range(0,2) power(1,2) coef(1,-0.5) title(Quadratic curve)
```

(graph appears, see Figure 1)

`fpcurve` displayed the simple quadratic function

$$y = x - \frac{1}{2}x^2$$

over the domain $x \in [0, 2]$. In this form, `fpcurve` does not alter the current data set. Indeed, there need not be any data to begin with. Note also the way that `fpcurve` labels the axes to remind you of the function definition and the domain of the function.

`fpcurve` will optionally create a new variable that contains the values of the fractional polynomial over the specified range. Instead of specifying a range, the user may also specify an expression involving existing variables, for instance,

```
. fpcurve y = 2*x*sin(z/q), ...
```

These two features are independent; a new variable can be generated when the domain is specified by the `range()` option, and the range can be defined in terms of existing variables when no new variable is generated.

The options to `fpcurve` are

`coefficients(#[,#[,...]])` lists the coefficients of each of the terms excluding the constant. The coefficients can be separated either by commas or by spaces.

`constant(#)` specifies the constant term. If this option is omitted, the constant is set to zero.

`nograph` suppresses the display of the graph.

`logpowers(#[,#[,...]])` specifies the powers of the $\ln X$ terms directly, overriding the usual rules for promoting the powers of $\ln X$. This option generalizes the definition of fractional polynomials and was added to accommodate a user request. You are unlikely to ever use the `logpowers()` option.

`obs(#)` is used with the `range` option to specify the number of observations to create temporarily when the data set is empty. More observations produce a smoother graph but take more time to calculate and draw. The default value is 1000.

`powers(#[,#[,...]])` specifies the sequence of powers to apply to each term in the fractional polynomial.

`range(#,#)` specifies the range of x-values to use when the data set is empty.

Example

Continuing with the example from above, we can create a data set containing the x and y values displayed in Figure 1.

```
. range x 0 2 100
obs was 0, now 100
. fpcurve y = x, power(1,2) coef(1,-0.5) nograph
. list x y in f/5
             x           y
  1.         0           0
  2.   .020202   .01999796
  3.   .040404    .0395878
  4.  .0606061   .05876952
  5.  .0808081   .07754311
```

Now we calculate some more exotic fractional polynomials.

```
. fpcurve y1 = x, power(2,2) coef(.85,-.5) nograph
. fpcurve y2 = x, power(-1/2,1/3,3) coef(1,5,-.1) nograph
. graph y1 y2 x, connect(ll) symbol(..) rescale title(Two fractional polynomials)
```

(graph appears, see Figure 2)

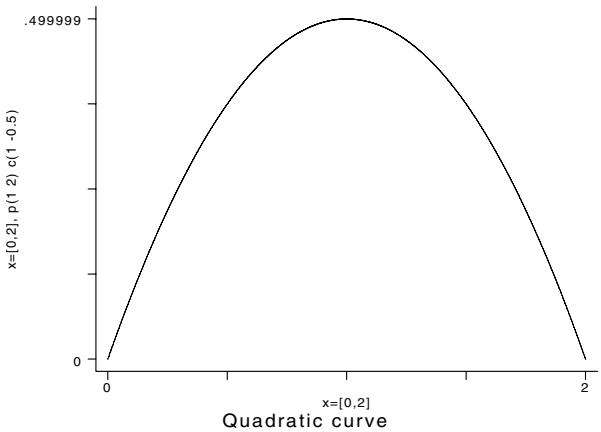

Figure 1

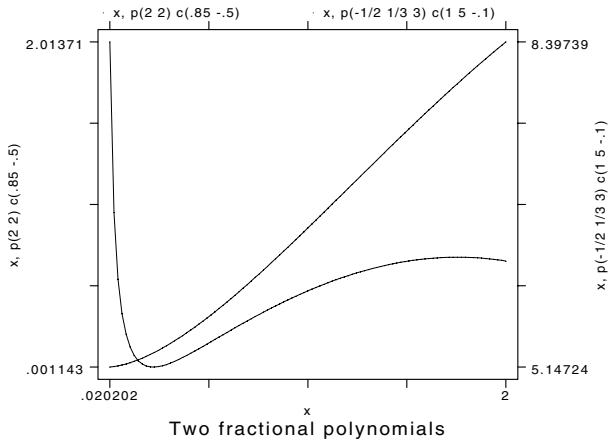

Figure 2

References

Royston, P. and D. G. Altman. 1994a. sg26: Using fractional polynomials to model curved regression relationships. *Stata Technical Bulletin* 21: 11–23.

——. 1994b. sg26.1: Fractional polynomials: Correction. *Stata Technical Bulletin* 22: 11–12.

sg32	Variance inflation factors and variance-decomposition proportions

James W. Hardin, Stata Corporation, FAX 409-696-4601, EMAIL stata@stata.com

Problems arise in regression when the predictors are highly correlated. In this situation, there may be a significant change in the regression coefficients if one adds or deletes an independent variable. The estimated standard errors of the fitted coefficients are inflated, or the estimated coefficients may not be statistically significant even though a statistical relation exists between the dependent and independent variables.

Data analysts rely on these facts to check informally for the presence of multicollinearity. In this article, I present two commands to analyze the independent variables in a regression for collinearity. The `vif` command calculates the variance inflation factors and tolerances for each of the independent variables, and the `colldiag` command calculates the variance-decomposition proportions based on a singular value decomposition of the independent variables.

Calculating variance inflation factors with vif

`vif` is a post-estimation command that must follow `fit`. `vif` calculates the variance inflation factors (VIFs) for the independent variables specified in a linear regression model. The syntax of `vif` is to type `vif` by itself as it takes no arguments nor options.

The output shows each of the variance inflation factors along with their reciprocals. Some analysts compare the reciprocals to a predetermined tolerance. In the comparison, if the reciprocal of the VIF is smaller than the tolerance, the associated predictor variable is removed from the regression model. However, most analysts rely on informal rules of thumb applied to the VIF. According to these rules, there is evidence of multicollinearity if

1) The largest VIF is greater than 10 (some choose a more conservative threshold value of 30).
2) The mean of all of the VIFs is considerably larger than 1.

Example 1

We examine a model fit using the ubiquitous automobile data set:

```
. fit price mpg rep78 trunk hdroom length turn displ gratio
  Source |       SS       df       MS                  Number of obs =      69
---------+------------------------------                F(  8,    60) =    6.33
   Model |   264102049     8  33012756.2                Prob > F      =  0.0000
Residual |   312694909    60  5211581.82                R-squared     =  0.4579
---------+------------------------------                Adj R-squared =  0.3856
   Total |   576796959    68  8482308.22                Root MSE      =  2282.9
------------------------------------------------------------------------------
   price |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
     mpg |    -144.84   82.12751     -1.764   0.083      -309.1195    19.43947
   rep78 |   727.5783   337.6107      2.155   0.035       52.25641      1402.9
   trunk |   44.02061    108.141      0.407   0.685      -172.2935    260.3347
  hdroom |  -807.0996   435.5802     -1.853   0.069       -1678.39    64.19057
  length |  -8.688914   34.89848     -0.249   0.804      -78.49626    61.11843
    turn |  -177.9064   137.3455     -1.295   0.200      -452.6382     96.8255
   displ |   30.73146   7.576952      4.056   0.000        15.5753    45.88762
  gratio |   1500.119   1110.959      1.350   0.182      -722.1302    3722.368
   _cons |   6691.976   7457.906      0.897   0.373      -8226.057    21610.01
------------------------------------------------------------------------------

. vif
Variable |      VIF      1/VIF
---------+----------------------
     mpg |      3.03    0.330171
   rep78 |      1.46    0.686147
   trunk |      2.88    0.347444
  hdroom |      1.80    0.554917
  length |      8.22    0.121614
    turn |      4.85    0.205997
   displ |      6.50    0.153860
  gratio |      3.45    0.290068
---------+----------------------
Mean VIF |      4.02
```

The results here are mixed. While we do not have any VIFs greater than ten, the mean VIF is greater than one, though not considerably so. One could continue the investigation of collinearity, but given that other authors advise that collinearity is only a problem when VIFs exist that are greater than 30 (contradicting our rule above), we will not do so here.

Example 2

This example comes from a data set described in Neter, Wasserman, and Kutner (1989) which examines bodyfat as modeled by caliper measurements on the tricep, midarm, and thigh.

```
. use bodyfat
(Body Fat)
. fit bodyfat tricep thigh midarm
  Source |       SS       df       MS                  Number of obs =      20
---------+------------------------------                F(  3,    16) =   21.52
   Model |  396.984607     3  132.328202                Prob > F      =  0.0000
Residual |  98.4049068    16  6.15030667                R-squared     =  0.8014
---------+------------------------------                Adj R-squared =  0.7641
   Total |  495.389513    19  26.0731323                Root MSE      =    2.48
------------------------------------------------------------------------------
 bodyfat |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  tricep |   4.334085   3.015511      1.437   0.170      -2.058512    10.72668
   thigh |  -2.856842   2.582015     -1.106   0.285      -8.330468    2.616785
  midarm |  -2.186056   1.595499     -1.370   0.190      -5.568362     1.19625
   _cons |   117.0844   99.78238      1.173   0.258      -94.44474    328.6136
------------------------------------------------------------------------------
. vif
Variable |      VIF      1/VIF
---------+----------------------
  tricep |    708.84    0.001411
   thigh |    564.34    0.001772
  midarm |    104.61    0.009560
---------+----------------------
Mean VIF |    459.26
```

In this example, we see very strong evidence of multicollinearity in our model. Further investigation reveals that the measurements on the thigh and the tricep are highly correlated:

```
. corr tricep thigh midarm
(obs=20)
        |   tricep    thigh   midarm
--------+---------------------------
  tricep|   1.0000
   thigh|   0.9238   1.0000
  midarm|   0.4578   0.0847   1.0000
```

If we remove the predictor `tricep` from the model (since it had the highest VIF), we get

```
. fit bodyfat thigh midarm
  Source |       SS       df       MS                  Number of obs =      20
---------+------------------------------                F(  2,    17) =   29.40
   Model |  384.279748     2  192.139874                Prob > F      =  0.0000
Residual |  111.109765    17  6.53586854                R-squared     =  0.7757
---------+------------------------------                Adj R-squared =  0.7493
   Total |  495.389513    19  26.0731323                Root MSE      =  2.5565
------------------------------------------------------------------------------
 bodyfat |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
   thigh |   .8508818   .1124482      7.567   0.000       .6136367    1.088127
  midarm |   .0960295   .1613927      0.595   0.560      -.2444792    .4365383
   _cons |  -25.99696    6.99732     -3.715   0.002      -40.76001    -11.2339
------------------------------------------------------------------------------
```

```
. vif
Variable |      VIF      1/VIF
---------+----------------------
   thigh |      1.01    0.992831
  midarm |      1.01    0.992831
---------+----------------------
Mean VIF |      1.01
```

Note how the coefficients change and the estimated standard errors for each of the regression coefficients becomes much smaller. The calculated value of R^2 for the overall regression for the subset model does not appreciably decline when we remove the correlated predictor. Removing an independent variable from the model is one way in which the analyst may deal with multicollinearity. Other methods include ridge regression, weighted least squares, and restricting the use of the fitted model to data that follows the same pattern of multicollinearity. In economic studies, it is sometimes possible to estimate the regression coefficients from different subsets of the data using cross-section and time series.

Calculating collinearity diagnostics with colldiag

`colldiag` is a post-estimation command that follows the `fit` command. The calculations performed by `colldiag` require storing the explanatory variables from the regression in matrices. As a consequence, the user is limited to relatively small data sets that do not exceed the size of the maximum value of `matsize`. This restriction is a severe limitation for users of Small Stata as the `matsize` is fixed at 40. For users of Intercooled Stata, this restriction means that the command will only work for data sets with less than 800 observations for version 4.0 (800 for Unix and 400 for all other platforms) and 400 observations for version 3.1.

The syntax of `colldiag` is

`colldiag` [`, nocons`]

The data are loaded in scaled form into a matrix $\mathbf{X}$ and the singular value decomposition is evaluated using Stata's matrix commands. From the condition numbers that are calculated for the matrix, the variance of the coefficients for each of the independent variables may be apportioned among the principal components of the $\mathbf{X}'\mathbf{X}$ matrix.

Belsley, Kuh, and Welsch (1980) provide the algorithm used in obtaining the matrix of variance-decomposition proportions and advise that the analyst should examine the dependencies by focusing on those components with condition numbers η exceeding some chosen threshold η^*. Typical values for the threshold are 10, 15, or even 30. In the case of orthogonal predictors, the variance-decomposition proportion matrix (the variance-decompositions are denoted by π) is an identity matrix in which each of the variates is completely explained by a unique principal component.

The proportions π are more easily understood in the context of the following example.

Example 3

We have data on men involved in a physical fitness course. The purpose of the study is to model the oxygen uptake rate by the age, weight, time to run one and a half miles, the heart rate while resting, heart rate while running, and the maximum heart rate while running.

```
. describe
Contains data from fitness.dta
  Obs:    31 (max= 50172)                    Fitness data
 Vars:     7 (max=    99)                    16 Nov 1994 15:47
Width:    28 (max=   200)
  1. age          float  %9.0g
  2. weight       float  %9.0g
  3. oxy          float  %9.0g
  4. runtime      float  %9.0g
  5. rstpulse     float  %9.0g
  6. runpulse     float  %9.0g
  7. maxpulse     float  %9.0g
Sorted by:
```

```
. fit oxy age weight runtime rstpulse runpulse maxpulse
  Source |       SS       df       MS                  Number of obs =      31
---------+------------------------------                F(  6,    24) =   22.43
   Model |  722.543528     6  120.423921                Prob > F      =  0.0000
Residual |  128.837947    24   5.3682478                R-squared     =  0.8487
---------+------------------------------                Adj R-squared =  0.8108
   Total |  851.381475    30  28.3793825                Root MSE      =  2.3169
------------------------------------------------------------------------------
     oxy |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
     age |  -.2269738   .0998375     -2.273   0.032      -.4330282   -.0209194
  weight |  -.0741774   .0545932     -1.359   0.187      -.1868521    .0384974
 runtime |  -2.628653   .3845622     -6.835   0.000       -3.42235   -1.834955
rstpulse |  -.0215336   .0660543     -0.326   0.747      -.1578629    .1147957
runpulse |  -.3696278   .1198529     -3.084   0.005      -.6169921   -.1222634
maxpulse |   .3032171   .1364952      2.221   0.036       .0215049    .5849294
   _cons |   102.9345   12.40326      8.299   0.000       77.33541    128.5335
------------------------------------------------------------------------------
```

As there may be a dependency among the pulse variables, we investigate with the new diagnostic tool.

```
. colldiag
Proportion of variance associated with the decomposition
  Cond   |
 Number  |     age    weight   runtime  rstpulse  runpulse  maxpulse     _cons
---------+-----------------------------------------------------------------------
       1 |  0.0002    0.0002    0.0002    0.0003    0.0000    0.0000    0.0000
 19.2909 |  0.1463    0.0104    0.0252    0.3906    0.0000    0.0000    0.0022
 21.5007 |  0.1501    0.2357    0.1286    0.0281    0.0012    0.0012    0.0006
 27.6212 |  0.0319    0.1831    0.6090    0.1903    0.0015    0.0012    0.0064
 33.8292 |  0.1128    0.4444    0.1250    0.3648    0.0151    0.0083    0.0013
 82.6376 |  0.4966    0.1033    0.0975    0.0203    0.0695    0.0056    0.7997
 196.786 |  0.0621    0.0228    0.0146    0.0057    0.9128    0.9836    0.1898
```

By examining the matrix of variance-decomposition proportions, we should note that

1) **Near Dependency:** The independent variable will have a degraded coefficient because of a near dependency if (i) the condition number η is greater than the threshold value η^* and (ii) it is one of two or more variates with associated variance-decomposition proportions in excess of some threshold value π^*, such as .50.

2) **Competing Dependency:** Those variates whose aggregate variance-decomposition proportion exceed the threshold value π^* are involved in at least one of the dependencies. The aggregate is formed by adding the π values over the competing condition numbers (condition numbers of the same order of magnitude that exceed the threshold value η^*).

3) **Dominating Dependency:** A dominating dependency occurs when the condition number is an order of magnitude larger than the other condition numbers. This can obscure information about the variate's simultaneous involvement in a weaker dependency. In this case, additional analysis is warranted to investigate the relationships of all potentially involved variates.

If we use 30 as our value for η^* and .50 as our threshold for π^*, we see that points 1) and 2) are exhibited in our output. There is a near dependency with a condition number greater than 196 involving the `runpulse` and `maxpulse` variables. The competing dependency is for the condition numbers 33.8292 and 82.6376 which are of the same order of magnitude and both exceed our threshold value of 30. Aggregating the variance-decomposition proportions, we note that `age` ($.1128 + .4966 = .6014$), `weight` ($.4444 + .1033 = .5477$), and the constant ($.0013 + .7997 = .8010$) are involved in a competing dependency.

Since we have three near dependencies (three condition numbers greater than $\eta^* = 30$), we should be able to express three of our independent variables in terms of the remaining four. How do we choose the variates for which to solve? Beginning with the largest condition number, we see that we should choose either `runpulse` or `maxpulse`. Since `maxpulse` has the remainder of its variance determined in a more removed dependency, we can choose it as our first dependent variable in the auxiliary regression. Now, since we are not as interested in the constant term, we may choose the `weight` and `age` as our remaining pivots.

For example, we may readily see the linear dependence of the `maxpulse` variate on the remaining independent variables using `fit`:

```
. fit maxpulse runtime rstpulse runpulse
  Source |       SS       df       MS                  Number of obs =      31
---------+------------------------------                F(  3,    27) =   59.81
   Model |  2189.89997     3  729.966658                Prob > F      =  0.0000
Residual |   329.51938    27  12.2044215                R-squared     =  0.8692
---------+------------------------------                Adj R-squared =  0.8547
   Total |  2519.41935    30  83.9806452                Root MSE      =  3.4935
------------------------------------------------------------------------------
maxpulse |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
 runtime |  -.4869733   .5239808     -0.929   0.361      -1.562093    .5881464
rstpulse |   .0034019    .096809      0.035   0.972      -.1952338    .2020376
runpulse |   .8508719   .0676534     12.577   0.000       .7120586    .9896852
   _cons |   34.40121   10.70103      3.215   0.003       12.44452     56.3579
------------------------------------------------------------------------------
```

One should continue analyzing the auxiliary regressions in order to assess the effect of the linear dependence among the independent variables.

Example 4

Returning to the data on bodyfat, `colldiag` leads us to the same conclusions we reached using the `vif` command.

```
. fit bodyfat tricep thigh midarm
  Source |       SS       df       MS                  Number of obs =      20
---------+------------------------------                F(  3,    16) =   21.52
   Model |  396.984607     3  132.328202                Prob > F      =  0.0000
Residual |  98.4049068    16  6.15030667                R-squared     =  0.8014
---------+------------------------------                Adj R-squared =  0.7641
   Total |  495.389513    19  26.0731323                Root MSE      =    2.48
------------------------------------------------------------------------------
 bodyfat |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  tricep |   4.334085   3.015511      1.437   0.170      -2.058512    10.72668
   thigh |  -2.856842   2.582015     -1.106   0.285      -8.330468    2.616785
  midarm |  -2.186056   1.595499     -1.370   0.190      -5.568362     1.19625
   _cons |   117.0844   99.78238      1.173   0.258      -94.44474    328.6136
------------------------------------------------------------------------------

. colldiag
Proportion of variance associated with the decomposition
  Cond   |
 Number  |  tricep     thigh    midarm     _cons
---------+------------------------------------
       1 |  0.0000    0.0000    0.0000    0.0000
 677.372 |  0.9985    0.9996    0.9917    0.9990
 13.9048 |  0.0013    0.0000    0.0014    0.0004
 18.5657 |  0.0002    0.0003    0.0069    0.0006
```

Note the dominating dependency. We may choose to model `tricep` or `thigh` by the remaining variables.

```
. fit tricep thigh midarm
  Source |       SS       df       MS                  Number of obs =      20
---------+------------------------------                F(  2,    17) = 6016.66
   Model |  478.753112     2  239.376556                Prob > F      =  0.0000
Residual |  .676355525    17  .039785619                R-squared     =  0.9986
---------+------------------------------                Adj R-squared =  0.9984
   Total |  479.429468    19  25.2331299                Root MSE      =  .19946
------------------------------------------------------------------------------
  tricep |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
   thigh |   .8554801   .0087733     97.509   0.000       .8369701    .8739902
  midarm |   .5265439    .012592     41.816   0.000       .4999771    .5531107
   _cons |  -33.01306   .5459378    -60.470   0.000      -34.16489   -31.86123
------------------------------------------------------------------------------
```

Below we choose to eliminate `tricep` from the model based on the linear dependence with `thigh`.

```
. fit bodyfat thigh midarm
  Source |       SS       df       MS                  Number of obs =      20
---------+------------------------------                F(  2,    17) =   29.40
   Model |  384.279748     2  192.139874                Prob > F      =  0.0000
Residual |  111.109765    17  6.53586854                R-squared     =  0.7757
---------+------------------------------                Adj R-squared =  0.7493
   Total |  495.389513    19  26.0731323                Root MSE      =  2.5565
------------------------------------------------------------------------------
 bodyfat |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
   thigh |   .8508818   .1124482      7.567   0.000       .6136367    1.088127
  midarm |   .0960295   .1613927      0.595   0.560      -.2444792    .4365383
   _cons |  -25.99696    6.99732     -3.715   0.002      -40.76001   -11.2339
------------------------------------------------------------------------------

. colldiag
Proportion of variance associated with the decomposition
  Cond   |
 Number  |   thigh    midarm     _cons
---------+----------------------------
       1 |  0.0011    0.0018    0.0007
 15.4756 |  0.2735    0.7979    0.0198
 26.3842 |  0.7254    0.2003    0.9794
```

References

Belsley, D. A., E. Kuh, and R. E. Welsch. 1980. *Regression Diagnostics.* New York: John Wiley & Sons.

Neter, J., W. Wasserman, and M. H. Kutner. 1989. *Applied Linear Regression Models*. Homewood, IL: Irwin.

sg33 | Calculation of adjusted means and adjusted proportions

Joanne Garrett, University of North Carolina, FAX 919-966-2274

After fitting a multiple linear regression model or a logistic regression model, we often find it useful to calculate covariate adjusted means or proportions (probabilities) by different categories of a nominal independent variable. Although this is not particularly difficult to do in Stata by using the `_coef[`*varname*`]` construct and solving either the linear or logistic equation, this insert offers two utilities that make the calculations even simpler.

`adjmean` estimates a linear regression model (using `regress`) and calculates means and 95 percent confidence intervals for categories of a specified nominal independent variable. `adjprop` fits a logistic regression model (using `logistic`) and calculates proportions ("probabilities") and 95 percent confidence intervals. Both utilities adjust the estimates to the means of any covariates in the model. The means or proportions and confidence intervals are always shown. Optionally, the model parameter estimates and/or a plot may be printed. Dummy variables are automatically generated for the nominal variable and assigned the names `x1`, `x2`, etc. The highest category is the dummy variable excluded from the analysis (the reference category).

The syntax of `adjmean` (for multiple linear regression models) is

`adjmean` *yvar* [`if` *exp*] `,` `by(`*xvar*`)` [`adjust(`*covlist*`)` `model` `plot` *graph_options*]

where *yvar* is a continuous outcome variable, *xvar* is a nominal independent variable (any number of categories). The syntax of `adjprop` (for logistic regression models) is similar:

`adjprop` *yvar* [`if` *exp*] `,` `by(`*xvar*`)` [`adjust(`*covlist*`)` `model` `plot` *graph_options*]

where, in this case, *yvar* is a binary outcome variable that must be coded 0/1.

The options are the same for both commands:

`adjust(`*covlist*`)` specifies the list of additional covariates, that is, the variables for which the estimates are adjusted. If this option is not specified, unadjusted estimates are reported.

`by(`*xvar*`)` specifies the nominal variable that defines the categories for the estimated means or proportions. This option is required.

`model` displays the output from the `regress` or `logistic` command.

`plot` displays a graph of point estimates and confidence intervals.

Examples

To illustrate `adjmean` and `adjprop`, we have constructed a fictitious data set. We suppose we have collected data on obstetrical deliveries at six hospitals.

Example 1: The mean cost in dollars (with 95 percent confidence interval) of a delivery, adjusted for maternal age (`matage`), gestational age (`gestage`) at time of delivery, and whether a prior cesarean section (`priorc`) occurred, is estimated using data from each of the six hospitals. In addition to the means and confidence intervals, the results of the regression and a plot are requested.

```
. adjmean cost, by(hosp) adj(gestage matage priorc) model plot xlab(1,2,3,4,5,6)
                ylab(4000,6000,8000,10000,12000,14000) l2(Total Costs per Delivery)
  Source |       SS       df       MS                  Number of obs =    1550
---------+------------------------------               F(  8,  1541) =   30.37
   Model |  3.1620e+10     8  3.9525e+09               Prob > F      =  0.0000
Residual |  2.0058e+11  1541   130164038               R-squared     =  0.1362
---------+------------------------------               Adj R-squared =  0.1317
   Total |  2.3220e+11  1549   149905011               Root MSE      =   11409
------------------------------------------------------------------------------
    cost |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
      x1 |   1692.215   1087.979      1.555   0.120      -441.8608    3826.292
      x2 |   1203.107    1042.24      1.154   0.249      -841.2515    3247.466
      x3 |   6002.749   1062.853      5.648   0.000       3917.957     8087.54
      x4 |   3334.742   1074.987      3.102   0.002        1226.15    5443.335
      x5 |   5758.187   1210.601      4.756   0.000       3383.588    8132.786
 gestage |  -58.69252   50.19798     -1.169   0.242      -157.1561    39.77103
  matage |   563.6855   42.45172     13.278   0.000       480.4162    646.9547
  priorc |   1264.597   638.0788      1.982   0.048       13.00213    2516.191
   _cons |  -7930.245   2020.677     -3.925   0.000      -11893.81   -3966.677
------------------------------------------------------------------------------
```

(graph appears, see Figure 1)

```
Adjusted Means and 95% Confidence Intervals
  Outcome variable:      Total Delivery Cost -- cost
  Categorical variable:  Hospital -- hosp
  Covariates:   gestage matage priorc

       hosp      numobs    adjmean      lower      upper
1.   Hosp.A         264    7616.28   6238.008   8994.553
2.   Hosp.B         346   7127.172   5911.485   8342.859
3.   Hosp.C         296   11926.81    10623.8   13229.83
4.   Hosp.D         285   9258.808   7920.985   10596.63
5.   Hosp.E         169   11682.25   9957.575   13406.93
6.   Hosp.F         190   5924.065   4295.041   7553.089
```

The regression table was printed because the `model` option was requested. Beta estimates are displayed for five of the dummy variables, with `x1` representing the cost difference between Hospital F and Hospital A, `x2` the cost difference between Hospital F and Hospital B, etc. According to this regression, older mothers and mothers with prior cesarean sections tend to increase costs, whereas longer gestation time lowers costs slightly (but not significantly).

Following the regression table in this listing is the default summary table listing the outcome variable, categorical variable, any covariates, and the adjusted mean costs and 95 percent confidence intervals for each hospital. Because the `plot` option was specified, these means and confidence intervals are also displayed in a graph. As can be seen (from both the output and graph), Hospital C and Hospital E have significantly higher delivery costs than Hospitals A, B, and F, and moderately higher costs than Hospital D, after adjusting for the covariates.

Example 2: For this example, cesarean births (`csection`) is the outcome. A mother who receives a cesarean is coded as '1', while a mother with a vaginal birth is coded as '0'. A logistic regression model is estimated to calculate the proportion of cesarean births (or adjusted probability of having a cesarean) for each hospital, once again adjusted for maternal age (`matage`), gestational age (`gestage`) at time of delivery, and whether or not a prior cesarean section (`priorc`) occurred. In addition to the proportions and confidence intervals, the results of the logistic regression and a plot are requested.

```
. adjprop csection, by(hosp) adj(gestage matage priorc) model plot xlab(1,2,3,4,5,6)
                    ylab l2(Received Cesarean Section)
Logit Estimates                                         Number of obs =   1555
                                                        chi2(8)       = 146.63
                                                        Prob > chi2   = 0.0000
Log Likelihood = -661.36643                             Pseudo R2     = 0.0998
------------------------------------------------------------------------------
csection | Odds Ratio   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
      x1 |    1.32775   .4662503      0.807   0.420       .6671287    2.642547
      x2 |   1.532045   .4989285      1.310   0.190        .809222    2.900517
      x3 |    5.97663   1.844764      5.792   0.000       3.263788    10.94437
      x4 |   4.483571   1.418875      4.741   0.000       2.411304    8.336737
      x5 |   2.498061   .8669734      2.638   0.008       1.265279    4.931963
 gestage |   .9960092   .0119784     -0.333   0.740       .9728065    1.019765
  matage |   1.082634   .0116927      7.351   0.000       1.059957    1.105795
  priorc |   1.543302   .2309072      2.900   0.004       1.151052     2.06922
------------------------------------------------------------------------------
```

(graph appears, see Figure 2)

```
Adjusted Proportions and 95% Confidence Intervals
  Outcome variable:      cesarean Section -- csection
  Categorical variable:  Hospital -- hosp
  Covariates:   gestage matage priorc
        hosp      numobs    adjprop      lower      upper
1.   Hosp.A         265   .0905053    .061584   .1311111
2.   Hosp.B         347   .1029967   .0761353   .1379204
3.   Hosp.C         296   .3093608     .25779   .3661588
4.   Hosp.D         285   .2515154   .2022776   .3081095
5.   Hosp.E         170   .1576988     .11108   .2190612
6.   Hosp.F         192   .0697221   .0413566   .1152049
```

The logistic regression table, summary of variables, adjusted proportions (probabilities) and 95 percent confidence intervals, and plot are printed. As anticipated, hospitals with higher proportions of cesarean deliveries tend to have higher costs. The exception is Hospital E. Although they have a moderately low proportion of cesarean births (.158), their average cost for a delivery is one of the highest ($11,682). This discrepancy must be due to something other than maternal age, gestational age, or prior history of cesarean section.

Example 3: Calculate the mean costs for a cesarean section vs. a vaginal delivery, adjusted for maternal age, gestational age, and prior cesarean. Do not request the regression table or plot.

```
. adjmean cost, by(csection) adj(matage gestage priorc)
Adjusted Means and 95% Confidence Intervals
  Outcome variable:      Total Delivery Cost -- cost
  Categorical variable:  cesarean Section -- csection
  Covariates:   matage gestage priorc
   csection      numobs    adjmean      lower      upper
1.     0:no        1269   7635.516   7011.214   8259.818
2.    1:yes         281   14434.88   13092.56   15777.19
```

The average cost for a cesarean delivery is almost twice the cost of a vaginal birth, adjusted for the covariates.

These examples, although fictitious, should give a feel for the capabilities of the two utilities. Although `adjmean` and `adjprop` were designed to be used with nominal main effects, there is no reason why an interaction term could not be constructed and used to calculate either means or proportions. For instance, suppose one were interested in an interaction between two dichotomous variables, say education (less than high school, high school graduate) and gender (male, female). A new variable (`newvar`) could be generated that takes the value '1' for males with less than a high school education, '2' for females with less than a high school education, '3' for male high school graduates, and '4' for female high school graduates. The new variable would then be used in place of the main effects (education and gender) and the interaction term. For example,

```
. adjmean yvar, by(newvar) adj( covlist) ...
```

The results would print adjusted means for each of the four categories.

Figures

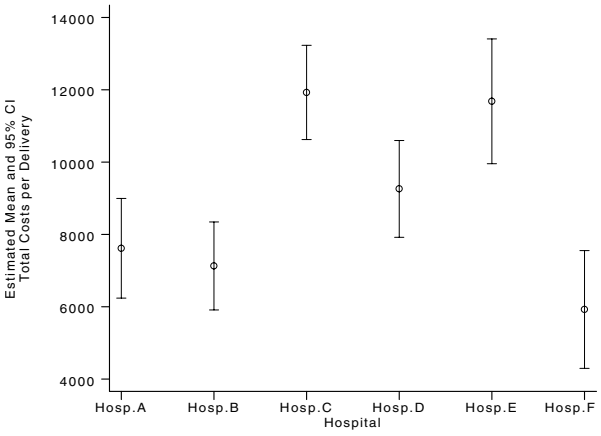

Figure 1

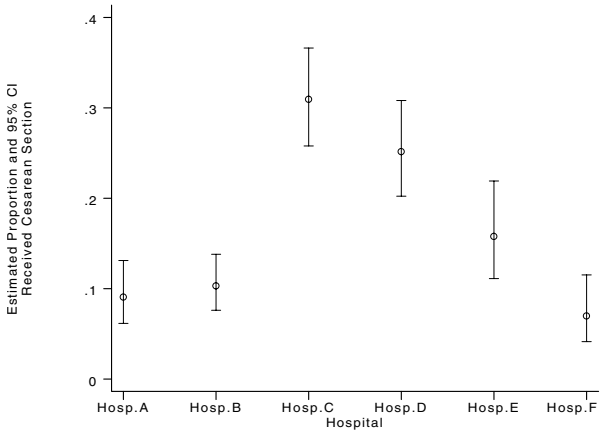

Figure 2

sg34	Jackknife estimation

William Gould, Stata Corporation, FAX 409-696-4601 EMAIL stata@stata.com

The syntax of `jknife` is

```
jknife clear
jknife cmd stata_cmd                              (required)
jknife stat # [ newvar = exp ]                    (required)
jknife n = exp
jknife [ query ]
jknife do [, replace level(#) ]                   (required)
```

`jknife` performs jackknife estimation. A problem is first defined using (at a minimum) `jknife cmd` and `jknife stat` and then executed using `jknife do`.

`jknife clear` undefines any previous problem.

`jknife cmd` defines the Stata command that is to be executed, a command which calculates one or more statistics and saves them so that they can be subsequently accessed. (Typically, Stata commands save results in `_result()` or in `S_#` macros; see Saved Results in the command's printed documentation.) `jknife` assumes that *stata_cmd* follows standard Stata syntax and, in particular, allows `if` *exp*.

`jknife stat` defines a statistic to be stored in a new variable. There can be one or more `jknife stat` commands for a particular problem. (`jknife stat` can also be used to edit definitions.)

`jknife n`, which is optional, states where *stata_cmd* saves the number of observations on which it based the calculated results. This is typically either `_result(1)` or the macro `S_1`; again, see Saved Results under the particular command of interest. If `jknife n` is not specified, `_N`, the number of observations in the data set, will be assumed, and missing values will not be handled correctly. You are strongly advised to define this parameter.

`jknife query`, or `jknife` by itself, lists the currently defined problem.

`jknife do` executes the problem.

Options

`replace` specifies that the variables defined by `jknife stat` may already exist and that their contents may be replaced.

`level(#)` specifies the significance level in percent for the confidence intervals; see [5u] level.

Remarks

While the jackknife—developed in the late 1940s and early 1950s—is of largely historical interest today, it is still useful for searching for overly influential observations. This feature is often forgotten. In any case, the jackknife is

1. an alternative, first-order unbiased estimator for a statistic;
2. a data-dependent way to calculate the standard error of the statistic, and so obtain significance levels and confidence intervals;
3. a way of producing measures called pseudovalues for each observation reflecting the observation's influence on the overall statistic.

The idea behind the simplest form of the jackknife—the one implemented here—is to calculate the statistic in question N times, each time omitting just one of the data set's observations. Write S for the statistic calculated on the overall sample and $S_{(j)}$ for the statistic calculated when the jth observation is removed. If the statistic in question were the mean, then

$$S = \frac{(N-1)S_{(j)} + s_j}{N}$$

where s_j is the value of the data in the jth observation. Solving for s_j, we obtain

$$s_j = NS - (N-1)S_{(j)}$$

These are the pseudovalues the jackknife calculates even though the statistic in question is not the mean. The jackknife estimate is $\overline{s}$, the average of the s_j's, and its estimate of the standard error of the statistic is the corresponding standard error of the mean (Tukey 1958).

These days, the jackknife estimate of variance has been largely replaced by bootstrapping, which is widely viewed as more efficient and robust. But the use of the jackknife pseudovalues as a way of detecting outliers is too often forgotten and is something the bootstrap is unable to provide. See Mosteller and Tukey (1977, 133–163) and Mooney and Duval (1993, 22–27) for more information.

Example: Jackknifed standard deviation

Mosteller and Tukey (1977, 139–140) request a 95% confidence interval for the standard deviation from which the eleven values

0.1, 0.1, 0.1, 0.4, 0.5, 1.0, 1.1, 1.3, 1.9, 1.9, 4.7

were observed. Stata's `summarize` command calculates the mean and standard deviation and, looking under Saved Results in [5s] summarize, we find the number of observations is saved in `_result(1)` and the variance (square of the standard deviation) in `_result(4)`. We have already entered the data into Stata.

```
. list
                x
  1.          .1
  2.          .1
 (output omitted)
 10.         1.9
 11.         4.7
. jknife clear
. jknife cmd summarize x
. jknife stat 1 sd = sqrt(_result(4))
. jknife n = _result(1)
. jknife do
 cmd:  summarize x
   n:  _result(1)
stat:  [1]  sd = sqrt(_result(4))
Variable |     Obs    Statistic    Std. Err.       [95% Conf. Interval]
---------+--------------------------------------------------------------
sd       |
 overall |      11     1.343469
  jknife |             1.489364     .624405        .0981028    2.880625
```

Interpreting the output, the standard deviation reported by '`summarize mpg`' is 1.34. The jackknife estimate is 1.49 with standard error 0.62. The 95% confidence interval for the standard deviation is .10 to 2.88.

In addition, jknife creates a new variable in our data—sd, the pseudovalues.

```
. list
             x         sd
  1.         .1   1.139978
  2.         .1   1.139978
  3.         .1   1.139978
  4.         .4   .8893153
  5.         .5   .8242676
  6.          1   .6324884
  7.        1.1   .6203195
  8.        1.3   .6218883
  9.        1.9   .8354196
 10.        1.9   .8354196
 11.        4.7   7.703949
```

The jackknife estimate is the average of sd, so sd contains the individual "values" of our statistic. We can see that the variance of the last observation is substantially larger than that for the others. The last observation is certainly an outlier, but whether that merely reflects the considerable information it contains or indicates that it should be excluded from analysis is a decision that must be based on the context of the problem. In this case, Mosteller and Tukey created the data by sampling from an exponential distribution, so the observation is quite informative.

Example: Jackknifed standard deviation 2

Let us repeat this example using the automobile data, obtaining the standard deviation of the standard deviation of mpg.

```
. use c:\stata\auto, clear
(1978 Automobile Data)
. jknife clear
. jknife cmd summ mpg
. jknife stat 1 sd = sqrt(_result(4))
. jknife n = _result(1)
. jknife do
 cmd:  summ mpg
   n:  _result(1)
stat:  [1]  sd = sqrt(_result(4))
Variable |     Obs    Statistic    Std. Err.       [95% Conf. Interval]
---------+--------------------------------------------------------------
sd       |
 overall |      74     5.785503
  jknife |             5.817374    .6072509        4.607125    7.027624
```

Now looking at sd more carefully,

```
. sum sd, detail
                            sd
-------------------------------------------------------------
      Percentiles      Smallest
 1%     2.870471       2.870471
 5%     2.870471       2.870471
10%     2.906255       2.870471       Obs                  74
25%     3.328489       2.870471       Sum of Wgt.          74
50%     3.948335                      Mean           5.817374
                        Largest       Std. Dev.       5.22377
75%     6.844418       17.34316
90%     9.597018        19.7617       Variance       27.28777
95%     17.34316        19.7617       Skewness        4.07202
99%     38.60905       38.60905       Kurtosis       23.37823
. list make mpg sd if sd>30
                  make       mpg         sd
 72.          VW Diesel        41   38.60905
```

In this case, the VW Diesel is the only diesel car in our data.

Other features: collecting multiple statistics

`jknife` is not limited to collecting one and only one statistic. For instance, using `summarize, detail` you wish to obtain the jackknife estimate of the standard deviation and skewness. `summarize, detail` saves the variance in `_result(4)` and the skewness in `_result(14)`, so you might type

```
. jknife clear
. jknife cmd summ mpg, detail
. jknife n = _result(1)
. jknife stat 1 sd = sqrt(_result(4))
. jknife stat 2 skew = _result(14)
. jknife do
```

Other features: editing

There is no required order to the `jknife` commands except that `jknife clear` be used first and `jknife do` last. Moreover, between the `clear` and `do` you can define and redefine the elements and you can type `jknife` by itself to review where you are.

```
. jknife
 cmd:  summ mpg mpg , detail
   n:  _result(1)
stat:  [1]  sd = sqrt(_result(4))
stat:  [2]  skew = _result(14)

. jknife stat 1 mean = _result(3)
. jknife
 cmd:  summ mpg mpg , detail
   n:  _result(1)
stat:  [1]  mean = _result(3)
stat:  [2]  skew = _result(14)

. jknife stat 2
. jknife
 cmd:  summ mpg mpg , detail
   n:  _result(1)
stat:  [1]  mean = _result(3)
```

Note that typing '`jknife stat 2`' deleted the second statistic.

Other features: accessing `S_` macros

Some Stata commands store results in the global `S_` macros. For instance, you wish to obtain an estimate of the standard deviation of Cronbach's alpha, the scale reliability coefficient. `alpha`—see [5s] alpha—stores α in the global macro `S_4`; it does not save the number of observations. You could type

```
. jknife clear
. jknife cmd alpha quest1-quest10, std
. jknife stat 1 alpha = S_4
. jknife do
```

Note that in the `jknife stat` command, you type `S_4`, not `$S_4`. `jknife` assumes that expressions starting with "`S_`" are macro references; do not type a dollar sign before them.

In some other problem you may need to refer to a global scalar that starts with `S_`, such as `S_1`. (Future versions of Stata are moving toward storing numerical results in scalars, not macros.) To define that the statistic *whatever* is obtained from scalar `S_1`, you type

```
. jknife stat 1 whatever = scalar(S_1)
```

A note on commands and programs that may be used with jknife

Most Stata commands and user-written programs can be used with `jknife`. The requirement is that the command follows standard Stata syntax and that it allows `if exp`. There is nothing more to say, but the following may interest programmers:

Assume you have defined

`jknife cmd` *stata_cmd*

`jknife`'s main loop repeatedly executes

stata_cmd `if _n~=`j

or

stata_cmd`, if _n~=`j

for $j = 1, \ldots,$ `_N`. Which form `jknife` chooses depends on the number of commas outside parentheses and brackets in *stata_cmd*.

For example, if *stata_cmd* is "`summarize mpg`", then `jknife` executes

`summarize mpg if _n~=`j

If *stata_cmd* is "`summarize mpg, detail`", then `jknife` executes

`summarize mpg, detail, if _n~=`j

Many users do not realize it, but commas do not merely set off options from the standard command syntax; they toggle back-and-forth between options and standard syntax.

Saved Results

`jknife` saves in the global `S_#` macros values corresponding to the last (which is typically the only) statistic reported:

`S_1`	the number of observations
`S_2`	the overall statistic
`S_3`	bootstrap estimate (mean of pseudovalues)
`S_4`	standard error of the mean
`S_5`	lower confidence bound
`S_6`	upper confidence bound

References

Mooney, C. Z. and R. D. Duval. 1993. *Bootstrapping: A Nonparametric Approach to Statistical Inference.* Newbury Park, CA: Sage Publications.

Mosteller, F. and J. W. Tukey. 1977. *Data Analysis and Regression.* Reading, MA: Addison–Wesley Publishing Company.

Tukey, J. W. 1958. Bias and confidence in not-quite large samples. Abstract in *Annals of Mathematical Statistics* 29: 614.

snp7.1	Natural cubic splines: Correction

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

I have discovered and corrected a minor bug in the programs described in *snp7* (Sasieni 1994). This bug was unlikely to affect results. Corrected versions of the programs are supplied on this month's distribution diskette.

Reference

Sasieni, P. 1994. snp7: Natural cubic splines. *Stata Technical Bulletin* 22: 19–22.

ssi5.4	Correction to error term in Ridders' method

Alan R. Riley, Stata Corporation, FAX 409-696-4601, EMAIL stata@stata.com

The `ridder` program described in *ssi5.2* and *ssi5.3* calculates $|x - \widehat{x}|$ incorrectly. This causes the `tol` value to be interpreted incorrectly—sometimes `ridder` will not converge even though $|x - \widehat{x}| < tol$. A corrected version appears on the STB-24 media.

References

McGuire, T. 1994a. ssi5.2: Equation solving by Ridders' method. *Stata Technical Bulletin* 17: 19–22.

——. 1994b. ssi5.3: Correction to Ridders' method. *Stata Technical Bulletin* 19: 28.

sts7.6	A library of time series programs for Stata (Update)

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

In *sts7*, a library of time series programs for Stata was introduced (Becketti 1994). That insert described an approach to time series analysis that builds on Stata's core commands and on its extensibility. The insert also cataloged the programs in the time series library.

As *sts7* promised, the time series library is updated in each issue of the STB. New programs and revisions are posted on the STB distribution diskette. If you use the time series library, you should copy each new version over your existing version. Type `help tsnew` to see a history of the changes in the library. Type `help ts` to see a catalog of all the programs in the library.

New features

The additions to this version of the time series library focus on the problem of time series forecasting. There are four new programs—`date2obs`, `filldate`, `projdate`, and `scenario`—that help in producing forecasts under user-chosen scenarios. Also, the dynamic forecasting program, `tspred`, has been upgraded and is documented here for the first time.

Time series forecasting

The programs described in this insert are designed to produce dynamic forecasts from a time series regression model. Suppose that you wish to produce forecasts for the simple time series model

$$y_t = \mu + \alpha y_{t-1} + \beta x_t + \epsilon_t$$

First you need an estimate of the model. `tsfit` and `tsreg` (described in *sts4*, Becketti 1993) can be used to calculate an estimate. Once the model is estimated, you must decide whether you want a static or a dynamic forecast of y_t. A static forecast is essentially a within-sample forecast. In each period, the actual value of y_{t-1} is used in computing the forecast. Stata's `predict` command computes static forecasts.

A dynamic forecast is more difficult to calculate. For this illustrative model, period 2 ($t = 2$) is the first period for which a forecast, either static or dynamic, can be calculated, because an initial value of y_{t-1} is needed to begin the calculations. This first forecast is

$$\widehat{y}_2 = \widehat{\mu} + \widehat{\alpha} y_1 + \widehat{\beta} x_2$$

The dynamic forecast for period 2 is identical to the static forecast, because the realized value of y_1 is used to compute the forecast.

The dynamic forecast for period 3 is

$$\widehat{y}_3 = \widehat{\mu} + \widehat{\alpha}\widehat{y}_2 + \widehat{\beta} x_3$$

The dynamic forecast for this period differs from the static forecast because $\widehat{y}_2$, not y_2, is used to compute the forecast. If the static forecast for period t is denoted by $\widetilde{y}_t$, the difference between the static and dynamic forecasts for this model is

$$\widetilde{y}_t - \widehat{y}_t = \widehat{\alpha}(y_{t-1} - \widehat{y}_{t-1}) = \widehat{\alpha}\widetilde{\epsilon}_{t-1}$$

In a more general time series regression, the difference can be written as

$$\widetilde{y}_t - \widehat{y}_t = \widehat{\alpha}(L)(y_{t-1} - \widehat{y}_{t-1}) = \widehat{\alpha}(L)\widetilde{\epsilon}_{t-1}$$

where $\alpha(L)$ is a p-th order polynomial in the lag operator L.

The difference between static and dynamic forecasts can be substantial. Using the realized value of lagged y's prevents the static forecasts from drifting too far from the realizations. Static forecasts can only be calculated after the fact, that is, when a forecast is no longer needed. Dynamic forecasts can drift very far from the realizations; one bad residual can throw the entire forecast profile out of whack. Because dynamic forecasts use only the information that would actually be available to the forecaster, they are closer to being an honest measure of the accuracy of a time series model. (In practice, the forecaster does not know the realized value of future x's, and the parameter estimates also may improve as more realizations are observed.)

In the ordinary dynamic forecast described above, there is a fixed terminal date of history but the forecast horizon grows over time. In the illustrative model, period 1 is the terminal date of history. Realizations of y after period 1 are unknown—they are part of the forecast or projection period. The forecast horizon grows each period, though. The dynamic forecast of y_t is a horizon $t-1$ forecast, that is, it is a forecast of the dependent variable $t-1$ periods after the terminal date of history.

There is an important class of forecasts that inverts this arrangement. These forecasts—called k-step ahead forecasts—hold the forecast horizon constant but allow the terminal date of history to change. k-step ahead forecasts are typically used to assess the reliability of a model. For instance, if we are interested in producing accurate forecasts of inflation over the coming year, we might estimate a model for the monthly Consumer Price Index (CPI). To see whether this model would work well in practice, we could calculate 12-step ahead forecasts to estimate the 12-month ahead forecasts of the CPI. These forecasts would produce a time series of the forecasts we would have produced if we had in fact used the estimated model over the observed sample. For actual forecasts, of course, we would use the ordinary dynamic forecasts.

Calculating dynamic forecasts with `tspred`

`tspred` is a generalization of Stata's `predict` command. `tspred` calculates dynamic and k-step ahead forecasts, in contrast to the static forecasts computed by `predict`. `tspred` can also compute dynamic simulations.

The syntax of `tspred` for calculating forecasts is

`tspred` *newvar* [`if` *exp*] [`in` *range*] [, `residual` `steps(`#`)`]

The syntax for calculating simulations is

`tspred` *newvar* [`if` *exp*] [`in` *range*] [, `error(`*varname*`)` `normal` `simulate` { `rmse(`#`)` | `variance(`#`)` }]

In the first syntax, `tspred` produces ordinary dynamic forecasts, or their residuals, by default. If the `steps(`k`)` option is specified, then `tspred` produces the 1-step ahead, 2-step ahead, ..., and k-step ahead forecasts. The variable name specified in the command is prefixed with the `P` operator, to signify a linear projection. For instance, the command

```
tspred x, steps(3)
```

creates the three variables `P.x`, `P2.x`, `P3.x`. This notation will be familiar to users of the `lag`, `dif`, and `growth` commands in the time series library.

In the second syntax, `tspred` produces dynamic simulations with normally distributed innovations, by default. If no variance or root mean squared error is specified, the estimated error variance from the most recent time series regression is used to scale the errors. The `error()` option can be used to supply a user-generated variable containing an arbitrary sequence of innovations. This option makes it possible to analyze the behavior of the model under a variety of distributions for the innovations.

`tspred` has been in the time series library for over a year. In its original form, though, `tspred` was very difficult to use. As a consequence, its availability was not highlighted, and it was classified as a Level B program in the library. Level A programs are fully documented, provide a reliable user interface, and meet all the requirements of Stata estimation commands. Level B programs are accurate but do not meet at least one of the requirements of a Level A program. Additions to Stata's programming commands in the last two releases made it possible to improve `tspred` and to raise it to its new status as a Level A program in the time series library.

Specifying forecast scenarios

`tspred` calculates dynamic forecasts and simulations of the dependent variable in a time series regression. It is the user's responsibility, however, to specify the forecast scenario, that is, the sequence of values for the explanatory variables in the model.

There are several approaches to scenario generation. When all the substantive variables in the model are jointly endogenous, the model can be specified as a vector autoregression, or VAR. While `tsfit` can be used to estimate the equations of a VAR, the time series library does not yet provide any tools for calculating VAR forecasts, impulse response functions, or variance decompositions. Tools for forecasting the variables in structural, or simultaneous equations models, are also unavailable at present. In the single equation context, the future values of the explanatory variables typically are specified either as the forecasts from an auxiliary model, such as a Box–Jenkins model, or, more commonly, as ad hoc scenarios. For example, in a model of housing demand, we might want to forecast the future path of new home sales in the event that mortgage interest rates increase three percentage points over the next twelve months.

A variety of ad hoc scenarios can be generated quickly and easily using the `scenario` command. Its syntax is

`scenario` *existing varname* `,` [`begin(`*date*`)`] `action(`*action*$_1$ [`,` *action*$_2$ [`,` *action*$_3$ ...]])

`amount(`*amount*$_1$ [`,` *amount*$_2$ [`,` *amount*$_3$ ...]]) `length(`*length*$_1$ [`,` *length*$_2$ [`,` *length*$_3$ ...]])

`scenario` specifies a program—a sequence of "actions"—to fill in the values of an existing variable in the projection period. Each action has associated amounts and lengths. For instance, the action "`jump`" specifies that the value of the variable is to jump immediately by the amount specified by the associated `amount` option and to remain at the new value for the number of observations specified by the associated `length` option. `scenario` makes it easy to generate complicated scenarios, such as "immediately increase the mortgage rate by two percentage points and hold it at its new value for six months, then let the mortgage rate decline a quarter point every month for a year, then hold it constant for the remainder of the projection period".

The available actions are

`flat`	hold the variable at its current value;
`grow`	increase in equal increments over the specified length for a cumulative change equal to the specified amount;
`incr`	increment the variable by a fixed amount each period;
`jump`	increment the variable immediately by a fixed amount, then hold it constant;
`pchange`	grow or decline at a constant;
`pgrow`	grow by a fixed amount over the specified length, but at a constant percentage rate of change; and
`set`	set the variable to an explicit value and hold it constant.

When the `flat` action is specified, an amount must be entered, but it will be ignored. Also a length of "`.`" indicates the remainder of the projection period.

The beginning of the projection period—that is, the first period for which a forecast will be calculated—can be specified either by the `begin()` option or by the `projdate` command. The beginning of the projection is specified as a date, in the same format used by the `datevars` command. For instance, to produce monthly forecasts beginning in April 1995, you could specify the `begin(1995 4)` option, or you could type

```
. projdate 1995 4
First projection date: April 1995
```

prior to using the `scenario` command. Note that this method of specifying the projection period differs from the approach used by `tspred` (and `predict`) where the projection period is set implicitly by the `if` and `in` qualifiers and by the pattern of missing values in the explanatory variables. In a subsequent section, we discuss some of the inconsistencies in the time series library in the specification of date ranges.

When the `set obs #` command is used to add observations for a projection period to an existing data set, all variables are set to missing in the projection period. One of the minor annoyances of scenario generation is remembering to fill in the values of the date variables during the projection period so forecasts can be labeled with dates when they are listed or graphed. The `filldate` command is a simple utility that fills in the current date variables, as defined by the `datevars` and `period` commands.

The syntax for `filldate` is

`filldate ,` [`begin(`*date*`) datevars(`*varnames*`) period(`*period*`)`]

The options can be used either to override current settings or supply values that have not been previously set.

Example

It is probably easier to understand the forecasting commands from an example rather than from their formal descriptions. In the following, artificial example, we want to estimate the model

$$y_t = \mu + \alpha y_{t-1} + \beta_1 \Delta x_{t-1} + \beta_2 \Delta x_{t-2} + \epsilon_t$$

using data from the period April 1993 through December 1994, and we want to calculate dynamic forecasts of y_t for the 12 months of 1995. These forecasts are calculated for a scenario where the value of x is assumed to grow by two units, in equal-sized steps, for the first six months of 1995, and then to remain constant at its new, higher value.

We begin by bringing the data into memory, setting the period and the date variables, and estimating the model.

```
. use model, clear
. describe
Contains data from model.dta
  Obs:    24 (max= 30460)
 Vars:     4 (max=    99)                     8 Feb 1995 16:46
Width:    12 (max=   200)
  1. year         int    %8.0g
  2. month        int    %8.0g       month
  3. y            float  %9.0g
  4. x            float  %9.0g
Sorted by:  year  month
. period 12
12 (monthly)
. datevars year month
. dif x
. tsfit y D.x, lags(1,2)
Monthly data:  April, 1993 to December, 1994    (21 obs)
  Source |       SS       df       MS                  Number of obs =      21
---------+------------------------------                F(  3,    17) =   33.74
   Model |  519.778579     3  173.259526                Prob > F      =  0.0000
Residual |   87.298904    17  5.13522965                R-squared     =  0.8562
---------+------------------------------                Adj R-squared =  0.8308
   Total |  607.077483    20  30.3538741                Root MSE      =  2.2661
------------------------------------------------------------------------------
       y |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
     L.y |   .7755407   .0922801      8.404   0.000       .5808467    .9702347
    LD.x |    4.51251   .6210512      7.266   0.000       3.202206    5.822813
   L2D.x |  -.3960156    .644075     -0.615   0.547      -1.754895    .9628639
   _cons |   4.219835   1.622043      2.602   0.019       .7976232    7.642047
------------------------------------------------------------------------------
```

Now that the model is estimated, we extend the data set to include a twelve month projection period. We set the first projection date to January 1995, the month following the final observation in the estimation period, and we fill in the date variables. We also create a Stata date variable and format it using Stata 4.0's new date format. This variable is used to display graphs of the variables in the model.

```
. local Np12 = _N + 12
. set obs `Np12´
obs was 24, now 36
. projdate 1995 1
First projection date: January 1995
. filldate
. generate long date = mdy(month,1,year)
. format date %dm_y
```

Now we are almost ready to compute the forecasts. We use the `scenario` command to generate the desired scenario. If this were a larger model, we would use the `scenario` command once for each explanatory variable in the model.

```
. scenario x, action(grow,flat) amount(2,0) length(6,6)
. graph x date, c(l) s(o) rescale xline(12768) xlabel(12054,12235,12419,12600,12784,12965,13118)
```

(graph appears, see Figure 1)

We discovered the date values used in the `xlabel()` option by trial and error. The `datelab` command described earlier in this issue (*dm26*) simplifies this process.

Prior to calculating the dynamic forecasts, we must regenerate the various lags and differences in the model, to propagate the effects of the `scenario` command.

```
. lag y
(note:  L.y replaced)
. dif x
(note:  D.x replaced)
. lag 2 D.x
(note:  LD.x replaced)
(note:  L2D.x replaced)
. tspred fit
. graph y fit date, c(ll) s(op) xline(12768) xlabel(12054,12235,12419,12600,12784,12965,13118)
```

(graph appears, see Figure 2)

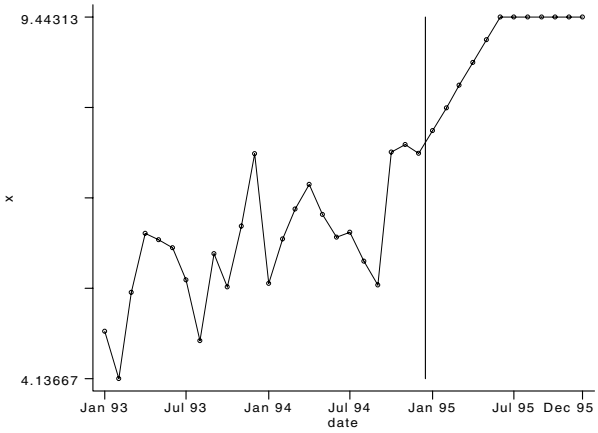

Figure 1

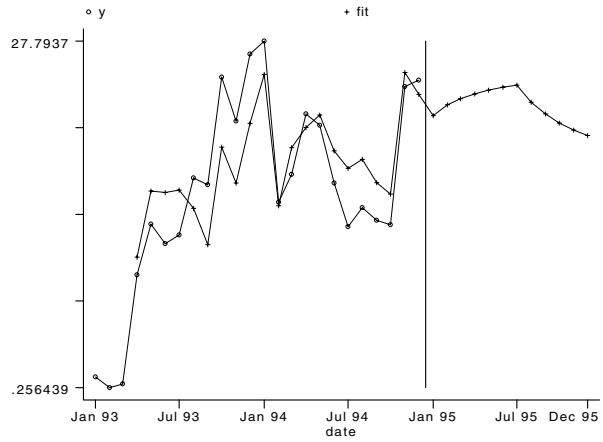

Figure 2

Miscellaneous new commands

The time series library contains a host of utility programs written to support the development of the library. Many of these programs are quite general and provide solutions to problems faced by many Stata programmers, even those who have no interest in time series analysis. These programs are usually added without mention, but, in this issue, we briefly note two utility programs. The first program, `date2obs`, was requested by a user of the time series library, and it may be useful to nonprogrammers.

`date2obs` returns in `S_1` the observation number corresponding to a particular date or a '0' if the date is not contained in the current data set. The syntax is

`date2obs` *date values*

The date values are specified according to the current `period` and `datevars` settings. For instance, in the data set used for the example above, the 15th observation contains information for March 1994. Thus

```
. list in 15
         year     month          y          x
 15.     1994     March    17.2034   6.625315
. date2obs 1994 3
. display "$S_1"
15
```

The second utility program is a very low-level routine that has been in the time series library for some time. It is described here to give an idea of the types of programming tools that can be found in the library. This utility program is called `_partset` and its syntax is

`_partset` *first-set-of-tokens second-set-of-tokens*

`_partset` partitions two sets (comma-enclosed lists) of tokens into three distinct sets: the set that appears only in the first list, the set common to both lists, and the set that appears only in the second list. The following example should be self-explanatory.

```
. _partset "a b c d e" "c e f g h"
. disp_s
S_1:        a b d
S_2:        c e
S_3:        f g h
```

`_partset` was written to solve a problem that arose in handling lists inside a Stata program. The time series library includes many low-level utility programs of this type. If you run into problems in writing your own Stata programs, you should browse in the time series library, and among the programs that accompany other STB inserts, to see if someone else has already developed a solution for your problem.

Inconsistencies in the time series library

The time series library evolved over a long period, largely in response to specific user requests. Certain conventions have turned out to be useful unifying devices, and these conventions have been implemented in most of the programs in the library. For instance, three options—`current()`, `lags()`, `static()`—are commonly used in the library to specify time series models. Adopting this syntax has made it easier to write a common set of utilities for parsing, estimating, and otherwise manipulating time series models.

The treatment of dates is not handled consistently in the time series library, and that inconsistency appears in the programs described in this insert. The convention promoted by the `period` and `datevars` commands is to store dates in multiple, "natural" variables. For instance, two variables—a year variable and a month variable—are used to identify observations in a monthly data set. Moreover, the "slowest-moving" variables are listed first when date variables are specified: for instance, we type '`datevars year month`', not '`datevars month year`'. Other routines, such as `tspred`, refer to dates implicitly, through the devices of the `if` and `in` command modifiers. With the introduction in Stata 4.0 of date formats, it is reasonable to consider specifying dates using Stata elapsed dates, rather than sets of natural variables. This approach would avoid the need to create a duplicate date variable to use in graphs.

Inconsistencies such as the treatment of dates reflect both the evolution of the tools available in Stata and the accumulation of user reactions to the time series library over time. In effect, the time series library is an experimental component of Stata. Features that work well are retained and eventually propagated throughout the library. Features that do not meet with broad acceptance tend to wither away.

The point of this discussion is to solicit your reactions, especially to the forecasting routines introduced in this issue. Your comments have a significant impact on the development of the library.

Reference

Becketti, S. 1993. sts4: A suite of programs for time series regression. *Stata Technical Bulletin* 15: 20–28.

——. 1994. sts7: A library of time series programs for Stata. *Stata Technical Bulletin* 17: 28–32.

STB categories and insert codes

Inserts in the STB are presently categorized as follows:

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

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

International Stata Distributors

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

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

Company:	Howching
Address:	11th Fl. 356 Fu-Shin N. Road Taipei, Taiwan, R.O.C.
Phone:	+886-2-505-0525
Fax:	+886-2-503-1680
Countries served:	Taiwan

Company:	Metrika Consulting
Address:	Ruddammsvagen 21 11421 Stockholm Sweden
Phone:	+46-708-163128
Fax:	+46-8-6122383
Countries served:	Baltic States, Denmark, Finland, Iceland, Norway, Sweden

Company:	Oasis Systems BV
Address:	Lekstraat 4 3433 ZB Nieuwegein The Netherlands
Phone:	+31 3402 66336
Fax:	+31 3402 65844
Countries served:	The Netherlands

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

Company:	Timberlake Consultants
Address:	47 Hartfield Crescent West Wickham Kent BR4 9DW, U.K
Phone:	+44 181 462 0495
Fax:	+44 181 462 0493
Countries served:	Eire, Portugal, U.K.