STATA TECHNICAL BULLETIN

January 1994

STB-17

A publication to promote communication among Stata users

Contents of this issue | page

an1.1 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.

an38 Stata distributor in France

Stata Corporation announces Ritme Informatique as its new distributor in France.

Ritme has specialized in distributing multi-platform scientific software in France for more than four years in the fields of mathematics, statistics, systems control and simulation, and data analysis and visualization. Ritme is also the official distributor of Mathematica, Spyglass, Scientific Word, and VisSim. In addition, Ritme provides a broad range of services to its customers, from technical support to specific developments and training. Ritme also publishes its own line of scientific software and books.

Please contact Ritme if you have any questions about Stata, or if you would like to be added to their mailing list and receive, on a regular basis, their scientific software catalog and Ritme Informatique newsletter (with product announcements, information on new releases, training sessions, etc.)

Please contact:

RITME INFORMATIQUE
34, Boulevard Haussmann
75009 PARIS
(33-1) 42.46.00.42 (voice)
(33-1) 42.46.00.33 (FAX)

an39 NSF funds workshops on exploratory data analysis for social scientists

Two one-week workshops on exploratory data analysis will be held this summer at San Francisco State University. These workshops are funded by the National Science Foundation's Faculty Enhancement Program and are designed to introduce social sciences faculty to exploratory data analysis on microcomputers. The workshops are intended for faculty who are familiar with statistical methods through multiple regression, who teach undergraduate statistics and data analysis courses, and who are familiar with the use of either DOS or Macintosh computers.

The workshops will use Stata to cover the major techniques of exploratory and robust data analysis. A partial list of topics includes: data smoothing, median polishing, rescaling and transforming data using the ladder of powers, five-number summaries, box-and-whisker plots, stem-and-leaf plots, robust statistics, analytical graphics, and regression diagnostics including graphical methods of regression diagnosis. The workshops will be taught by Professors J. Theodore Anagnoson of the Department of Political Science, California State University, Los Angeles; Richard E. DeLeon of the Department of Political Science at San Francisco State University; and Richard Serpe of California State University, San Marcos.

Each workshop participant will receive:

- hotel accommodations at the San Francisco State University Guest Center.
- all meals during the workshops.
- a copy of Stata, with manual.
- a copy of Student Stata and StataQuest, both developed for teaching students about statistics.
- copies of Lawrence Hamilton's books *Statistics With Stata* and *Modern Data Analysis* and Anagnoson and DeLeon's *Using StataQuest.*
- numerous data sets that the instructors have produced for use in instruction exercises and student research.
- numerous handouts and exercises.
- a bound copy of relevant readings.

Workshop participants are responsible for travel costs to and from the workshop.

The first workshop covers beginning and intermediate topics and will be held from July 6–13. The second workshop covers intermediate and advanced topics and will be held from August 3–8. Each workshop is limited to twenty participants. The deadline for applications is April 1, 1994. For more information and for applications, please contact:

Professor J. Theodore Anagnoson
Department of Political Science
California State University
Los Angeles, CA 90032-8226
(213) 343-2230 (voice)
(213) 343-6452 (FAX)
tanagno@atss.calstatela.edu (Internet)

an40 Italian translation of Principles of Biostatistics

An Italian translation of Marcello Pagano and Kimberlee Gauvreau's *Principles of Biostatistics* will be published next month. *Principles of Biostatistics* is written for students of the health sciences and serves as an introduction to the study of biostatistics. Pagano and Gauvreau, both of the Harvard School of Public Health, use data from published studies to illustrate biostatistical concepts, and they supply these data—both in raw form and as Stata `.dta` files—with the book. Volume 1 of the Stata 3.1 Reference Manual contains a detailed description of the contents of *Principles of Biostatistics* along with ordering information (see [0] biostat).

The Italian translation is titled *Fondamenti di Biostatistica* and is published by Liviana Medicina-Gruppo Idelson. The translation is edited by Italo Angelillo, D.D.S., M.P.H., associate professor of hygiene; Maria Pavia, M.D., M.P.H., assistant professor of hygiene; and Paolo Villari, M.D., M.P.H., assistant professor of hygiene. Angelillo and Pavia are with the Medical School of the University of Reggio Calabria. Villari is with the Medical School of the University "G. D'Annunzio" of Chieti.

For information on ordering *Fondamenti di Biostatistica*, contact:

Liviana Medicina-Gruppo Idelson
Via A. De Gasperi, 55—Napoli
(011) 39-81-552-4733 (voice)
(011) 39-81-552-8295 (FAX)

cc2 Running the Stata tutorials on a network

In a recent communication, Allan Reese of Hull University writes

> I just solved a long-standing mystery. We have always had to tell students to switch to the network disk as their default to run the tutorials. Just having the tutorial files in the default path or even appending didn't help. Poking around the directory now, I found `_setpath.tut` and a simple edit makes the tutorials fully visible. I obviously have never noticed that in the documentation, and I can't find a mention now in the installation notes.

Explanation

This communication highlights a gap in the Stata documentation. We thank Allan Reese for raising this issue and giving us the opportunity to fill this gap.

The Stata tutorials provide an efficient way to introduce new users to Stata. The tutorials are particularly helpful for those who do not have extensive experience with computers. This aspect makes the tutorials popular with instructors who wish to include a computer-based component in their courses. Instructors frequently send their students to a computer lab or college network to complete their course assignments. This can raise problems for students attempting to run the Stata tutorials, since each computer lab or network may install Stata differently.

The Stata tutorials are `do`-files, albeit with an extension of `.tut` rather than the traditional `.do`. Stata `do`-files are ordinary files containing one or more Stata commands. A `do`-file is similar to a DOS `.BAT` or a Unix shell script in that it can reduce a long and possibly tedious sequence of commands to a single, easily remembered command. To execute a Stata `do`-file, you type '`do` *name-of-*`do`*-file*' or '`run` *name-of-*`do`*-file*'. A related construct is the Stata program or `ado`-file—the mainstay of this bulletin. `ado`-files are invoked simply by typing the name of the `ado`-file.

Stata searches for `ado`-files in the path specified by the global macro `S_ADO`. In contrast, Stata looks only in the current directory for `do`-files. There is no path for `do`-files. If the desired file is in another directory, the full name of the file, including the path, must by typed—for example, '`do c:\myfile\thistut.tut`'.

This feature of `do`-files complicates the use of the Stata tutorials. The tutorials are interconnected. User-level tutorials call lower-level `do`-files to perform some tasks, and the user-level tutorials need some way to find the lower-level files. The `do`-file `_setpath.tut` is designed to solve this problem.

Suppose a user starts Stata from his or her personal directory and enters the tutorial system by typing

```
. do c:\stata\intro.tut
```

The `do`-file `intro.tut` is designed to execute other tutorial files. `intro.tut` runs `_setpath.tut` which "looks" for these tutorials in several likely locations. If it finds them, it tells `intro.tut` the path to use to locate the tutorials. Otherwise, it exits with an error message.

The listing below displays the contents of `_setpath.tut`.

```
version 2.1
mac def path
capture run ./nullfile.tut
if _rc == 0  exit

mac def path "c:\stata\"
capture run if _rc == 0  exit

mac def path "\stata\"
capture run if _rc == 0  exit

mac def path "/usr/local/stata/"
capture run if _rc == 0  exit

#delimit ;
di in red
"I cannot find the other tutorial files.  I have looked in the current" _n
"directory, in \stata (DOS) and in /usr/local/stata (Unix)."
_n "Is Stata installed correctly?" _n(2)
"In any case, I cannot run the tutorial." ;
#delimit cr
exit 601
```

`_setpath.tut` tries to execute `nullfile.tut` in a variety of locations. `nullfile.tut` is stored with the rest of the tutorial files. `nullfile.tut` doesn't do anything; its only command is `exit`. If `nullfile.tut` executes without an error, `_setpath.tut` knows it has found the location of the tutorial files. If `nullfile.tut` doesn't execute, `_setpath.tut` keeps looking. If `_setpath.tut` never finds `nullfile.tut`, it displays an error message.

It is straightforward to customize `_setpath.tut` for your network. Simply add the directory where you have stored the tutorial files to the list of locations searched by `_setpath.tut`.

crc34	Programming command: marking observations for inclusion

The syntaxes of `mark` and `markout` are

> `mark` *newmarkvar* [*weight*] [`if` *exp*] [`in` *range*]
>
> `markout` *markvar* [*varlist*]

`aweight`s, `fweight`s, `pweight`s, and `iweight`s are allowed.

Description

`mark` and `markout` are for use in Stata programs.

`mark` creates *newmarkvar* containing 0's and 1's. Observations meeting the `if`, `in`, and having nonzero weight will have *newmarkvar* set to 1; the remaining observations will be set to 0. Weights, if specified, are checked for sensibleness (not negative if not `iweight`s, integers if `fweight`s) and the appropriate error message issued if not.

`markout` resets 1's in a marking variable created by `mark` to contain 0 wherever any of the variables in *varlist* contain missing values. String variables are treated as if they contain missing values.

Remarks

`mark` and `markout` are fast commands for managing sample selection in Stata programs. Consider a Stata program that begins

```
program define myprog
        version 3.1
        local varlist "required existing"
        local if "optional"
        local in "optional"
        parse "`*´"
        (program continues)
```

Pretend this Stata program continues to make a statistical calculation based on the observations specified in the varlist that do not contain missing values (such as a linear regression). The program must identify the observations that it is to use. Moreover, since the user can specify `if` or `in`, these restrictions, too, must be taken into account. `mark` and `markout` make this easy:

```
        (myprog, continued from above)
        tempvar touse
        mark `touse´ `if´ `in´
        markout `touse´ `varlist´
        (program continues)
```

The `mark` command creates a (temporary) variable `` `touse´ `` (temporary because of the preceding `tempvar`; see [6a] macro) based on the `if` and `in`. If there is no `if` or `in`, `` `touse´ `` will contain 1 for every observation in the data. If '`if wgt>1000`' was specified by the user, only observations for which `wgt` is greater than 1,000 have `touse` set to 1; the remaining have `touse` set to zero.

The `markout` command updates the `` `touse´ `` marker created by `mark`. For observations for which `touse` is 1—observations that might be potentially used—the variables in the user-specified varlist are checked for missing values. If such an observation has any of the variables in the varlist equal to missing, its `touse` value is reset to 0.

Thus, the observations to be used by this command all have `` `touse´ `` set to 1; including "`` if `touse´ ``" at the end of statistical or data-management commands will restrict the command to operate on the appropriate sample.

Comments

1. `mark` and `markout` are internal commands of Stata 3.1; they were added at the last minute and so were not documented. Because they are internal, they are very fast.
2. By far the most common programming error—made by us and others—is to use different samples at different parts of the program. We strongly recommend that programmers identify the sample at the outset of the program. `mark` and `markout` make this easy and fast.

Examples

1. Write a program to do the same thing as `summarize`, except that it is to engage in casewise deletion—if an observation has a missing value in any of the variables, it is to be excluded from all the calculations.

```
program define cwsumm
        version 3.1
        local varlist "optional"
        local if "optional"
        local in "optional"
        local weight "aweight fweight"
        local options "Detail noFormat"
        parse "`*´"
        tempvar touse
        mark `touse´ `if´ `in´ [`weight´`exp´]
        markout `touse´ `varlist´
        summarize `varlist´ [`weight´`exp´] if `touse´, `detail´ `format´
end
```

2. Write `fsumm` for use after `fit` that reports the means of all the variables used in the estimation.

   Solution: [5s] fit (page 357, volume 2) reports that `fit` saves the original varlist (dependent and independent variables) in `$S_E_vl`, saves the original `if` and `in` (if any) in `$S_E_if` and `$S_E_in`, and the weight type and expression for the weight (if any) in `$S_E_wgt` and `$S_E_exp`. Finally, `fit` saves its own name in `$S_E_cmd`. Thus:

```
program define fsumm
        version 3.1
        if "$S_E_cmd"~="fit" { error 301 }
        if "`*´"~="" { error 198 }
        tempvar touse
        mark `touse´ $S_E_if $S_E_in [$S_E_wgt $S_E_exp]
        markout `touse´ $S_E_vl
        summarize $S_E_vl [$S_E_wgt $S_E_exp] if `touse´
end
```

   As an alternative, if we did write `cwsumm`, we could write

```
program define fsumm
        version 3.1
        if "$S_E_cmd"~="fit" { error 301 }
        if "`*´"~="" { error 198 }
        cwsumm $S_E_vl [$S_E_wgt $S_E_exp] $S_E_if $S_E_in
end
```

crc35 Warning about parity checking on DOS computers

A few unethical, no-name computer manufacturers have sold fast 486 computers with the memory parity checking turned off. They do this so they can use substandard, and cheaper, memory chips. When parity is turned on—as it should be—and the memory makes a "mistake," the computer is stopped and the words "`parity error`" are displayed. When parity is turned off, mistakes are not detected and the computer keeps right on running. If this should happen while you are running Stata, results of calculations can be incorrect and your data can even be permanently corrupted. Some 486s are designed so that parity can be turned off for testing purposes. The computer will still run, and special software can be executed that will determine the bad memory chip. Except for such diagnostic purposes, the parity should never be turned off.

If you have a computer of uncertain manufacture, you can verify that parity is turned on by entering the ROM setup when you power-up your computer. This is often done by holding down a certain key or keys during the boot process—typically the *Del* key. The startup screen usually says what the keys are. On other computers, there is a separate setup diskette you boot from. The parity setting is typically buried deep in the advanced setup menus. Make sure it is on or enabled. If not, enable it and exit the setup menus saving the changes. Now, if your computer has a hardware difficulty, it will stop before any damage is done.

dm16 Compact listing of a single variable

Patrick Royston, Royal Postgraduate Medical School, London, FAX (011)-44-81-740 3119
Peter Sasieni, Imperial Cancer Research Fund, London, FAX (011)-44-71-269 3429

The syntax of `lw` is

`lw` *varname* [`if` *exp*] [`in` *range*] [, `cols(`*#*`)` [`obsno`|`rownum`|`n`]
`linenum` `dense` `format(`%*fmt*`)` `width(`*#*`)` `nohead`]

`lw` is a variation on `list`—it can list only one variable but, rather than listing the data verbosely down the page, it lists it more compactly across the page.

`lw` can be helpful (especially with the `obsno` option) when searching for data errors since you can scan more data at once.

Options

`cols(`*#*`)` sets the total number of items to be displayed on a line. An item is a data value or an (optional) observation number. The default is to display as many columns as will fit (see `width()` below). The number of columns actually used is saved in `$S_1`.

`obsno` (synonyms `rownum` and `n`) displays the observation numbers as well as the data; the default is to display solely the data. If `obsno` is specified, the observation number followed by a fullstop (period) is listed in front of each data value. A variation especially useful for persons with monochrome monitors is available (see *Technical note* below). Note that the observation numbers count as items in `cols()`. If `obsno` is specified, half the items on each row will be observation numbers with the remaining half the corresponding data values. If # is odd and greater than 1, only $\# - 1$ items will be displayed per line.

`linenum` is a variation on `obsno`; it displays the observation number only once per line, listing the number at the start of the line. The number of data values per line will be one fewer than the number of items as specified by `cols()`.

`dense` lists the data more densely than otherwise by placing the columns closer together.

`format(`%*fmt*`)` specifies the display format for displaying the data values. Its use is not encouraged because `lw` engages in sophisticated logic to determine the format on its own (see *Methods and Formulas* below).

`width(`*#*`)` specifies the maximum number of characters per line; the default is 79. # must lie between 10 and 32000 inclusive.

`nohead` suppresses the header at the top identifying the variable and the number of observations listed.

Technical note

On color-text monitors, the optionally presented observation numbers are shown in green and the data in yellow. On monochrome-text monitors, the observation numbers are not so clearly distinguished. To remedy this, `lw` offers the alternative of surrounding the observation numbers with `[]` square brackets. Define the global macro `S_LW` by typing

```
. global S_LW  anything_except_period_or_blank
```

This need be done only once per session. (You might wish to include this in a do-file as part of your start-up procedure; see [1] start/stop.)

The default display mode can be reestablished by typing

```
. global S_LW
```

Example

```
. use \stata\auto, clear
(1978 Automobile Data)

. * Keeping the first 40 observations will suit our purpose and save paper:
. keep in 1/40
(34 observations deleted)
```

```
. lw mpg
mpg (listing 40 observations):
 22  17  22  20  15  18  26  20  16  19  14  14  21  29  16  22  22  24  19
 30  18  16  17  28  21  12  12  14  22  14  15  18  14  20  21  19  19  18
 19  24

. lw gratio
gratio (listing 40 observations):
 3.58  2.53  3.08  2.93  2.41  2.73  2.87  2.93  2.93  3.08  2.28  2.19  2.24
 2.93  2.56  2.73  2.73  2.73  2.56  3.54  2.47  2.47  2.94  3.15  3.08  2.47
 2.47  2.47  2.73  2.75  2.26  2.43  2.75  3.08  2.41  2.93  2.93  2.73  3.08
 2.73

. lw gratio, dense
gratio (listing 40 observations):
3.58 2.53 3.08 2.93 2.41 2.73 2.87 2.93 2.93 3.08 2.28 2.19 2.24 2.93 2.56
2.73 2.73 2.73 2.56 3.54 2.47 2.47 2.94 3.15 3.08 2.47 2.47 2.47 2.73 2.75
2.26 2.43 2.75 3.08 2.41 2.93 2.93 2.73 3.08 2.73

. lw mpg, obsno
mpg (listing 40 observations):
 1.  22     2.  17     3.  22     4.  20     5.  15     6.  18     7.  26
 8.  20     9.  16    10.  19    11.  14    12.  14    13.  21    14.  29
15.  16    16.  22    17.  22    18.  24    19.  19    20.  30    21.  18
22.  16    23.  17    24.  28    25.  21    26.  12    27.  12    28.  14
29.  22    30.  14    31.  15    32.  18    33.  14    34.  20    35.  21
36.  19    37.  19    38.  18    39.  19    40.  24

. lw mpg, linenum
mpg (listing 40 observations):
 1.  22  17  22  20  15  18  26  20  16  19  14  14  21  29  16  22  22
18.  24  19  30  18  16  17  28  21  12  12  14  22  14  15  18  14  20
35.  21  19  19  18  19  24
```

For users of monochrome monitors:

```
. global S_LW "mono"
. lw mpg, obsno
mpg (listing 40 observations):
[ 1] 22    [ 2] 17    [ 3] 22    [ 4] 20    [ 5] 15    [ 6] 18    [ 7] 26
[ 8] 20    [ 9] 16    [10] 19    [11] 14    [12] 14    [13] 21    [14] 29
[15] 16    [16] 22    [17] 22    [18] 24    [19] 19    [20] 30    [21] 18
[22] 16    [23] 17    [24] 28    [25] 21    [26] 12    [27] 12    [28] 14
[29] 22    [30] 14    [31] 15    [32] 18    [33] 14    [34] 20    [35] 21
[36] 19    [37] 19    [38] 18    [39] 19    [40] 24

. lw mpg, linenum
mpg (listing 40 observations):
[ 1] 22  17  22  20  15  18  26  20  16  19  14  14  21  29  16  22  22
[18] 24  19  30  18  16  17  28  21  12  12  14  22  14  15  18  14  20
[35] 21  19  19  18  19  24
```

Methods and Formulas

The only trick to `lw` has to do with determining the default display format. `lw` scans the data to be printed and determines the units of the variable. The units u are defined as the largest u such that every data value x_j can be expressed as $k_j \times 10^u$ for k_j an integer. If $u \geq 0$, then no decimal places are required; otherwise, $-u$ decimal places are required to accurately portray the data.

ip5 A temporary solution to a problem with temporary variable names

Craig S. Hakkio, Federal Reserve Bank of Kansas City, FAX 816-881-2199

Stata's `tempvar` command generates names that can be used for temporary variables used in `ado`-files. These names are guaranteed to be unique, and Stata automatically drops variables named by `tempvar` at the end of the program in which the name is created. `tempvar`, along with `tempname` and `tempfile`, make it easy to design elaborate, interconnected `ado` files without inadvertently reusing temporary names for variables, scalars, matrices, or files ([6a] macro).

The problem

I recently discovered a problem in combining operators with the temporary variable names generated by `tempvar`. This problem occurs frequently in time series applications.

The STB has featured a number of useful time series programs. Among these are utility programs for generating lags, leads, differences, and growth rates of existing variables (Becketti 1992, 1993). These commands employ an operator notation that is well-suited to naming variables that are produced by applying an operator to an existing variable. For example, the command

```
. lag 4 gnp
```

creates the four variables `L.gnp`, `L2.gnp`, `L3.gnp`, and `L4.gnp` which contain the first through fourth lags of the variable `gnp`.

The problem arises when `lag` or one of the other operator commands is used to create temporary variables. The problem is simplest to understand when one of these commands is applied to a temporary variable, but the problem arises in other situations as well. Consider the following sequence of commands:

```
. tempvar gnp
. generate `gnp´ = consume + invest + govt + exports - imports
. lag 4 `gnp´
```

The intent of these commands is, as before, to create four lagged variables, `L.`gnp´-L4.`gnp´`. In this case, though, there is no guarantee that this code will produce unique variable names for the lagged variables.

The problem lies in the technique used by `tempvar` to create temporary variable names. `tempvar` generates eight-character names of the form `__######`, that is, two underscores followed by six digits. The first time in a session that `tempvar` is called, it generates the name `__000000`. The second time it is called, it generates the name `__000001`, and so on. On the millionth call to `tempvar` in a single Stata session, `tempvar` generates the name `__000000` and begins the cycle again. (In a Monte Carlo study, I actually hit this limit once with disastrous results.)

Returning to the example above, imagine that the call to `tempvar` assigns the string `__000003` to the local macro `gnp`. Then the command `lag 4 `gnp´` will create the four variables `L.__0000`, `L2.__000`, `L3.__000`, and `L4.__000`. The crucial identifying character in the temporary name for `gnp` is the rightmost digit. However the two rightmost digits are "pushed" off the end of the variable name to make room for the `L.` prefix. Thus `L.__0000` could be the lag of any of the 100 temporary variables from `__000000` through `L.__0099`. Since temporary variables are created within programs, this type of name collision is highly likely to occur.

A temporary solution

There is no simple or obvious solution to this problem. The ideal solution would be for Stata to incorporate the operator notation in its variable name syntax. Thus if `X.` is a legal Stata operator and `x` is an existing variable, then Stata would treat the construction `X.x` as an existing variable without actually creating a separate variable to contain the values of `X.x`. Alternatively, Stata could allocate separate space in variable names to contain the operators. Then operators would not eliminate characters in the base name.

Both of these solutions involve drastic changes in Stata itself. These changes may never occur, and I need a workaround solution in the meantime. One unattractive solution is to return to the technique used to name temporary variables prior to the introduction of the `tempvar` command, that is, to choose unlikely names for temporary variables and hope that no name collisions occur. This technique is virtually guaranteed to fail.

I have devised an alternative solution to the problem with temporary variables. It is not as elegant and easy-to-use as Stata's `tempvar` command, and I regard it as a stopgap until Stata provides a permanent solution to the problem of combining operators with temporary variable names. However, this temporary solution works.

The `faketemp` command creates temporary variable names that can safely be combined with operators. The syntax is

faketemp [#]

The optional number following the command name specifies the number of temporary names to generate. The temporary names are stored in the global macros `S_1`, `S_2`, ...

`faketemp` creates legal, four-character variable names that do not exist in the current data set. It forms these names from upper and lower case letters chosen at random. For example,

```
. faketemp
. display "$S_1"
WpxI
```

Because the names generated by `faketemp` are only four characters long, up to four operator characters can be prefixed without losing any identifying information. Because the names generated by `faketemp` are completely random—in contrast to the sequential names generated by `tempvar`—a name collision is unlikely even when five or six operator characters are prefixed to a name.

One problem remains. `tempvar` automatically drops temporary variables at the end of the program that created them. You can approximate this feature by keeping a list of temporary variables created by your programs and explicitly dropping them at the end of the program. The following artificial example illustrates this approach.

```
. use example, clear
(National income accounts)
. faketemp
. display "$S_1"
WpxI
. local gnp "$S_1"
. local fake "`gnp´"
. generate `gnp´ = consume + invest + govt + netex
. label variable `gnp´ "Gross national product"
. lag 4 `gnp´
. local fake "`fake´ L.`gnp´-L4.`gnp´"
. describe
Contains data from example.dta
  Obs:   133 (max= 32766)                 National income accounts
 Vars:    12 (max=    99)
Width:    44 (max=   200)
  1. year         int    %8.0g            Year
  2. quarter      int    %8.0g    quarter  Quarter
  3. date         float  %9.0g            Date
  4. consume      float  %9.0g            Consumption
  5. invest       float  %9.0g            Investment
  6. govt         float  %9.0g            Government spending
  7. netex        float  %9.0g            Exports - imports
  8. WpxI         float  %9.0g            Gross national product
  9. L.WpxI       float  %9.0g            L.WpxI
 10. L2.WpxI      float  %9.0g            L2.WpxI
 11. L3.WpxI      float  %9.0g            L3.WpxI
 12. L4.WpxI      float  %9.0g            L4.WpxI
Sorted by:  year  quarter
Note:  Data has changed since last save
. drop `fake´
```

This approach will fail if the program is interrupted before the `drop` command is reached. Alternatively, you can use the `preserve` command before creating any temporary variables. This command takes a "snapshot" of the current data set and restores it at the end of a program's execution regardless of how the program is terminated. The `preserve` command saves the current data set in a temporary file, then `use`s the temporary data set at the end of the program. This approach can be very time-consuming in a program that is used frequently or in a repetitive program such as a Monte Carlo study.

While `faketemp` has limitations, it does solve the immediate problem of combining operators with temporary variable names. I hope that the next version of Stata improves the treatment of operators so that `faketemp` becomes obsolete.

References

Becketti, S. 1992. sts2: Using Stata for time series analysis. *Stata Technical Bulletin* 7: 18–26.

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

os5.1 Running Intercooled Stata under OS/2 2.1

William Gould, Stata Corporation, FAX 409-696-4601

Installing Intercooled Stata 3.0 or 3.1 under OS/2 2.1 is no different than under OS/2 2.0. You execute Stata under one of the IBM-supplied DOS windows, which may be full screen or not. The only problem is that Stata cannot determine how much memory to allocate to itself, so you must specify the `/k` startup option (see [5u] stata).

Invoke a DOS window. At some point, you should modify your `autoexec.bat` file and set the PATH to include the Stata directory just as you would were you running a real DOS computer; this way, Stata can be invoked from any directory. Right now, change to the directory in which Stata is installed. Type '`istata`'. Stata will respond with an allocation-failure message; this is the problem: Stata's automatic memory-allocation routines try to allocate too much memory and fail.

If you type '`istata /k1000`', however, Stata will come up. The `/k1000` overrides Stata's automatic determination and tells Stata to allocate 1 megabyte of memory.

`/k1000` works because IBM set the amount of DPMI memory to 2 megabytes. Stata uses this memory both for its code and data. Stata's code takes about 800K, so `/k1000` will work and, in fact, leave some memory free. (You can experiment to see how large you can make the data area: try `/k1100`, exit Stata if it works, and then keep trying larger numbers until you get the message "allocation failure." Use the largest number that worked. In practice, this does not make much difference.)

You will probably want more than 1 megabyte of memory for your data. To do this, you must increase the amount of DPMI memory allocated to the DOS window. This setting is found in the *DOS settings* menu under the window's configuration parameters. There are lots of settings here, but only the DPMI_MEMORY_LIMIT matters. As shipped by IBM, the limit is set to 2, meaning 2 megabytes. If you change the limit to 4, then 4 megabytes of DPMI memory will be allocated. Correspondingly, when you invoke Stata, you will have to specify the `/k3000` option, meaning 3 megabytes to Stata's data area (with the remaining used by Stata for its code).

You may make the DPMI_MEMORY_LIMIT as large as you want: 12 means 12 megabytes; 16, 16 megabytes; and so on. OS/2 provides virtual memory, so you can even push this limit beyond the physical amount of memory on your computer. If you do that, however, Stata will execute quite slowly. For performance reasons, we recommend setting the limit to 3 or 4 megabytes below the physical amount of memory on your computer. Just remember, whatever you change the limit to, you must give the corresponding `/k` option to Stata when you invoke it. If you change DPMI_MEMORY_LIMIT to 12, you invoke Stata by typing '`istata /k11000`'.

Performance under OS/2

OS/2 is an excellent performer. In our timings, Stata runs between 0.5 and 5 percent slower than it runs directly under DOS. In return for the slight speed degradation, OS/2 provides multitasking. Nevertheless, many users will think Stata under OS/2 is running slower than it really is. This is a perception issue.

OS/2 slows down the rate at which Stata can display characters on the screen considerably—approximately 15 percent. The more slowly appearing output fools most users into thinking everything is that much slower. That is not true. There is virtually no difference in the time it takes Stata to make statistical calculations and, in fact, that is where most cpu time is spent. (We also use Unix workstations with slow output times to the screen even when compared to OS/2 and yet, in fact, these computers are much faster than the computers on which we run DOS and OS/2.)

One occasion where OS/2 is slow is drawing Stata graphs in a DOS window. This is because OS/2 must monitor the graph, pixel by pixel, and clip it at the borders of the window. Performance is still not bad, but if this bothers you, use the fullscreen version of DOS, not the DOS window. This is why IBM supplied two window types for DOS: the fullscreen version does not require OS/2 monitor output for the borders around the window and so runs faster. Again we emphasize, this is only an output-to-the-screen issue: the time to perform statistical calculations is the same no matter which type of DOS window you use.

sg20 Point biserial correlation

John A. Anderson, Pennsylvania Transportation Institute, The Pennsylvania State University
FAX 717-948-6031, EMAIL jaa5@psuvm.psu.edu

`pbis` calculates a point biserial correlation coefficient and tests if the coefficient is significant from zero. The syntax of `pbis` is

`pbis` *bvar cvar* [`if` *exp*] [`in` *range*]

where *bvar* is a binary variable and *cvar* is a continuous variable.

Point biserial correlation is the preferred statistical procedure for calculating a simple correlation coefficient between a continuous variable and a dichotomous variable. Point biserial correlations are commonly used for comparing a “pass/fail” type variable with test scores. It is also appropriate to use point biserial correlations in other situations. A sociologist or economist, for example, may wish to calculate simple correlations using individuals from two different geographical regions and compare them on some continuous measure of socioeconomic status or attitude.

Point biserial correlation is a true Pearson product-moment correlation and therefore when using `pbis` the assumptions underlying Pearson’s r apply. Similarly, a point biserial correlation coefficient ($r_{\rm pbi}$) is interpreted identically to Pearson’s r. However, $r_{\rm pbi}$ is more appropriate when one of the variables is dichotomous.

When using `pbis`, *bvar* must be in the form of a zero/one dummy variable. The equation `pbis` uses for calculating a point biserial correlation coefficient is

$$r_{\rm pbi} = \frac{\overline{X}_p - \overline{X}}{S_X}\sqrt{\frac{p}{q}}$$

In this formula S_X=the standard deviation of *cvar* (the continuous variable); X_p=a value of *cvar* when *bvar*=1; X=any value of *cvar*; *p*=the proportion of *bvar* equal to one; and *q*=the proportion of *bvar* equal to zero.

`pbis` tests the $r_{\rm pbi}$ for significance from zero by calculating a t-ratio using the following formula:

$$t = r_{\rm pbi}\sqrt{\frac{N-2}{1-r_{\rm pbi}^2}}$$

In this formula N=the total number of observations. The degrees of freedom (df) used for a two-tailed test are ${\rm df} = N - 2$.

When `pbis` is the more appropriate choice, substituting correlation as an alternative may lead to false conclusions. The following is an example demonstrating a situation when Pearson’s r results in an inflated coefficient and leads the researcher to a false conclusion.

Example

In this example I look at the correlation between two geographical locations regarding some measure of individuals’ attitudes. The first variable, `location`, was coded such that zero represents location A and one represents location B. The second variable, `scale`, consists of measures of individuals’ attitudes

```
. use pbisdemo, clear

. describe

Contains data from pbisdemo.dta
  Obs:    11 (max= 32766)
 Vars:     2 (max=    99)
Width:     3 (max=   200)
  1. location     byte   %8.0g             Geographical location
  2. scale        int    %8.0g             Attitudinal scale
Sorted by:  scale

. list

      location      scale
  1.        0          8
  2.        0         11
  3.        0         20
  4.        1         29
  5.        0         30
  6.         0         33
  7.         1         35
  8.         1         39
  9.         0         42
 10.         1         44
 11.         1         54
. pbis location scale
(obs= 11)
Np= 5  p= 0.45
Nq= 6  q= 0.55
------------------+------------------+------------------+------------------+
Coef.= 0.5767          t= 2.1177        P>|t| = 0.0633        df=      9
```

The results displayed after employing `pbis` include Np, Nq, p, and q (the number of observations and respective ratios determined by the dichotomous variable). With df=9 the critical value of t for a two-tailed test at $\alpha = .05$ is 2.262. The t produced by `pbis` is not large enough to reject the null hypothesis that $\rho = 0$. Had we used the `correlate` command to calculate a Pearson's r, however, we might have falsely rejected the null hypothesis. As shown below r is inflated compared to r_{pbi}.

```
. correlate location scale
(obs=11)

        | location    scale
--------+------------------
location|   1.0000
   scale|   0.6048   1.0000
```

Using $r = 0.6048$ reveals an inflated t-ratio where t is defined by the equation:

$$t = r\sqrt{\frac{N-2}{1-r^2}}$$

The result, $t = 2.278$, is just large enough to reject the null hypothesis. However, in this case the researcher will have committed a Type-I error.

Saved Results

`pbis` produces the following system macros:

S_1	r_{pbs}
S_2	t-ratio
S_3	total number of observations
S_4	Np or number of observations for *bvar*==1
S_5	Nq or number of observations for *bvar*==0
S_6	mean of *cvar*
S_7	variance of *cvar*

sg21 Equivalency testing

Richard Goldstein, Qualitas Inc., Brighton MA, EMAIL goldst@harvarda.bitnet

When one fails to reject a null hypothesis that two means (or proportions) are equal, what is one's substantive conclusion? Technically, one cannot conclude that the means are equal, only that one cannot reject the hypothesis that they are equal. Traditionally, one would examine the power of the test and if it was relatively high, one would act as though the means were equal.

However, over the last twenty years or so, pushed by the Food and Drug Administration, a number of statisticians have devised tests of equivalence which can be contrasted with the usual test of difference. Note that in the usual test, here called a difference test, the null hypothesis is that there is no difference and the alternative hypothesis is that the means are different. In an equivalence test, the null hypothesis is that the means are different by at least X (where X is chosen to be substantively, or clinically, important), and the alternative hypothesis is that the means, if different, are different by less than X. Here, then, a rejection of the null hypothesis means we can conclude that the means are equivalent (within whatever band of equivalence is substantively important to us).

The FDA, and much of the literature, discusses this test in the context of crossover trials rather than simple tests of means or of proportions; however, the translation into t-test terms, for example, is relatively simple and much more useful to me. Thus, I present only the test comparing means and the test comparing proportions; the crossover factor is not implemented, but certainly one can obtain the same results using these ado-files.

Two types of equivalence tests are presented here, one for two means and one for two proportions. There are also immediate versions of both tests. The Westlake (1976, 1981, 1988) versions of both tests are presented and the Hauck-Anderson (Anderson and Hauck 1983, Hauck and Anderson 1984) version of the t-test is also presented. The H-A test is more powerful than the Westlake procedure; however, this is at least in part due to the anti-conservatism of its p-values, while the Westlake version p-values can be slightly conservative (Anderson and Hauck 1983, Frick 1987); further, the Westlake version also provides confidence intervals that may be helpful.

The Westlake confidence intervals (CIs) can be used instead of the p-values: if the CI is entirely contained within the equivalence interval (EI) (just a function of `delta`—see below), then one can reject the null hypothesis of difference of at least size `delta`. Two different types of CIs and EI are shown: if one is comparing one value to a benchmark, the CI and EI are intervals around the value being tested (the benchmark for the EI); if one is comparing two proportions or two means, then the CI and EI are around the differences between the two values (see examples below). In all cases the standard difference test results, and confidence intervals, are also presented.

Results were checked against those in Westlake's articles, those in Hauck and Anderson, and those in Rogers, Howard and Vesey (1993); all results matched.

Rogers, et al., argue that it can be informative to compare the results of the difference and equivalency tests. There are four possible combinations and their interpretations are shown in the cells of the following table:

	Equivalence test	
Difference test	reject null	do not reject null
reject null	difference larger than standard null, but not as large as `delta` (maybe too much power)	different
do not reject null	equivalent	insufficient evidence (too little power)

At least as a first cut, I believe this makes sense. Note that to reject the null hypothesis for the Westlake version of the equivalence test, one must reject the null for each of the two one-sided tests (or reject for the test with the larger p-value).

Equivalency testing for means

The `equim` command can be invoked in three different ways:

```
equim varname [= exp] [if exp] [in exp] [, delta(str) sdelta(str) random by(varname) ]
```

or

```
equimi n1 mean1 sd1 n2 mean2 sd2 [, delta(str) sdelta(str) random ]
```

or

```
equimi n1 mean1 sd1 benchmean [, delta(str) sdelta(str) ]
```

The null hypothesis is that the means differ by at least `delta` (`delta`) or by `delta` in one direction and `sdelta` (`sdelta`) in the other direction. For the third syntax, the null is that the mean differs from the benchmark mean by at least `delta` (or by `delta` in one direction and `sdelta` in the other direction).

The first set of information entered (*n1*, *mean1*, and *sd1*) should be the values for the control, baseline, or reference group, except in the third syntax where they are for the test group.

If `delta` is not entered, then a default of `.2*`*mean1* (or `.2*`*benchmean*) is used as both `delta` and `sdelta`. Delta can be entered either as a real number or as a percent; if you enter it as a percent, then you *must* include the percent symbol (`%`). Enter the percent with no decimal place unless you want a fraction of a percent; that is, 10 percent is entered as "`10%`", not as "`.1%`".

If `random` is entered, then `delta` is treated as an *estimated* amount and the variance is increased (that is, the confidence intervals are widened). If `random` is not entered—the default—then `delta` is treated as fixed and known. If `delta` is chosen to be a percentage, you would normally *not* use this option.

Instead of entering a second variable name, you can enter "`by(`*varname*`)`" as with the `ttest` command (this option is of course not available for the immediate version of the command). Do not enter both a second variable name and a "`by()`" or you will return to the Stata prompt with an error message.

The relevant t-test and confidence intervals are also presented. Results are presented for both the normal distribution and the t-distribution for the test of equivalence; the literature tends to use the normal distribution probability and confidence interval, but I prefer the t. Here the confidence interval is for the difference in the means; if both ends of the CI are within the `delta-sdelta` equivalence interval (range), then you conclude that the means are equivalent.

Finally, the p-value for the Hauck–Anderson test is also presented.

Example 1

The examples below follow the `ttest` examples on pages 230–231 of Volume 3 of the Version 3.1 *Stata Reference Manual* and use the `census.dta` data file supplied by Stata on disk.

```
. use census
(1980 Census data by state)

. equim medage if region==1 | region==4, by(region)
Westlake version of test of Equivalence w/delta fixed and known:
   test1:    13.953   normal p-value: 0.0000   t p-value: 0.0000
   test2:    -5.004   normal p-value: 0.0000   t p-value: 0.0001
  Confidence Interval: Lower Limit     1.865     Upper Limit     4.033
t-Confidence Interval: Lower Limit     1.574     Upper Limit     4.323
Equivalence Interval (-/+ (s)delta)   -6.247                     6.247

t-test

  Variable |      Obs        Mean   Std. Dev.
  ---------+---------------------------------
         x |        9    31.23333    1.023474
         y |       13    28.28462    1.775221
  ---------+---------------------------------
  combined |       22    29.49091    2.098051

           Ho:  mean(x) = mean(y)  (assuming equal variances)
                     t = 4.47 with 20 d.f.
               Pr > |t| = 0.0002

confidence interval: Lower Limit     2.991     Upper Limit     4.323

H-A p-value:  0.9993

. equimi 20 20 5 32 15 4

Westlake version of test of Equivalence w/delta fixed and known:
   test1:     7.165   normal p-value: 0.0000   t p-value: 0.0000
   test2:     0.796   normal p-value: 0.2130   t p-value: 0.4297
  Confidence Interval: Lower Limit     2.934     Upper Limit     7.066
t-Confidence Interval: Lower Limit     2.477     Upper Limit     7.523
Equivalence Interval (-/+ (s)delta)   -4.000                     4.000

t-test

  Variable |      Obs        Mean   Std. Dev.
  ---------+---------------------------------
         x |       20          20           5
         y |       32          15           4
  ---------+---------------------------------
  combined |       52    16.92308    5.007235

           Ho:  mean(x) = mean(y)  (assuming equal variances)
                     t = 3.98 with 50 d.f.
               Pr > |t| = 0.0002

confidence interval: Lower Limit     5.079     Upper Limit     7.523

H-A p-value:  0.5661
```

Equivalency testing for proportions

The three syntaxes for testing proportions are

`equip` *varname* [`=` *exp*] [`if` *exp*] [`in` *exp*] [, `delta(`*str*`)` `sdelta(`*str*`)` `random` `by(`*varname*`)`]

or

`equipi` *p1 n1 sd1 p2 n2 sd2* [, `delta(`*str*`)` `sdelta(`*str*`)`]

or

`equipi` *p1 n1 pbase* [, `delta(`*str*`)` `sdelta(`*str*`)`]

The null hypothesis is that the proportions differ by at least `delta` (or by `delta` in the negative direction and `sdelta` in the positive direction). That is, `delta` is used for *p1*−`delta` while `sdelta` is used for *p1*+`sdelta`.

If entering two variables, the first set of information entered (*p1*, *n1*, and *sd1*) should be the values for the control, baseline, or reference group.

If `delta` is not entered, then a default of `.2*`*p1* (or `.2*`*pbase*) is used as both `delta` and `sdelta`; if the expression in `=` *exp* is a number, then the default `delta` is 20 percent of this number. Delta can be entered as either a real number or as a percent; if you enter it as a percent, then you *must* include the percent symbol (`%`). Enter the percent with no decimal place unless you want a fraction of a percent; that is, 10 percent is entered as "10%", not as ".1%".

Instead of entering a second variable name (on the right side of the equals sign in `=` *exp*), you can enter a number against which the proportion of the first variable is to be tested, or you can enter "`by(`*varname*`)`" as with the `ttest` command (this option is, of course, not available for the immediate version of the command). Do not enter both `=` *exp* and `by()` or you will return to the Stata prompt with an error message. In this case, the *p*-values for the one-sided exact binomial tests are reported as well as Westlake's traditional tests.

The relevant binomial test and confidence interval are also presented. The confidence interval is for the difference in the proportions if two variables or two full proportions (see above); if both ends of the CI are within the `delta-sdelta` equivalence interval (range), then you conclude that the proportions are equivalent. Both exact and traditional CIs are presented for the case of comparing a proportion to a base proportion; make sure you compare exact to exact or traditional to traditional; do not compare exact to traditional.

Results were again checked against those in Rogers, et al. and all results matched for the equivalence tests.

Example 2

The following examples use `auto.dta` which is supplied on the Stata disk.

```
. use auto
(1978 Automobile Data)
. equip foreign=.3
Westlake version of test of Equivalence:
   test1:     1.078   p-value: 0.1404
   test2:    -1.180   p-value: 0.1190
  Confidence Interval: Lower Limit   -0.09010     Upper Limit     0.08469
  Exact Westlake CI: Lower Limit      0.21063     Upper Limit     0.39663
Equivalence Interval (-/+ (s)delta):  0.24000                     0.36000
Exact p-value from binomial test for single proportion:  1.00000
Exact CI: Lower Limit                 0.19655     Upper Limit     0.41464
Traditional CI: Lower Limit          -0.10684     Upper Limit     0.10144
```

Although the actual proportion for `foreign` is .297, we do not reject the null hypothesis because of low power; this is shown by tripling the size of the data set via Stata's `expand` command:

```
. expand 3
(148 observations created)
. equip foreign=.3
Westlake version of test of Equivalence:
   test1:     1.868   p-value: 0.0309
   test2:    -2.044   p-value: 0.0205
  Confidence Interval: Lower Limit   -0.05316     Upper Limit     0.04776
  Exact Westlake CI: Lower Limit      0.24685     Upper Limit     0.35185
Equivalence Interval (-/+ (s)delta):  0.24000                     0.36000
Exact p-value from binomial test for single proportion:  1.00000
Exact CI: Lower Limit                 0.23807     Upper Limit     0.36215
Traditional CI: Lower Limit          -0.06283     Upper Limit     0.05742
```

Note that if we want to compare proportions, we must ensure that we have a number between 0 and 1—this program does not make this check for you, but will give results that are all missing value indicators (dots) if your “proportion” is illegitimate:

```
. use auto, clear
(1978 Automobile Data)

. equip foreign=rep78 if rep78==3 | rep78==4
Westlake version of test of Equivalence:
   test1:         .   p-value:      .
   test2:         .   p-value:      .
  Confidence Interval: Lower Limit          .     Upper Limit          .
Equivalence Interval (-/+ (s)delta): -0.05000                    0.05000

standard test for comparing proportions:
    z =         .     p-value:       .
confidence interval: Lower Limit            .     Upper Limit          .
```

This problem is solved by recoding the categories to 0/1 as compared to the 3/4 used above:

```
. gen byte repa=0 if rep78==3
(44 missing values generated)

. replace repa=1 if rep78==4
(18 real changes made)

. equip foreign=repa
Westlake version of test of Equivalence:
   test1:    -0.208   p-value: 0.5823
   test2:    -1.563   p-value: 0.0591
  Confidence Interval: Lower Limit   -0.22209     Upper Limit    0.06669
Equivalence Interval (-/+ (s)delta): -0.05946                    0.05946

standard test for comparing proportions:
    z =    -0.888     p-value:  0.1874
confidence interval: Lower Limit     -0.24976     Upper Limit    0.09435

. equip foreign=repa, d(.2)
Westlake version of test of Equivalence:
   test1:     1.393   p-value: 0.0818
   test2:    -3.163   p-value: 0.0008
  Confidence Interval: Lower Limit   -0.22209     Upper Limit    0.06669
Equivalence Interval (-/+ (s)delta): -0.20000                    0.20000

standard test for comparing proportions:
    z =    -0.888     p-value:  0.1874
confidence interval: Lower Limit     -0.24976     Upper Limit    0.09435

. equip foreign=repa, d(.3)
Westlake version of test of Equivalence:
   test1:     2.532   p-value: 0.0057
   test2:    -4.303   p-value: 0.0000
  Confidence Interval: Lower Limit   -0.22209     Upper Limit    0.06669
Equivalence Interval (-/+ (s)delta): -0.30000                    0.30000

standard test for comparing proportions:
    z =    -0.888     p-value:  0.1874
confidence interval: Lower Limit     -0.24976     Upper Limit    0.09435
```

Also, note the different results from the use of different values for `delta`.

References

Anderson, S. and W. W. Hauck. 1983. A new procedure for testing equivalence in comparative bioavailability and other clinical trials. *Communications in Statistics—Theory and Methodology* A12: 2663–2692.

Food and Drug Administration, Office of Generic Drugs, Division of Bioequivalence. undated. Statistical procedures for bioequivalence studies using a standard two-treatment crossover design. photocopy; footnote states that document “is an informal communication under 21 CFS 10.90(b)(9) that represents the best judgment of the Division of Bioequivalence and the Office at this time.”

Frick, H. 1987. On level and power of Anderson and Hauck’s procedure for testing equivalence in comparative bioavailability. *Communications in Statistics—Theory and Methods* A16: 2771–2778.

Hauck, W. W. and S. Anderson. 1984. A new statistical procedure for testing equivalence in two-group comparative bioavailability trials. *Journal of Pharmacokinetics and Biopharmaceutics* 12: 83–91.

Rogers, J. L., K. I. Howard, and J. T. Vessey. 1993. Using significance tests to evaluate equivalence between two experimental groups. *Psychological Bulletin* 113: 553–565.

Westlake, W. J. 1976. Symmetrical confidence intervals for bioequivalence trials. *Biometrics* 32: 741–744.

——. 1981. Response. *Biometrics* 37: 591–593.

——. 1988. Bioavailability and bioequivalence of pharmaceutical formulations. In *Biopharmaceutical Statistics for Drug Development*, ed. K. E. Peace, 329–352. New York: Marcel Dekker.

sqc1 Estimating process capability indices with Stata

Sutaip L. C. Saw and Teck Wong Soon, National University of Singapore, FAX (011)-65-775-2646

Process capability indices are dimensionless summary measures used in quality control to provide information on the extent to which a process can produce items which conform to specifications. Kane (1986) discusses a number of such indices. Two indices in common use are

$$C_p = \frac{\mathrm{USL} - \mathrm{LSL}}{6\sigma} \quad \text{and} \quad C_{pk} = \min(\mathrm{CPL}, \mathrm{CPU})$$

where

$$\mathrm{CPL} = \frac{\mu - \mathrm{LSL}}{3\sigma} \quad \text{and} \quad \mathrm{CPU} = \frac{\mathrm{USL} - \mu}{3\sigma}$$

are single specification limit capability indices. In the above formulas, LSL and USL denote lower and upper specification limits on a quality characteristic whose mean and variance are μ and σ^2, respectively. The statistic C_p compares the natural tolerance (6σ) of the process with the allowable spread (USL−LSL). The statistic C_{pk}, on the other hand, compares the distance between the process mean and the nearest specification limit with half the natural tolerance. Large values of these indices are desirable, since they signify conformance to specifications and, hence, a capable process.

Point estimation

`pciest` calculates estimates of C_p and C_{pk} given user-supplied estimates of μ and σ together with one or both of the specification limits. The syntax to calculate C_p and C_{pk} is

`pciest` *mean sd*`, f(`*#*`) s(`*#*`)`

where *mean* and *sd* are estimates of μ and σ, respectively. `f(#)` and `s(#)` are the first (lower) and second (upper) specification limits.

The syntax to calculate CPL or CPU is

`pciest` *mean sd*`, f(`*#*`) t(`{`l` | `u`}`)`

The single specification limit is indicated by `f(`*#*`)`. The `t()` option indicates whether the lower (`t(l)`) or upper (`t(u)`) specification limit is indicated by `f(`*#*`)`.

Example

In this example, we use a subset of the data found on page 259 of DeVor, Chang, and Sutherland (1992). This data set consists of coded measurements of the inside diameter of machined cylinder bores. The data set contains 35 samples with 5 measurements per sample. The lower and upper specification limits are 195 and 203, respectively.

We begin by reading the measurements into Stata and estimating the process mean and standard deviation.

```
. infile x using cylinder.dat
(155 observations read)

. summarize x

Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+-----------------------------------------------------
       x |     155    199.9484   2.902738        194        207
```

To calculate C_p and C_{pk}, we issue the command

```
. pciest 199.95 2.90, f(195) s(203)

        LSL         =  195.0000       USL         =  203.0000

        Cp          =    0.4598       Cpk         =    0.3506

        (LSL+USL)/2 =  199.0000       (USL-LSL)/6 =    1.3333

        Mean        =  199.9500       Std Dev     =    2.9000

        p           =    0.1904       Yield (%)   =   80.9616
```

`pciest` reports the specification limits, the midpoint of the specification range, the allowable spread divided by six, estimates of C_p, C_{pk}, the mean, standard deviation, fraction nonconforming , and percent yield for a process.

To calculate CPU, we give the command

```
. pciest 199.95 2.90, f(203) t(u)

        USL         =  203.0000

        CPU         =    0.3506

        Mean        =  199.9500       Std Dev     =    2.9000

        p           =    0.1465       Yield (%)   =   85.3537
```

Similarly, to calculate CPL, we give the command

```
. pciest 199.95 2.90, f(195) t(l)

        LSL         =  195.0000

        CPL         =    0.5690

        Mean        =  199.9500       Std Dev     =    2.9000

        p           =    0.0439       Yield (%)   =   95.6079
```

Concluding remarks

In order for the process capability indices calculated by `pciest` to be meaningful, the process being examined must be under control, and the quality characteristic of interest must be normally distributed. In particular, the estimates of the yield and of the fraction nonconforming are based on the normality assumption. Stata's `xchart` and `rchart` commands can be used to check for process control, and the `swilk` command can be used to test for normality.

We have included a tutorial file, `pca.tut`, on the STB diskette that demonstrates the use of `pciest` in combination with these other Stata commands. `pca.tut` is written in a style similar to the tutorials that come with Stata. To run this tutorial, type

```
. do pca.tut
```

References

DeVor, R. E., T. H. Chang, and J. W. Sutherland. 1992. *Statistical Quality Control and Design.* New York: Macmillan.

Kane, V. E. 1986. Process capability indices. *Journal of Quality Technology* 18: 41–52.

ssi5.2 | Equation solving by Ridders' method

Tim McGuire, Stata Corporation, FAX 409-696-4601

Gould in *sii5* presented an equation solver `bisect` based on the bisection method. Clearly, with enough effort, we can always locate a solution to any desired accuracy with this method as long as the function is continuous. But compared with other methods, the bisection method converges rather slowly. (Mathematicians say it converges *linearly*.)

Ridders' method is one which retains most of the advantages of the bisection method—bounding of the solution, robustness—with considerably faster (*superlinear*) convergence.

`ridder` is an extension of `bisect` that uses the method of Ridders to find solutions to equations. It is also extended to bound not only the function value, but also the solution value, and to detect nonconvergence.

The syntax of `ridder` is

$$\texttt{ridder}\ \textit{fcnmask}\ \texttt{=}\ \textit{exp}_y\ \texttt{from}\ \textit{exp}_0\ \texttt{to}\ \textit{exp}_1\ \big[\texttt{tol}\ \textit{exp}_t\big]\ \big[\texttt{ltol}\ \textit{exp}_l\big]$$

where exp_y, exp_0, exp_1, exp_t, and exp_l are expressions (but are typically specified as numbers) and where *fcnmask* is either

(1) an expression containing `X`
(2) *progname* [*args*] `X` [*args*] `returns` { `exp` | `macro` } *name*

If one of `tol` or `ltol` is omitted, `ridder` ignores that constraint. `tol 1e-6` is assumed if neither `tol` nor `ltol` is specified.

Description

`ridder` finds the value of x such that $f(x) = exp_y$ or, more precisely

$$|x - \widehat{x}| < exp_t$$

and

$$|f(x) - exp_y| < \epsilon$$

where $\widehat{x}$ is the exact solution and $\epsilon = exp_l \max(|exp_y|, 1)$. The search is carried out over the range $exp_0 \le x \le exp_1$. The function $f()$ may be specified on the command line (first syntax for *fcnmask*) or as a user-written program (second syntax).

Example 1

Find the value of χ^2 with 2 degrees of freedom that is just significant at the 5% level.

Stata does not provide the inverse χ^2 function that could directly answer this question, but it does provide a `chiprob(`d`,`x`)` function that returns the reverse cumulative for χ^2 value x with d degrees of freedom. If you type '`display chiprob(2,3)`', Stata will respond with `.22313017`, meaning a χ^2 of 3 with 2 degrees of freedom is significant at the 22% level. '`display chiprob(2,8)`' results in `.01831564`, meaning the 1.8% level. Therefore, if we wish to find x such that `chiprob(2,`x`)` $= .05$ we can look for x in the range 3 to 8 (since we know that the function in question is continuous.)

```
. ridder chiprob(2,X)=.05 from 3 to 8
    Find  chiprob(2,X)=f() == c=.05, |X_error|<1.000e-06, |f()-c|<.
                  lower                  upper               weighted
iteration         bound     f()-c        bound     f()-c     midpoint     f()-c
-----------------------------------------------------------------------------
       0.             3  .1731302            8 -.0316844     5.962029  .0007413
       1.      5.962029  .0007413            8 -.0316844     5.992071 -.0000152
       2.      5.962029  .0007413     5.992071 -.0000152     5.991465  6.70e-11
       3.      5.991465  6.70e-11     5.992071 -.0000152     5.991465  4.16e-17
       4.      5.991465  4.16e-17     5.992071 -.0000152     5.991465 -6.94e-17
       5.      5.991465  4.16e-17     5.991465 -6.94e-17     5.991465 -6.94e-17
                                                           |X_error|<= 1.78e-15
```

The answer is 5.991465, shown in the last row in the column labeled "`weighted midpoint`".

The -6.94e–17 printed to the right of the solution means that `chiprob`$(2, 5.991465) = .05 - 6.94 \times 10^{-17}$. `ridder` stopped iterating when the difference between the true answer and the computed answer fell below 10^{-6}. The actual known bound is reported under the last line of the table, that is, $|x - \widehat{x}| < 1.78 \times 10^{-15}$. If we would be satisfied with an answer yielding a result accurate to 10^{-3}, we would type

```
. ridder chiprob(2,X)=.05 from 3 to 8 tol 1e-3
       Find  chiprob(2,X)=f() == c=.05, |X_error|<.001, |f()-c|<.
                  lower                  upper               weighted
iteration         bound     f()-c        bound     f()-c     midpoint     f()-c
-----------------------------------------------------------------------------
       0.             3  .1731302            8 -.0316844     5.962029  .0007413
       1.      5.962029  .0007413            8 -.0316844     5.992071 -.0000152
       2.      5.962029  .0007413     5.992071 -.0000152     5.991465  6.70e-11
       3.      5.991465  6.70e-11     5.992071 -.0000152     5.991465  4.16e-17
                                                           |X_error|<= .0006067
```

Similarly, if we wanted a more accurate answer, we could specify a smaller tolerance.

In the event we wish to bound the value of $f(x)$ rather than x, the `ltol` keyword is provided for that purpose. For example, if we want $|\texttt{chiprob}(2, x) - .05| < 1 \times 10^{-5}$, we would type

```
. ridder chiprob(2,X)=.05 from 3 to 8 ltol 1e-5
      Find  chiprob(2,X)=f() == c=.05, |X_error|<., |f()-c|<.00001
                 lower                   upper                 weighted
iteration        bound     f()-c         bound     f()-c      midpoint     f()-c
-----------------------------------------------------------------------------
      0.             3  .1731302             8 -.0316844      5.962029  .0007413
      1.      5.962029  .0007413             8 -.0316844      5.992071 -.0000152
      2.      5.962029  .0007413      5.992071 -.0000152      5.991465  6.70e-11
                                                            |X_error|<= .0294355
```

Finally, if we wish to bound the errors on both x and $f(x)$, we may specify `tol` and `ltol` on the same command line.

Example 2

As with `bisect`, we may write a program to calculate the function. An example program computing the same value as above is

```
program define mychi
        global S_1 = chiprob(2,`1')
end
```

The `` `1' `` in our program means substitute the first argument here. Typing '`mychi 3`', for instance, will store the evaluation of `chiprob(2,3)` in the global macro `S_1`. This time, to obtain the solution, we type

```
. ridder mychi X returns macro S_1 =.05 from 3 to 8
  (output omitted)
```

The output we see will be the same as in our first example. For additional programming examples, see the article by Gould in *sii5*.

Saved Results

`ridder` saves the solution in the global macro `S_1`.

Methods and Formulas

`ridder` is a variation of the bisection methods. Recall that the bisection method proceeds by evaluating the function at the midpoint of each successive interval. In Ridders' method, an additional step is performed which, in effect, factors out the function e^Q which transforms the residual function into a straight line. In other words, if x_1 and x_2 are the endpoints of the bracketing interval, and $x_m = (x_1 + x_2)/2$ is the midpoint of the interval, the equation

$$f(x_1) - 2f(x_m)e^Q + f(x_2)e^{2Q} = 0$$

is solved using the quadratic formula to give

$$e^Q = \frac{f(x_m) + \operatorname{sign}[f(x_2)]\sqrt{f(x_m)^2 - f(x_1)f(x_2)}}{f(x_2)}$$

Next, a false position technique is applied to the points $(x_1, f(x_1))$, $\bigl(x_m, f(x_m)e^Q\bigr)$, and $\bigl(x_2, f(x_2)e^{2Q}\bigr)$ to find a "weighted" midpoint:

$$w = x_m + (x_m - x_1)\frac{\operatorname{sign}[f(x_1) - f(x_2)]\, f(x_m)}{\sqrt{f(x_m)^2 - f(x_1)f(x_2)}}$$

w is the point at which the linearized function intersects the x axis. (We transform the equation $F(x) = c$ into $f(x) = F(x) - c = 0$ so this is valid.) Since w is guaranteed to lie in the interval (x_1, x_2), the method brackets the solution. As in the bisection method, a test is made to determine which side of w the solution lies on, and one of $\bigl(x_1, f(x_1)\bigr)$ or $\bigl(x_2, f(x_2)\bigr)$ is updated with $\bigl(w, f(w)\bigr)$.

To analyze the convergence rate of a solution-finding algorithm, we examine the change in the known bounds on the solution. Let Δx_k be the size of the bracketing interval after the k^{th} iteration. The bisection method obviously has the relationship

$$\Delta x_{k+1} = 0.5\Delta x_k$$

Any method that converges according to the formula

$$\Delta x_{k+1} = c\Delta x_k, \qquad 0 < c < 1$$

is said to converge *linearly*. Methods that converge as a higher power

$$\Delta x_{k+1} = c(\Delta x_k)^p, \qquad 0 < c < 1, \qquad p > 1$$

are said to converge *superlinearly*. If $p = 2$, the method is said to be *quadratic*.

For successive applications of the formula for w above, Ridders' method is quadratic. In actuality, since the application of the formula requires two function evaluations $\big(f(x_m)$ and $f(w)\big)$, the method actually has an order of $\sqrt{2} = 1.414\ldots$.

For a further discussion of Ridders' method, see Press et al. (1992, 358–359).

References

Gould, W. 1993. Equation solving by bisection. *Stata Technical Bulletin* 16: 20–23.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. 2nd ed. Cambridge: Cambridge University Press.

sts5 Detrending with the Hodrick–Prescott filter

Timothy J. Schmidt, Federal Reserve Bank of Kansas City, 816-881-2307

`hpfilter` smooths a time series using the Hodrick–Prescott detrending procedure.

The syntax of `hpfilter` is

`hpfilter` *varname* [`if` *exp*] [`in` *range*] `, lambda(`*#*`) suffix(`*newvar*`)`

`hpfilter` smooths the series *varname* and stores the smoothed values in the new series `H.`*newvar*.

Options

`lambda(`*#*`)` specifies the value for λ, the smoothing parameter. The default is 1600, the value suggested by Prescott (1986) for quarterly data. Higher values of `lambda` remove more fluctuation from the data.

`suffix(`*newvar*`)` allows the user to specify a portion (the suffix) of the name for the series that contains the new smoothed values; the series name would be H.*newvar*. If this option is not used, the name for the new series would be H.*varname* where *varname* is the time series to be smoothed.

Discussion

Researchers who work with time series data know that the fluctuations evident in the series can be the result of seasonal, cyclical, random or trend factors. For research purposes it is often desirable to separate a series' underlying trend from its other components. The process of isolating the trend component from a time series is known as "smoothing" the series.

All smoothing procedures involve transforming a time series with a class of operators that filter out undesirable elements ("noise") while extracting the trend. Many researchers have developed their own class of operators, known as "filters," to accomplish this task. One such filter was devised by a team of two economists—Robert Hodrick and Edward Prescott (1981). The Hodrick–Prescott filter uses a method of constrained optimization to fit the trend path to the time series. In effect, the trend is the curve that minimizes the sum of squared deviations from the time series subject to the constraint that the sum of the squared second differences not be too large (Prescott 1986). The *Methods and Formulas* section below describes the mathematical form of the optimization problem.

Given the wide assortment of other available filters, why should one use the Hodrick–Prescott method? The Hodrick–Prescott filter offers several distinct features that may recommend it in certain situations. First, unlike some filters, the Hodrick–Prescott technique does not require a stationary time series. Second, the Hodrick–Prescott filter treats the observations at both ends of the sample differently from the rest of the observations (see the conversion matrix in the *Methods and Formulas* section). Third, the Hodrick–Prescott technique allows the user to influence the extent of smoothing through the input of λ, the smoothing parameter.

The user's choice of λ influences the degree to which noise is removed from the time series. Higher values of λ result in the removal of more noise. Since time series with shorter frequencies tend to exhibit more noise, one will generally want to use a higher value of λ for monthly data, for instance, than for annual data. Prescott (1986) suggests a value of 1600 for λ when smoothing quarterly data.

Example

I will illustrate the use of the Hodrick–Prescott filter on a data set that contains monthly observations on the federal funds interest rate from January 1982 to August 1993.

```
. use fedfunds
(fed funds rate, 1982:1-1993:8)
. describe
Contains data from fedfunds.dta
  Obs:   140 (max=  5119)               fed funds rate, 1982:1-1993:8
 Vars:     4 (max=    99)
Width:    16 (max=   200)
  1. month        float  %9.0g
  2. year         float  %9.0g
  3. date         float  %9.0g
  4. ff           float  %9.0g             federal funds rate, percent
Sorted by:  year  month
. summarize
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+-----------------------------------------------------
   month |     140    6.385714   3.440315          1         12
    year |     140    1987.343   3.383777       1982       1993
    date |     140    87.79167   3.379883         82   93.58334
      ff |     140    7.581583   2.561527     2.8586    14.9914
```

Figure 1 graphs the federal funds rate and shows the distinct fluctuations evident from month to month.

The Hodrick–Prescott filter is intended to be used on data expressed in logs, so the time series must first be transformed.

```
. generate ln_ff = log(ff)
```

Now the log of the federal funds rate is smoothed using the Hodrick–Prescott filter. One can investigate the filter's sensitivity to the smoothing parameter by generating a couple versions of the smoothed fed funds rate, each with a different value of λ. I will choose values of 800 and 3200 for λ. A value of 800 has been used by some researchers in smoothing annual data (Hakkio, Rush, and Schmidt 1993), and a value of 3200 might be reasonable for monthly data since Prescott (1986) suggests a value of 1600 for quarterly data.

```
. hpfilter ln_ff, lambda(800) suffix(ff_8)
. hpfilter ln_ff, lambda(3200) suffix(ff_32)
. describe
Contains data from fedfunds.dta
  Obs:   140 (max=  5098)               fed funds rate, 1982:1-1993:8
 Vars:     7 (max=    99)
Width:    28 (max=   200)
  1. month        float  %9.0g
  2. year         float  %9.0g
  3. date         float  %9.0g
  4. ff           float  %9.0g             federal funds rate, percent
  5. ln_ff        float  %9.0g
  6. H.ff_8       float  %9.0g
  7. H.ff_32      float  %9.0g
Sorted by:  year  month
Note:  Data has changed since last save
```

Notice that the variable names for the new smoothed series begin with "H." Finally, the scale of the original series is restored by exponentiating the smoothed series.

```
. replace H.ff_8 = exp(H.ff_8)
(140 real changes made)
. replace H.ff_32 = exp(H.ff_32)
(140 real changes made)
```

Figure 2 below depicts the original federal funds rate series and the smoothed federal funds rate series. Note that the series became more smoothed as the value of λ increased.

Methods and Formulas

To obtain the smoothed series S_t, from time series Y_t, the Hodrick–Prescott method solves the following constrained optimization problem:

$$\min \sum_{t=1}^{T}(Y_t - S_t)^2 + \lambda \sum_{t=2}^{T-1}[(S_{t+1} - S_t) - (S_t - S_{t-1})]^2$$

It can be shown that the general solution to this problem is the product of a symmetric, band-diagonal matrix and the unsmoothed time series. Therefore, the fastest and most efficient way to apply the Hodrick–Prescott filter to a time series is to employ a little matrix algebra, a task to which the new version 3.1 of Stata is well suited. `hpfilter` uses Stata's new matrix programming features to construct the conversion matrix, adapting it to the situation at hand by incorporating the user's value of λ.

The conversion matrix is

$$\begin{pmatrix}
(1+\lambda) & -2\lambda & \lambda & & & & & & \\
-2\lambda & (1+5\lambda) & -4\lambda & \lambda & & & & & \\
\lambda & -4\lambda & (1+6\lambda) & -4\lambda & \lambda & & & & \\
 & \lambda & -4\lambda & (1+6\lambda) & -4\lambda & \lambda & & & \\
 & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\
 & & & \lambda & -4\lambda & (1+6\lambda) & -4\lambda & \lambda & \\
 & & & & \lambda & -4\lambda & (1+6\lambda) & -4\lambda & \lambda \\
 & & & & & \lambda & -4\lambda & (1+5\lambda) & -2\lambda \\
 & & & & & & \lambda & -2\lambda & (1+\lambda)
\end{pmatrix}$$

References

Hakkio, C. S., M. Rush, and T. J. Schmidt. 1993. The marginal income tax rate schedule from 1930 to 1990. Research Working Paper, Federal Reserve Bank of Kansas City.

Hodrick, R. J. and E. C. Prescott. 1981. Postwar U.S. business cycles: an empirical investigation. Discussion paper 451, Northwestern University.

Prescott, E. C. 1986. Theory ahead of business cycle measurement. Research Department Staff Report 102, Federal Reserve Bank of Minneapolis.

Figures

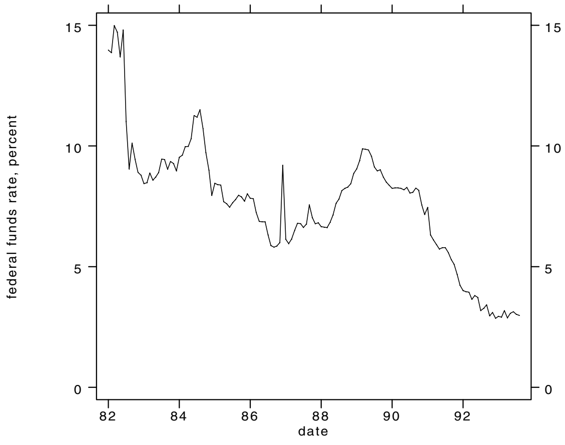

Figure 1

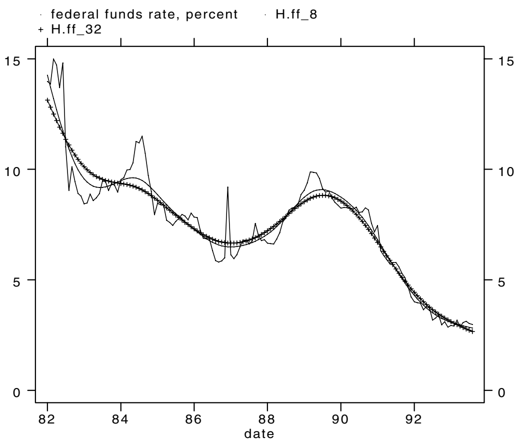

Figure 2

sts6 | Approximate p-values for unit root and cointegration tests

Craig S. Hakkio, Federal Reserve Bank of Kansas City, FAX 816-881-2199

Key related concepts in time series analysis are the notions of unit roots and of cointegration. Unfortunately, the most widely used statistics for testing for unit roots and for cointegration follow non-standard distributions. Thus, until recently, time series analysts have had to rely on only a few, limited tables of critical values for these test statistics. Using extensive Monte Carlo analysis, MacKinnon now has developed a low-order polynomial approximation to the asymptotic p-values for these test statistics. This insert describes `tauprob`, an `ado`-file that reports MacKinnon's approximate p-values. `tauprob.ado` is available on the STB-17 distribution diskette.

Background

Most time series techniques are applicable only to stationary variables, that is, variables whose distributions are independent of time. Many important real-world variables, though, are nonstationary. For example, the U. S. gross domestic product (GDP) does not have a constant mean; it grows apparently without bound reflecting growth in population, productivity, and the capital stock. Unbounded variables such as this can be modeled as having a unit root, that is, as following a time series process of the form

$$y_t = \mu + y_{t-1} + \epsilon_t \tag{1}$$

where y is, say, the log of GDP, μ is the drift or expected growth rate of GDP, and ϵ is a random disturbance. This type of time series process is called a random walk with drift. Equivalently, we say that y_t has a unit root.

It is not always known in advance that a variable follows a random walk. In this case, equation (1) becomes

$$y_t = \mu + \alpha y_{t-1} + \epsilon_t \tag{$1'$}$$

where $-1 \le \alpha \le 1$. To test whether y has a unit root (whether $\alpha = 1$), we rewrite $(1')$ by subtracting y_{t-1} from both sides of the equation, that is, we rewrite $(1')$ as

$$\Delta y_t = \mu + (\alpha - 1)y_{t-1} + \epsilon_t \tag{$1''$}$$

This equation can be estimated by ordinary least squares. The t-statistic on the parameter $(\alpha - 1)$ is called the Dickey–Fuller statistic and provides a test of the null hypothesis that $(\alpha - 1) = 0$ (or, equivalently, that $\alpha = 1$). The traditional t-statistic for $(\alpha - 1)$ follows a non-standard distribution. Fuller (1976) provides tables of critical values for the Dickey–Fuller statistic.

Many nonstationary variables are bound together by long-run, equilibrium relationships. For example, GDP and population are both nonstationary, but the growth of one depends on the growth of the other. If a linear combination of nonstationary variables is stationary, the variables are said to be *cointegrated.* (This definition is a bit too broad. See Engle and Granger (1987) for a formal definition of cointegration.) Intuitively, cointegrated variables cannot drift apart indefinitely.

For simplicity, consider two nonstationary variables, y and x. Engle and Granger suggest a two-step test to determine whether y and x are cointegrated. The first step is to estimate the equation

$$y_t = \beta_0 + \beta_1 x_t + \eta_t \tag{2}$$

by ordinary least squares. If y and x are cointegrated, this regression will recover the long-run relationship between the variables, that is, the estimates of β_0 and β_1 will define the stationary linear combination of y and x. As a consequence, the estimated residual from (2) will be stationary. The second step of the Engle–Granger test is just a Dickey–Fuller test of the hypothesis that $\widehat{\eta}_t$ is stationary. This test is carried out by estimating the equation

$$\Delta\widehat{\eta}_t = \mu + (\alpha - 1)\widehat{\eta}_{t-1} + \epsilon_t \tag{3}$$

and checking the t-statistic on $(\alpha - 1)$ against yet another set of critical values. The t-statistic in this regression follows a different non-standard distribution than the ordinary Dickey–Fuller statistic. Engle and Granger (1987) and Engle and Yoo (1991) provide tables of critical values for the Engle–Granger test.

MacKinnon’s method for finding approximate p-values

Programs to calculate the Dickey–Fuller and Engle–Granger statistic appeared in *sts2* (Becketti 1992). These programs are available in the library of time series programs that appears on this issue’s distribution diskette (see *sts7* below for details) along with programs to calculate the Phillips–Perron statistic, an alternative to the Dickey–Fuller statistic. These programs report the test statistic; users are still required to consult tables of critical values to determine the outcome of the test.

MacKinnon has developed a technique for calculating the p-values of Dickey–Fuller, Phillips–Perron, and Engle–Granger tests, eliminating the need to consult tables of critical values. The technique is straightforward. MacKinnon used Monte Carlo methods to construct detailed tables of critical values for these test statistics. Then he fit low order polynomials to these critical values. These polynomial expansions can be used to calculate approximate asymptotic p-values for a wide range of values of the test statistics.

MacKinnon’s approximations cover three versions of the test statistic, representing three versions of the Dickey–Fuller regression (or second-step regression in the Engle–Granger test):

$$\begin{aligned}\Delta y_t &= \mu_0 + (\alpha-1)y_{t-1} + \epsilon_t\\ \Delta y_t &= \mu_0 + \mu_1 t + (\alpha-1)y_{t-1} + \epsilon_t\\ \Delta y_t &= \mu_0 + \mu_1 t + \mu_2 t^2 + (\alpha-1)y_{t-1} + \epsilon_t\end{aligned} \tag{4}$$

As before, the test statistic, now called the τ statistic, is the t-statistic for $(\alpha-1)$. MacKinnon refers to the τ statistics based on the above equations as $\tau_c, \tau_{ct}, \tau_{ctt}$ respectively. The subscripts stand for “constant,” “constant and trend,” and “constant, trend, and trend squared.”

Lagged values of Δy_t are often added to the regression when calculating the Dickey–Fuller statistic. These lags serve to eliminate serial correlation in ϵ_t. When lagged values of Δy_t are included, the test is called the augmented Dickey–Fuller test. These lags do not affect the asymptotic distribution of τ. The Phillips–Perron statistic is robust to serial correlation, so lags are not included in that test.

MacKinnon’s formula for the p-value of the test statistic has the form

$$\Phi^{-1}(p) = \gamma_0 + \gamma_1\tau + \gamma_2\tau^2 + \gamma_3\tau^3 \tag{5}$$

where $\Phi(.)$ is the cumulative standard normal distribution function and the γ’s are the polynomial coefficients determined by MacKinnon. The values of the γ’s depend on the type of Dickey–Fuller regression (constant, constant with trend, or constant with trend and trend squared), the number of variables considered (1, for a unit root test; 2–6, for a cointegration test), and the value of τ itself. MacKinnon develops a separate set of γ’s for small values of p, that is, for values of p close to traditional cutoffs for significance levels. This separate set of γ’s permits greater accuracy in the region of greatest interest.

tauprob: a program to calculate MacKinnon’s p-values

`tauprob` reports MacKinnon’s approximate asymptotic p-values. The syntax of `tauprob` is

`tauprob` $\{$ `c` $\mid$ `ct` $\mid$ `ctt` $\}$ $\#_{\rm vars}$ $\#_\tau$

The first argument is a string indicating whether the p-value is for τ_c, τ_{ct}, or τ_{ctt}. The second argument is a number between one and six, inclusive, that indicates the number of variables in the cointegrating relationship. A “`1`” indicates a unit root test. The third argument is the value of the τ statistic. The p-value is not displayed. Instead, it is stored in the global macro `S_1`.

Example

As an example, we calculate the Dickey–Fuller test for a unit root in the monthly real exchange rate between the British pound and the U.S. dollar.

```
. use e1920, clear
(Exchange rates, 1920-1989)

. describe
Contains data from e1920.dta
  Obs:   840 (max= 32766)                Exchange rates, 1920-1989
 Vars:    14 (max=    99)
Width:    56 (max=   200)
  1. year        float  %9.0g            Year
  2. month       float  %9.0g            Month
  3. date        float  %9.0g            Date
  4. time         float  %9.0g            Time (=1 in January 1920)
  5. eeng         float  %9.0g            log(nominal ENG ex rate)
  6. eger         float  %9.0g            log(nominal GER ex rate)
  7. ecan         float  %9.0g            log(nominal CAN ex rate)
  8. efra         float  %9.0g            log(nominal FRA ex rate)
  9. eita         float  %9.0g            log(nominal ITA ex rate)
 10. qeng         float  %9.0g            log(real ENG ex rate)
 11. qger         float  %9.0g            log(real GER ex rate)
 12. qcan         float  %9.0g            log(real CAN ex rate)
 13. qfra         float  %9.0g            log(real FRA ex rate)
 14. qita         float  %9.0g            log(real ITA ex rate)
Sorted by:
```

`qeng` is the log of the real pound/dollar exchange rate. We wish to estimate the augmented Dickey–Fuller regression

$$A(L)\Delta\texttt{qeng}_t = \mu_0 + (\alpha - 1)\texttt{qeng}_{t-1} + \epsilon_t. \tag{6}$$

where $A(L)$ is a polynomial in the lag operator that indicates the inclusion of lagged values Δ`qeng` in the regression. We use the `dif` command to calculate `D.qeng` ($\equiv \Delta\texttt{qeng}_t$) and the `lag` command to calculate `L.qeng` ($\equiv \texttt{qeng}_{t-1}$). The `findlag` command reports the optimal number of lags of the dependent variable in a regression according to four criteria. (`dif`, `lag`, and `findlag` are documented in Becketti 1992).

```
. dif qeng
. lag qeng
. findlag D.qeng L.qeng
RMSE   AIC   PC    SC     (obs=830)
---------------------
   2     1    1     1
```

`findlag` indicates that one lag of `D.qeng` is probably adequate. (The root-mean-square-error criterion tends to overparameterize equations.) The `dickey` command (Becketti 1992) reports τ_c, that is, the t-statistic on $(\alpha - 1)$ from (6). Finally, we use `tauprob` to calculate the approximate asymptotic p-value of τ_c.

```
. dickey qeng
(obs=834, constant, no trend)
  Lags   tau
-------------
    0    -2.8
    1   -3.87
    2   -3.65
    3   -3.55
    4   -3.56
. tauprob c 1 -3.87
. di "$S_1"
.0022671722130163
```

The p-value is .002 indicating we can reject the null hypothesis of no unit root. In other words, the Dickey–Fuller test indicates that the real pound/dollar exchange rate is nonstationary. This p-value is consistent with published critical values. According to the table in Fuller (1976), the 5 percent critical value for τ_c is -2.88 and the 1 percent critical value is -3.44.

Now we repeat the test, including a trend term in the Dickey–Fuller regression.

```
. dickey qeng , trend
(obs=834, constant, trend)
  Lags   tau
-------------
    0   -2.87
    1   -3.93
    2   -3.75
    3   -3.65
    4   -3.65
. findlag D.qeng L.qeng time
RMSE   AIC   PC    SC     (obs=830)
---------------------
   2     1    1     1
. tauprob ct 1 -3.93
. di "$S_1"
.0110292682884601
```

The inclusion of a trend weakens somewhat the evidence of nonstationarity: the test is still significant at the 5 percent level, but not at the 1 percent level.

As a final check, we can use `tauprob` to calculate the asymptotic p-value for $\tau_c = -2.88$ and $\tau_c = -3.44$. According to the table in Fuller, the p-values should be .05 and .01, respectively.

```
. tauprob c 1 -2.88

. di "$S_1"
.0477244706026905

. tauprob c 1 -3.44

. di "$S_1"
.0096688520814145
```

References

Becketti, S. 1992. sts2: Using Stata for time series analysis. *Stata Technical Bulletin* 7: 18–26.

——. 1994. sts7: A library of time series programs for Stata. *Stata Technical Bulletin* 17: 30–32.

Engle, R. F. and C. W. J. Granger. 1987. Cointegration and error correction: Representation, estimation and testing. *Econometrica* 55: 251–276.

Engle, R. F. and B. S. Yoo. 1990. Forecasting and testing in co-integrated systems. *Journal of Econometrics* 35: 143–159.

Fuller, W. A. 1976. *Introduction to Statistical Time Series.* New York: John Wiley & Sons.

MacKinnon, James G. forthcoming. Approximate asymptotic distribution functions for unit root and cointegration tests. *Journal of Business and Economics Statistics.*

sts7 A library of time series programs for Stata

Sean Becketti, Stata Technical Bulletin, FAX 913-888-6708

This insert describes a library of time series `ado`-files that are available on this issue's distribution diskette. Some of these programs were presented in previous inserts. Others were mentioned in previous inserts with a promise of full documentation in a later insert. And some of these programs are completely new to readers of the STB.

This library is a "snapshot" of my personal directory of time series programs. The programs in this directory are used heavily by myself, by others at my research institution, and—judging by my mail—by a number of STB readers. This constant use generates a steady stream of requests for changes, improvements, and extensions in the programs. As a consequence, the programs are constantly evolving, and, at any given moment, the programs are in various stages of development. Some of them conform to all Stata standards for estimation routines and are fully debugged and documented. Others have limitations or are not documented yet.

I intend to include the latest version of this library on the STB distribution diskette from now on. The continual evolution of these programs has led to a proliferation of versions, many of them incompatible. It has become too difficult to publish updates of the programs individually. Republishing the entire directory each issue gives readers rapid access to the latest developments in these programs, and guarantees that the various programs will work together successfully.

Publishing time series programs as a library of `ado`-files makes sense because time series programs are highly interdependent. Time series analysis places demands on statistics and data analysis programs that do not occur in cross section or general statistics problems. The need to design a package around these demands accounts for the continued popularity of specialized time series programs such as RATS and TSP. The programs described in this insert incorporate a design philosophy that makes it easy to specify and analyze time series problems in Stata.

This insert has four parts. The first section describes how the time series library provides the basic features needed for time series analysis. The second section catalogs the user-level routines and indicates their stages of development. The third section describes the pool of utility programs used by the time series programs. This section is intended for users who wish to modify these programs or to add new time series programs to Stata. The fourth section discusses areas of future development.

How to do time series analysis in Stata

Modern time series programs have several features that make it easy to perform time series analysis. Stata, as it comes out of the shrink-wrap, offers relatively few of these features, and some of the ones it does offer are not in a form that is convenient for time series analysis. Here we list some of the more important of these features and comment on Stata's treatment of them.

Time variable: Time series programs provide a way to specify a time index or time variable. In some programs, a predefined variable (called `T` or `TIME` or `DATE` or ...) serves as the time index. In other programs, the user can state the frequency of the data (quarterly, annual, etc.) and the period covered by the data set, and the program checks that the number of observations matches the number implied by the sample coverage. A different approach is let the user specify the frequency of the data and the variable or variables that contain the date information.

Stata does not recognize any particular variable or variables as a time index, although it does store the sort order of the data, which is often the same as the time order. Stata does recognize an absolute time variable, the elapsed-date variable defined by such commands as `mdytoe` (see [5d] dates).

Sample coverage: Time series programs routinely display the sample used by a statistical command. It is important in time series analysis to know the beginning and ending dates of a sample and the number of gaps, if any, in the sample.

Most Stata commands report the number of observations used, but they do not report which observations were used nor do they report if any observations were skipped, that is, if there were any gaps in the sample.

Displaying dates: Time series programs display date identifiers in listings and graphs in a variety of easy-to-read formats. Knowing, for example, that data points are typically identified by their dates, rather than by their observation numbers, influences the design of even the simplest data management commands in a time series package.

Stata offers several commands for converting one date representation to another. For example, `etomdy` converts an elapsed-date variable to three variables containing the month, day, and year. These commands can produce readable versions of date information for listings, although these formats are not readily incorporated in Stata graphs.

Specifying time series models: Time series models are typically parameterized in terms of lag polynomials. For example, a relatively simple time series regression model is

$$A(L)y_t = \mu + B(L)x_t + \epsilon_t$$

where $A(L)$ and $B(L)$ are polynomials in the lag operator L, that is,

$$\begin{aligned} A(L) &= 1 - \alpha_1 L - \alpha_2 L^2 - \cdots - \alpha_p L^p \\ B(L) &= \beta_0 + \beta_1 L + \beta_2 L^2 + \cdots + \beta_q L^q \end{aligned}$$

and

$$\begin{aligned} Ly_t &\equiv y_{t-1} \\ L^k y_t &\equiv L^{k-1}(Ly_t) \\ &= y_{t-k} \end{aligned}$$

Modern time series programs make it easy to specify these kinds of models. To estimate the simple regression model above, for example, users typically need specify only the variables, y_t and x_t, and the lengths of the lag polynomials, p and q. The program then creates the needed lags of y_t and x_t and estimates the regression. The concept of a lag polynomial is fundamental, since the dynamic properties of the models are determined by functions of the lag polynomials.

Stata has no facilities for lag polynomials or other temporal connections between variables.

Dynamic forecasts: Dynamic forecasts are true *ex ante* forecasts. Static forecasts understate the uncertainty associated with a time series forecast. Consider the simplest autoregressive model:

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

The static h-period-ahead forecast of y_{t+h} is

$$\widehat{y}_{t+h} = \widehat{\alpha} y_{t+h-1}$$

If α is known with certainty, the variance of this forecast is σ_ϵ^2. If α is estimated, the uncertainty of this estimate also must be taken into account. The dynamic forecast of y_{t+h} is

$$\widehat{y}_{t+h} = \widehat{\alpha}\widehat{y}_{t+h-1}$$

The dynamic forecast adds the uncertainty about the lagged dependent variable to the uncertainty associated with the static forecast. The difference between the two types of forecast is often very large.

Stata's `predict` command provides only static forecasts.

Diagnostic statistics: Time series programs report a host of diagnostic statistics developed especially for time series analysis.

Stata has many commands to calculate diagnostic statistics, but some of the standard time series diagnostics are not included.

Model stability: Time series programs include commands for testing the stability of time series relationships.

Stata offers none of the standard methods for testing the stability of time series relationships.

The library of time series programs provides almost all the features discussed above that are not already provided by Stata. The first three features—specifying a time variable, displaying sample coverage, and displaying dates—are addressed explicitly by the programs `datevars`, `period`, `findsmpl` (see *sts2* (Becketti 1992) and *sts4* (Becketti 1993)) and implicitly by several of the other programs. The techniques used by `findsmpl` for displaying sample coverage add to the date formats already offered by Stata. `findsmpl` also reports any gaps in the sample.

The programs in the time series library use a common and convenient syntax for specifying a time series model. The common form of a time series command is

command-name varlist [*weight*] [`if` *exp*] [`in` *exp*] [`, current(`*varlist*`)`
`lags(`$\#_0, [\#_1[,\ldots]]$`) static(`*varlist*`)`]

The three options—`current()`, `lags()`, `static()`—handle most of the details needed to specify a time series regression model.

`current(`*varlist*`)` identifies variables in the *varlist* for which a contemporaneous term is to appear in the model. The default is to include only lagged values of explanatory variables in the model.

`lags(`$\#_0, [\#_1[,\ldots]]$`)` specifies the number of lags of each of the variables to include in the model. If fewer numbers than variables are specified, the last number specified is used for the remaining variables. If more numbers than variables are specified, the excess numbers are ignored.

`static(`*varlist*`)` specifies non-time series variables to include in the model. These variables can also be incorporated by setting their maximum lag to zero using the `lags()` option. The `static()` option provides a more efficient means of incorporating these variables.

These options are explained at greater length in *sts4* (Becketti 1993). A simple example, though, may make them easier to understand. Imagine we wish to estimate the time series regression

$$y_t = \mu + \alpha y_{t-1} + \beta_0 x_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \gamma_1 z_{t-1} + \gamma_2 z_{t-2} + \delta r_t + \epsilon_t$$

We could create the necessary lags of y_t, x_t, and z_t using either Stata's `generate` command or, more conveniently, the `lag` command included in the time series library. Then we could estimate the regression using the `regress` command. A better alternative is the time series regression command `tsfit`, also included in the time series library. The command

```
        tsfit y x z, current(x) lags(1,2) static(r)
```

creates all the lags, estimates and reports the regression, and stores information for use by follow-on time series commands.

To implement these options, the programs in the time series library have adopted the operator convention described in *sts2* and *sts4* (but see *ip5* in this issue for a limitation of this approach). Under this convention, applying the lag operator (`lag` program) to the variable `x` generates a new variable named `L.x`. The second lag of `x` is named `L2.x` and so on. This convention extends to the use of other operators, such as the difference (`D.`), growth (`G.`), and linear-least-squares projection operators (`P.`) have been defined. In *sts5* in this issue, Schmidt defines the operator `H.` to be the Hodrick–Prescott smoothing filter.

In addition to making it easy to specify time series models, the time series library provides `tspred`, a command for calculating dynamic forecasts. `tspred` can also calculate the static forecasts provided by Stata's `predict` command.

Diagnostic statistics are provided by `ac`, `coint`, `dickey`, `findlag`, `pac`, `regdiag`, and `xcorr`. These programs are described briefly in the next section. Perhaps the most important of them is `regdiag`, a general utility for calculating regression diagnostics including time series diagnostics.

Some of the time series diagnostics calculated by `regdiag` are useful for detecting model instability. The programs `chow`, `cusum`, and `quandt` provide formal tests of stability.

A catalog of programs

The following table lists the user-level programs in the time series library. Each program's status is indicated by a letter grade. An 'A' indicates a program that is safe for general use. An 'A' program has been documented—*in its current form*—in the STB and follows all Stata guidelines for an estimation command, where relevant (see [4] estimate). A 'B' program produces accurate results, but either is not fully documented, not completely compatible with the time series syntax described above, or not in conformance with the guidelines for an estimation command. Most 'B' programs receive that grade because they have been revised significantly since they were last documented. A 'C' program is incomplete in significant ways but can be used safely by an advanced Stata user. A 'D' program has serious deficiencies, however its code may provide a useful model to advanced Stata users writing their own time series programs. An 'O' program is obsolete, that is, it has been superseded by a newer program. An 'O' program is retained if it is still be called by one or two user-level programs. There are currently no 'D' or 'O' programs.

Command	Status	Documentation	Description
`ac`	A	*sts1*	display autocorrelation plot
`chow`	C	—	perform Chow test for a shift in regression coefficients
`coint`	B	*sts2*	perform Engle–Granger cointegration test
`cusum`	B	—	perform CUSUM test of regression stability. (**Note:** this name conflicts with Stata's `cusum` command for binary variables.)
`datevars`	A	*sts4*	specify date variables
`dickey`	B	*sts2*	perform unit root tests
`dif`	A	*sts2*	generate differences
`dropoper`	A	*sts2*	drop operator variables
`findlag`	B	*sts2*	find optimal lag length
`findsmpl`	B	*sts4*	display sample coverage
`growth`	A	*sts2*	generate growth rates
`lag`	A	*sts2*	generate lags
`lead`	A	*sts2*	generate leads
`pac`	A	*sts1*	display partial autocorrelation plot
`pearson`	A	*sg5.1*	calculate Pearson correlation with p-value
`period`	A	*sts2*	specify period (frequency) of data
`quandt`	B	—	calculate Quandt statistics for a break in a regression
`regdiag`	B	*sg20*	calculate regression diagnostics
`spear`	A	*sg5.1*	Spearman correlation with p-value
`testsum`	B	—	test whether the sum of a set of regression coefficients is zero
`tsfit`	A	*sts4*	estimate a time series regression
`tsload`	B	—	load an ad hoc time series equation into memory
`tsmult`	A	*sts4*	display information about lag polynomials
`tspred`	B	—	dynamically forecast or simulate a time series regression
`tsreg`	A	*sts4*	combined `tsfit`, `tsmult`, and `regdiag`
`xcorr`	A	*sts3*	calculate cross correlations

For more information on these programs, type "`help ts`" or "`help` *command-name*".

Utilities for time series analysis

Writing programs for time series analysis presents a variety of challenges. In developing this library of programs, I had to write a pool of utility programs to interpret the time series options, to generate lags, to manipulate the list of variables in a lag polynomial, and so on. I recommend that you familiarize yourself with these utilities, if you wish to write your own time series programs. A list of some of the most frequently used utility programs follows.

Command	Description
`_ac`	calculate autocorrelations, standard errors, and Q-statistics
`_addl`	"add" a lag operator to a variable name
`_addop`	"add" an arbitrary operator to a variable name
`_getrres`	calculate recursive residuals for a regression model
`_inlist`	determine whether a token appears in a token list
`_invlist`	determine whether a varname appears in a varlist
`_opnum`	decode the operators (and their powers) in a varname
`_parsevl`	parse a varlist to replace abbreviations
`_subchar`	replace one character in a string with another
`_ts_meqn`	parse a time series command and generate lags
`_ts_pars`	parse a time series command into useful macros

Future developments and call for comments

This library of time series programs is under constant revision and extension. Projects under development include programs to estimate rolling regressions, to estimate vector autoregressions, and to perform maximum-likelihood tests for cointegration. Older programs are being revised to bring them up to Stata's standards for estimation programs. A disadvantage of these constant revisions is the likelihood of inadvertently introducing errors into the programs. The advantage of constant revision is the ease and rapidity of fixing these errors. I encourage you to alert me to any errors or inconveniences you find.

References

Becketti, S. 1992. sts2: Using Stata for time series analysis. *Stata Technical Bulletin* 7: 18–26. Reprinted in *Stata Technical Bulletin Reprints*, vol. 2, pp. 209–218.

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

zz3.1 Computerized index for the STB (Update)

William Gould, Stata Corporation, FAX 409-606-4601

In the last issue of the STB, I introduced the STBinformer, a computerized index to every article and program published in the STB. The command (and entire syntax) to run the STBinformer is `stb`. Once the program has started, you can get complete instructions on how to search the index by typing `?` for help or `??` for more detailed help.

The version of the STBinformer included on the STB-16 distribution diskette included indices for the first fifteen issues of the STB. The STB-17 distribution diskette contains an updated version of the STBinformer which includes indices for the first *sixteen* issues of the STB. As the original insert stated, I intend to include an updated copy of this computerized index on every STB diskette. I encourage you to contact me with suggestions for changes and improvements in the program.

Reference

Gould W. 1993. Computerized index for the STB. *Stata Technical Bulletin* 16: 27–32.