STATA TECHNICAL BULLETIN

January 1995

STB-23

A publication to promote communication among Stata users

Editor

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

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

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

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

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

Contents of this issue

an51 Call for suggestions

Sean Becketti, Stata Technical Bulletin, stb@stata.com, FAX 914-533-2902

This issue contains, in addition to the usual articles, the announcement that Stata 4.0 is available. This new release of Stata includes many improvements requested by Stata users. Perhaps the most frequently requested improvement is Stata for Windows. STB contributors and readers also will be pleased with the new features directed at those of us who write ado-files. Some of the highlights of the new release are discussed in *an52* and *zz3.7* in this issue.

The release of Stata 4.0 is an opportune time to ask you, the readers of the STB, to contribute your suggestions for the next version of Stata. Stata is under continual development, and that development is strongly influenced by requests from Stata users. Many of those requests come from conversations with Stata users. Other developments are suggested by the pattern of contributions to the STB.

When Stata was new and its user community was relatively small, it was easy to know what users most wanted to see in subsequent releases. Stata is now a mature product, and a sampling of recent STBs indicates that the Stata user community is large and diverse. As a consequence, it has become more difficult to determine which additions to Stata would be most valuable.

I invite you to send me your suggestions for the next release. Feel free to suggest large, interconnected systems of features (for example, a complete system for time series analysis to replace and augment the time series library, *sts7.5*). Also feel free to suggest smaller items that would make your use of Stata more pleasant and productive (for example, altering the `use` command to automatically repartition memory when needed). Send me your complaints as well; these are often the best source of development ideas. Are there long-standing features of Stata that need "remodeling"? Let me know. I will communicate your suggestions to Stata Corporation, and I will summarize them in the STB.

To make it easier for you to send me your suggestions (and your submissions), we have added an EMAIL address for the STB. You now have four ways to reach me: voice mail (914-533-2278), FAX (914-533-2902), EMAIL (stb@stata.com), and, of course, the postal service. I look forward to hearing your ideas.

an52 Stata 4.0 released

Patricia Branton, Stata Corporation, FAX 409-696-4601

Stata 4.0 is now shipping. You should have already received information from us on the upgrade but, if not, call or fax us, or email stata@stata.com and we will send the information to you. Stata 4.0 has many new features.

In addition to Stata for DOS, Macintosh, and Unix, we now have Stata for Windows. Stata for DOS users can upgrade to the DOS version, the Windows version, or both.

Finally, given the recent problems uncovered with Intel's Pentium chip (see *os15* in this issue), the Intercooled versions of Stata for DOS and Stata for Windows are available in Pentium-aware forms. The Pentium-aware versions watch for the division problem, correct it if it occurs, and run about 3% slower than the regular Intercooled versions.

an53 Implications of Stata 4.0 for the STB

Sean Becketti, Stata Technical Bulletin, stb@stata.com, FAX 914-533-2902

I expect that most STB subscribers will upgrade quickly to Stata 4.0. Because of publication lags, some STB inserts will present commands developed under Stata 3.1. These inserts present no problem, because Stata 4.0 can run Stata 3.1 programs. The reverse is not always true, though. Stata 4.0 provides some new programming features not available in Stata 3.1; thus, users who fail to upgrade will not be able to run some commands published in future issues of the STB. If you try to run a Stata 4.0 ado-file using Stata 3.1, nothing terrible will happen. When you invoke the command, Stata will display the message "`version 4.0 not supported`".

In the next few issues of the STB, I will include an editor's note with each insert that presents a new command that requires Stata 4.0. After a transitional period, I will drop the notes and you should then assume that all published commands are certified only to work with Stata 4.0. I will include a brief announcement in the front of the first issue that drops the editor's notes. Of course, all inserts in this issue were submitted prior to the release of Stata 4.0; thus, they all are certified to work with Stata 3.1.

If you do not upgrade to Stata 4.0, be sure you do *not* install the official updates in the `crc` directory. (See the `README` file on the distribution diskette.) These updates supersede the ado-files in the *current* release of Stata. There are no official updates in this issue, because Stata 4.0, the current release, has just been announced and no updates have accumulated yet. Starting with the next issue of the STB, the `crc` directory will contain official updates for Stata 4.0. Do not install these updates if you are still running Stata 3.1. (Do install them, though, if you are running Stata 4.0.)

Finally, the DOS STB diskette can be used to update Stata for Windows.

crc37 | Commonly asked questions about Stata for Windows

During the development of Stata for Windows (SW), test users and other interested parties asked us many questions about the relation of Stata for Windows to the Stata with which you are already familiar. This insert presents the most frequently asked questions along with their answers.

1. Is SW like Stata for DOS? Can I can still type commands and use my old do-files and programs?

   Answer: Yes. SW has a Windows look and feel and a number of useful features, but you still type the same commands and, in general, use it in the same way as Stata for DOS.

2. Can data sets be moved easily from Stata for DOS to Stata for Windows and vice-versa?

   Answer: Yes. Stata data sets are portable across all versions of Stata: Stata for Windows, Stata for DOS, Stata for Macintosh, and Stata for Unix.

3. Does SW need a lot more memory than Stata for DOS?

   Answer: Yes and no. SW uses roughly 250K more memory than Stata for DOS, which is not much. Windows itself, though, consumes memory in substantial quantities. If you have been running Stata happily from a DOS window—that is, from inside Windows—then you will be able to run Stata for Windows and only a small amount of extra memory will be consumed by SW. If, however, you have been running Stata for DOS from DOS—outside of Windows—and you have been using nearly all the computer's memory, you will probably want to add more memory before switching to Stata for Windows.

4. Can I install both Stata for Windows and Stata for DOS on the same computer?

   Answer: Yes. Install SW first, ignoring the Stata for DOS diskettes. Then install Stata for DOS following the instructions in [8] windows, page 415 of volume 1. These instructions install Stata for Windows and DOS in the same directory and let them share files; the only additional disk space required is for the DOS Stata `.exe` file.

5. Can Stata for Windows use virtual memory?

   Answer: Yes, but we recommend you add more memory to your computer if you use large data sets frequently, because using virtual memory hurts the performance of your programs. To determine whether virtual memory is enabled, open the Main program group and double-click on the Control Panel icon. Double-click on the 386 Enhanced icon. Press the *Virtual Memory* button and follow the instructions. See your Windows documentation for more information.

6. Is Stata for Windows slower than Stata for DOS?

   Answer: No, although you may think we are wrong the first time you run SW. Stata for Windows scrolls output slower than Stata for DOS—how much slower depends on your video card—and this may give you the feeling that SW is slower. All calculations are made at roughly the same rate in both programs, with Stata for Windows sometimes faster and sometimes slower but never differing from Stata for DOS by much.

7. Can I run multiple Stata sessions with Stata for Windows?

   Answer: Yes. A new Stata session is brought up every time you double-click on the Stata icon.

8. Does Stata copy graphs to the clipboard in Metafile (WMF) format or just as bitmaps (BMP)?

   Answer: You can choose either, but the default is Metafile.

9. Can I run Stata for Windows using OS/2 Warp?

   Answer: Yes. Install Stata from the Windows' Program Manager; see *crc38* below for details.

10. Does Stata run slower under OS/2?

    Answer: No.

crc38	Installing Stata for Windows under OS/2 Warp

Installation

OS/2 Warp users install Stata for Windows from the Windows Program Manager.

1. Open *OS/2 System* on your desktop.
2. Open *Command Prompts*.
3. Open *WIN-OS/2 Full Screen* or *WIN-OS/2 Window*.

You are now in Program Manger. Follow the standard installation instructions in the *Getting Started with Stata for Windows* manual.

Using Stata for Windows

Use Stata from Program Manager, just as you would under Windows.

Advanced installation

After installing Stata, you can set things up so that you can invoke Stata without bothering to get into Program Manager first, but this is more complicated. The rest of this insert explains how to do this.

Advanced installation—Background

OS/2 will require you to fill in some tables with filenames, options, etc., and you will need to know what to type.

The Stata executables are named `wstata.exe` (Intercooled Stata), `wstatanc.exe` (Pseudo-Intercooled Stata), and `wsmstata.exe` (Small Stata). They reside in the Stata directory, probably `c:\stata`. Distinguish this directory from the working directory, which is probably `c:\data`. The working directory will be the current directory when Stata is invoked.

The Stata executables, with the exception of Small Stata, allow a `/k` option or parameter. `/k1000` means allocate 1 megabyte (1,000K) of memory to Stata's data areas. `/k2000` means 2 megabytes, and so on.

Advanced installation—Step by step

1. Install Stata as above.
2. Open *OS/2 System*. Open *System Setup*. Open *Add Programs*. Check *Search for & Select Programs to Add* and click on *OK*.
3. A menu will appear. Check *Windows Programs* and *Windows Groups* and click on *OK*.
4. OS/2 will present a long list of files. All are automatically selected. Click on *None* to deselect them.
5. Click on the *Other Programs* button. Another list of files will be presented. Look for the Stata executable (`wstata.exe`, etc.) and click on it.
6. Fill in the *Program Title* with whatever you desire—it will be "WSTATA.EXE" right now but "Intercooled Stata" would be a better choice.
7. Fill in *Parameters* with the `/k` option; skip this step if you are using Small Stata. Remember, there are no blanks between the `/k` and the following number; thus, you type `/k2000` to indicate 2 megabytes.
8. Fill in *Working Directory* with `c:\data` or whatever is appropriate. It will be filled in with `c:\stata` right now, but you should change this.

9. Click on *Add*. Click on *OK*; this takes you out of the *Other Programs* dialog box. Click on *OK* again.

You now have a new folder called *Additional Windows Programs*. You can rename this folder or drag the icons from it to another folder; see your OS/2 documentation.

os15 The impact of the Pentium FDIV bug on Stata users

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

The bug in the Pentium affecting division calculations has received widespread media coverage in the past two months. For 1 in 9 billion randomly chosen dividends and divisors, the Pentium returns a result that lacks precision (see the Technical Appendix below for details). This insert is an attempt to judge the likely impact of the Pentium's problem on Stata users.

The difficulty statisticians have in assessing this problem is that we—even those of us who write statistical software—have no accurate idea of how many divisions we do. I have calculated estimates of the number of divisions performed during different types of Stata sessions by modifying Stata's internal source code to count FDIV (floating point division) operations. Using this modified version of Stata, I ran some of the test scripts we use for certifying Stata. The results of these test runs are shown in Table 1.

Table 1. Test runs using a modified version of Stata

Test script	Number of FDIVs	Number of errors	Lines of output	Number of pages	FDIVs per page
SC-1	998,527	0	20,782	378	2,642
SC-2	1,331,294	0	20,867	379	3,512
SC-3	332,665	0	6,992	127	2,619
SC-4	504,675	0	5,039	92	5,486
SC-5	15,195	0	4,639	84	181
Subtotal	3,182,356	0	58,319	1,060	3,002
SC-6	13,884,782	0	3,221	59	235,335
Total	17,067,138	0	61,540	1,119	15,252

The first column of Table 1 reports Stata Corporation's internal name for each test script. Column two reports the number of FDIV operations performed by each test script. The third column displays the number of FDIV operations affected by the Pentium flaw. As Table 1 reports, no FDIV errors were detected in any of these test runs. The fourth column indicates the number of lines of output generated by Stata in running each script. Column five reports the number of lines of output divided by 55 and rounded to give an approximate number of pages of output. Column six displays the average number of FDIV operations per page of output. Column six is equal to column two divided by column five.

These tests produced a total of 1,119 pages of output and performed 17,067,138 double-precision divides. In all of this output, the FDIV bug did not bite once. If the FDIVs in these test scripts can be regarded as independent draws from the set of possible divisions, and if the probability, p, that a random FDIV operation produces an error is 1 in 9 billion, as reported, the probability of observing no errors in this many divisions is

$$P(\text{no errors}) = (1-p)^{17,067,138} = .998105$$

Alternatively, the probability of observing one or more errors in these runs is 0.001895. An exact, one-sided 97.5 percent confidence interval for p based on observing zero failures in 17,067,138 trials is 0 to 2.16×10^{-7}, which includes the 1 in 9 billion (1×10^{-9}) rate.

To get a better sense of the likely impact of the Pentium bug on users of statistical software, I am now going to leave firm ground and make more speculative calculations. Note that the number of FDIVs per page of output reported in Table 1 is much higher for test script SC-6 than for any of the others. SC-6 focuses on iterative techniques. In particular, SC-6 tests some of Stata's maximum-likelihood features. In the next two sections, I estimate the likelihood of encountering the Pentium error separately for iterative and noniterative procedures.

Noniterative procedures

Taking the scripts other than SC-6, there are an average of 3,002 FDIV operations per page of output. The data sets on which these scripts were run, however, are small—they average about 100 observations. Were the data sets larger, I would expect more divides per page than the 3,002 reported. How much more? I conjecture that the number of divides is linear in the log of the number of observations. In this case, the total number of divisions per page of output would be

$$D = (3,002/\log_{10} 100) * \log_{10} N = 1501 * \log_{10} N$$

Table 2 applies this formula to produce estimates of the probability of encountering one or more division errors in noniterative procedures.

Table 2. Estimated probability of at least one error in noniterative procedures

Data set size	Number of FDIVs	Probability of a failure in 1,000 pages	Number of pages for $P = .01$	Number of pages for $P = .05$
100	3,002	.000333	29,908	2,077,638
1,000	4,503	.000500	20,143	1,385,499
10,000	6,004	.000667	14,955	1,038,819
100,000	7,505	.000834	12,055	831,300

The first column of Table 2 lists N, the assumed data set size. The second column lists D, the estimated number of FDIV operations assuming D is linear in the log of N. Column three reports P, the probability of at least one FDIV error in 1,000 pages of output. Columns four and five reverse this calculation, reporting the minimum number of pages of output required to raise P to 0.01 and 0.05, respectively.

For instance, if you use data sets of roughly 1,000 observations, the probability you will observe one or more FDIV errors in 1,000 pages of Stata output is 0.0005. Looked at differently, you would have to produce 20,143 pages of computer output before the probability of the bug biting would reach 0.01. In 1,385,449 pages of output, the bug is as likely to strike as not.

Stata users should be able to place themselves somewhere in this table. I suggest you calculate the average number of pages of output you generate per week, multiply this number by 50, and use the table to obtain an estimate of your annual risk.

Iterative procedures

Iterative procedures, such as maximum-likelihood estimation, involve many more FDIV operations per page than noniterative procedures, and the number of divisions rises more rapidly as the data set size increases. Test script SC-6 estimates complex maximum-likelihood models (such as Heckman's selection model) and averages 235,335 FDIV operations per page of output generated. The script runs on data sets averaging 200 observations. For the iterative procedures used in SC-6, it seems reasonable to suppose that the number of FDIV operations increases approximately linearly with the number of observations. Thus I use the approximation

$$D = (235,335/200) * N = 1176.675 * N$$

to estimate the number of divisions performed for other data-set sizes. This approximation underlies Table 3.

Table 3. Estimated probability of at least one error in iterative procedures

Data set size	Number of FDIVs	Probability of a failure in 1,000 pages	Number of pages for $P = .01$	Number of pages for $P = .05$
100	117,668	.01299	764	53,025
1,000	1,176,675	.12256	77	5,301
10,000	11,766,750	.72948	8	530
100,000	117,667,500	1.00000	<1	53

The columns in Table 3 are defined the same as in Table 2. As with Table 2, users of statistical software should be able to place themselves somewhere on this table.

Table 3, however, reflects the probability of any error occurring. In the iterative techniques tested in script SC-6, division errors that occur during the maximization process will not affect the final result. Mistakes in intermediate iterations may affect the path by which the maximum is ultimately reached, but the final estimates will be correct as long as there are no division errors during the last iteration and the calculation of the reported statistics. As a consequence, the figures reported in Table 3 overestimate the likelihood that the Pentium bug will affect the results of an iterative procedure.

To correct this bias, I return to the results from the other test scripts. I use the observed 3,002 divides per page as an estimate of the number of divides per page of output generated from the onset of the last iteration. This time, however, I extrapolate these results to other data set sizes by assuming the number of divisions is linear in data set size:

$$D = (3,002/100) * N = 30.02 * N$$

This approximation leads to Table 4.

Table 4. Estimated probability that an error will affect the results of iterative procedures

Data set size	Number of FDIVs	Probability of a failure in 1,000 pages	Number of pages for $P = .01$	Number of pages for $P = .05$
100	3,002	.000333	29,908	2,077,638
1,000	30,020	.00333	3,014	207,826
10,000	300,200	.032805	301	20,781
100,000	3,002,000	.283629	30	2,078

Table 4 presents what I think is a reasonable approximation to the effect of the FDIV bug on the final results of iterative techniques.

Summary

The debate over the significance of the Pentium bug has generated a wide range of assessments, varying from apocalyptic claims made by some Internet correspondents to the sanguine claims, made by Intel, that the bug is expected to strike a user only once every 27,000 years. (A variant of this assessment is the claim that the mean time to the bug biting is longer than the mean time to failure of the other components of the computer.) Both types of claims are misleading for users of statistical software.

As Table 2 shows, most users of noniterative statistical procedures are indeed unlikely to be struck by the bug. Heavy users of statistical software—users producing, say, 500 pages per week—do have a non-negligible chance of observing the bug in a year.

Users of iterative statistical procedures have more reason for concern. If you perform maximum-likelihood estimation routinely on data sets of 1,000 or more observations, you are likely to observe the bug at least once over the course of a year. Users of larger data sets will almost certainly encounter the bug and more than once. Nevertheless, the chances that the bug will strike at a point where it affects the final results are much smaller than this. You would have to produce 500 pages per week of maximum-likelihood estimates on 10,000-observation datasets to have a 50 percent chance of being affected by the bug.

How much would an error affect the results of a statistical analysis? I suspect that the probability that an error has a meaningful affect on the results is small, but I have no way of casting any empirical light on that matter. For most real-world statistical applications, though, common sense suggests that unavoidable errors in the raw data are a much likelier source of misleading results than any Pentium-induced division errors.

Cleve Moler, Chairman and Chief Scientist of the Mathworks (the manufacturers of MATLAB) was the first to suggest a software workaround to the Pentium's bug, and he even offered code in C via Internet. Tim Coe of Vitesse Semiconductor has developed a thorough model of the Pentium's problem which has also been available via Internet. Great appreciation needs to be expressed to both these people, and to their respective organizations, for helping those of us in the scientific community understand and deal with this problem.

Finally, despite my estimates that the bug is highly unlikely to cause significant errors, we now offer a Pentium-aware version of Stata, and I use it on my Pentium.

Technical appendix: Characteristics of the FDIV bug

The Pentium usually calculates x/y correctly. There are, however, values of x and y for which x/y is calculated with reduced precision. For a randomly chosen x and y pair, the probability of reduced precision in the x/y calculation is about 1 in 9 billion.

In assessing the impact of the Pentium's problem on user of statistical software, I have used 1 in 9 billion as the probability of a division error, but this is not completely accurate because the numbers that appear in division in statistical packages are not uniformly distributed. Many formulas involve division by the number of observations, an integer, or by normalization factors such as 2. If the Pentium were to incorrectly divide by 2, it would be useless for statistical calculations, even if the rate for randomly chosen x and y combinations is 1 in 9 billion. Fortunately, the Pentium does divide by 2 correctly, but the question still remains: are the numbers more frequently used in statistical calculations more likely generate FDIV errors? In other words, does the 1 in 9 billion rate understate the probability of an error in statistical calculations?

I do not think so. We now know quite a lot about the problem numbers due to the work of Tim Coe of Vitesse Semiconductor. The nature of the Pentium bug is such that if x/y is a problem calculation, then $(x*2^a)/(y*2^b)$ is also a problem, where a and b are any integers. We also know that problem divisors are concentrated in five bands at every power of 2 and that, for a divisor in the bands, the probability of error is about 1 in 200,000. For divisors outside the bands, there are no errors (see Figure 1).

It is, of course, the concentration of the division errors at small powers of 2, near the so-called "nice" numbers, that is of greatest concern. One of the problem bands comes close, for instance, to the nice number '3', being only a relative distance of $3*10^{-7}$ away. This is, however, a considerable distance for double-precision calculations where the roundoff error on any single calculation is on the order of 10^{-16} and the combined error of most calculations is on the order of 10^{-12}.

Even for roundoff error on the order of 10^{-10}, the problem is almost three orders of magnitude away from 3, and, in any case, the nice numbers in statistical formulae tend to be inputs into calculations and not the results of calculations. When nice numbers are the results of calculations, however, it is important that the calculations be performed in double precision. Most statistical packages, Stata included, do this.

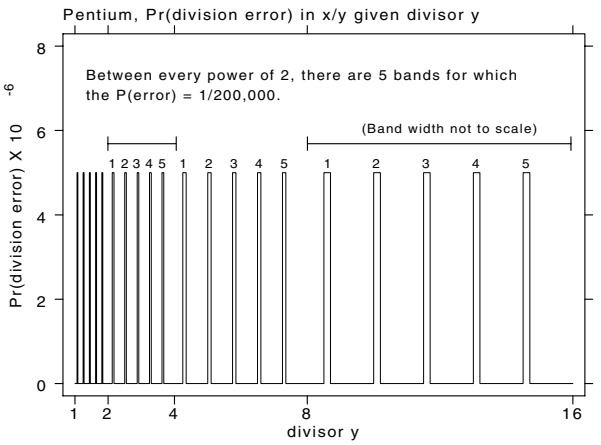

Figure 1

dm24	Producing formatted tables in Stata

William H. Rogers, Stata Corporation, FAX 409-696-4601

Most Stata commands make their results available for further computation. This characteristic makes possible the many ado-files that add to Stata's usefulness. In this insert, we introduce a new command, `table`, that takes advantage of this feature to produce nicely formatted tables from the results of Stata computations. These tables are suitable for placement in a draft word processing document.

We regard the `table` command presented here as a good start, but we would like to improve it in the future. We present it now for two reasons. First, we have found it useful, thus we believe you will find it useful as well. Second, we are interested in getting your reactions and suggestions for improvements. We think it would be useful if Stata produced more general types of tables, and it also might be desirable to integrate these tables more smoothly with some of the more-popular word processing and text formatting systems. Let us know what you think.

The table command

The `table` command builds tables according to definitions you supply. A table is defined by its stubs (the first row and first column of the table) and by its interior rows and columns, the ones that contain data. Each row and column has a name, supplied by you. The cells of the table may contain either numbers or text, and each cell may have its own display format.

There are six table subcommands. Each command begins with the word `table`. For example,

`table define` *table-name* `"`*table-title*`"`

defines a new table. Table subcommands are used to define new tables, specify the elements of each cell, print tables, and drop tables. The rows and columns of the table are defined implicitly by referring to them. Rows and columns are added to the table in the order they are defined.

The table subcommands

`table define` *table-name* `"`*table-title*`"`

The `table define` subcommand defines a new table. Table names must be four characters or less. More than one table can be active at a time.

`table number` *#* [`, table(`*table-name*`) row(`*row-name*`)`
`col(`*column-name*`) font(`*color*`)`]

— or —

`table text "`*text*`"` [`, table(`*table-name*`) row(`*row-name*`)`
`col(`*column-name*`) font(`*color*`)`]

The `number` and `text` subcommands define the contents of table cells. If the table, row, or column name is not specified, the most-recently specified table, row, or column is assumed. Row and column names must be eight alphanumeric (including underscore) characters or less. The stub row and column are always referred to by the reserved name "`stub`".

The `font()` option currently specifies only the color. Allowed colors are `yellow` (the default for numbers), `green` (the default for text), `white`, `blue`, and `red`.

The *#* in the `number` subcommand may be either a constant or a more general expression. If the expression contains spaces, it must be enclosed in quotes. After the expression is evaluated, the result is stored in a matrix. As a consequence, the precision of cell entries may be changed. To control the precision of cell entries, store numbers as text.

`table format` *format* [`, table(`*table-name*`) row(`*row-name*`)`
`col(`*column-name*`) font(`*color*`)`]

The `format` subcommand controls the display formats of cell entries. The *format* may be any Stata display format. As with the `number` and `text` subcommands, the most-recently specified table is assumed if none is explicitly indicated. However, in contrast to these subcommands, if no row or column is indicated, the format is applied to all rows and columns.

`table print` [`, table(`*table-name*`)`]

The `print` subcommand displays a table.

`table drop` *table-name*

The `drop` subcommand erases a table from Stata's memory together with all associated matrices and macros. Tables take up quite a bit of space, so it can improve performance to drop a table when it is no longer needed. Defining a new table with an existing table name effectively drops the existing table.

Note: There must be space on both sides of the comma (`,`) when options are specified.

Example

We use the familiar automobile data to illustrate the use of the `table` command. We want to display the estimated effects of automobile weight, displacement, mileage, and repair record on price together with summary statistics for each explanatory variable. The first interior row of the table will display the number of non-missing observations for each variable. The second and third interior rows will display the means and standard deviations, respectively, of each variable. The fourth interior row will display the estimated slope coefficient from a regression of price on each variable by itself. The final row will display the estimated coefficient on each variable from the regression that includes all the variables simultaneously as regressors.

Note that the upper-left cell, in the stub row and column, will display the string "`stub`" unless it is overwritten, as it is in this example.

```
. use auto
(1978 Automobile Data)
. table define A
. table text " " , row(stub) col(stub)
. table text "N" , row(N)                       /* Define row order */
. table text "Mean" , row(Mean) col(stub)
. table text "SD" , row(SD)
. table text "Univariate Coeff." , row(uni)
. table text "Multivariate Coeff." , row(multi)
. capture program drop doit     /* automate the univariate regressions */
. program define doit
  1.         quietly summ `1´
  2.         local NN = _result(1)
  3.         table text "`NN´" , row(N) col(`1´)        /* override format */
  4.         table number _result(3) , row(Mean) col(`1´)
  5.         table number "sqrt(_result(4))" , row(SD)
  6.         qui reg price `1´
  7.         table number _b[`1´] , row(uni)
  8.         table text "`2´" , row(stub)
  9. end
. doit weight  "Weight(lb)"
. doit displ   "Displacement"
. doit mpg     "MPG"
. doit rep78   "Repair"
. regress price weight displ mpg rep78
  Source |       SS       df       MS                  Number of obs =      69
---------+------------------------------               F(  4,    64) =    9.94
   Model |   221079849     4  55269962.3               Prob > F      =  0.0000
Residual |   355717110    64  5558079.84               R-square      =  0.3833
---------+------------------------------               Adj R-square  =  0.3447
   Total |   576796959    68  8482308.22               Root MSE      =  2357.6
------------------------------------------------------------------------------
   price |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  weight |   .8090008   1.120979      0.722   0.473      -1.430412    3.048414
   displ |   11.83023    8.51476      1.389   0.170      -5.179954    28.84042
     mpg |  -58.11299   83.25482     -0.698   0.488      -224.4336    108.2076
   rep78 |   875.7999    321.075      2.728   0.008       234.3789    1517.221
   _cons |   -394.836   3764.689     -0.105   0.917      -7915.666    7125.994
------------------------------------------------------------------------------
. table number _b[weight] , row(multi) col(weight)
. table number _b[displ] , row(multi) col(displ)
. table number _b[mpg] , row(multi) col(mpg)
```

```
. table number _b[rep78] , row(multi) col(rep78) font(red)
. table format %10s , row(N)
. table format %10.2f , col(weight)
. table format %10.2f , col(displ)
. table format %10.2f , col(mpg)
. table format %10.2f , col(rep78)
. table print , table(A)
                    Weight(lb) Displacement        MPG     Repair
                 N         74           74         74         69
              Mean    3019.46       197.30      21.30       3.41
                SD     777.19        91.84       5.79       0.99
  Univariate Coeff.       2.04        15.90    -238.89      19.28
Multivariate Coeff.       0.81        11.83     -58.11     875.80
```

Note that room is made for the longest element and that tables are formatted uniformly. I had to do some work to make the `N`'s come out as integers without breaking the column format.

gr16	Convex hull programs

J. Patrick Gray, University of Wisconsin-Milwaukee, EMAIL jpgray@alpha1.csd.uwm.edu
Tim McGuire, Pillsbury College, Owatonna, Minnesota 55060

This insert presents two ado-files, `conhull` and `condraw`, that find and graph points on the convex hull of a set of points. `conhull` identifies and graphs the outermost hull or all the hulls in a series of point sets. The program can also generate hulls for each level of a categorical variable. `condraw` graphs the hulls calculated by `conhull`.

Calculating and graphing convex hulls

The syntax of `conhull` is

`conhull` *varlist* `, hull(#)` [`coname(`*var1*`) nograph group(`*var2*`)`]

`conhull` generates a new variable, `coname`, containing the hull number of the points in the *varlist* and graphs each hull.

Options

`coname(`*var1*`)` is the variable name for the convex hull number. The default is `conhull`. Since `conhull` is not a temporary variable, the program will give a "`conhull already defined`" error message if your data set already contains a variable named `conhull`.

`nograph` suppresses the graphing of each hull. If you expect a large number of hulls, it may be best to allow `conhull` to assign the hulls without graphing and to use `condraw` to graph the hulls that interest you. The default is to graph each hull. Each graph contains all points referenced by their group value. All points on the lines of the graph are on the hull.

`group(`*var2*`)` is the name of a categorical variable for which convex hulls will be generated. The default is to use the entire sample.

`hull(#)` specifies the number of hulls to be generated for each group. `h(1)` would calculate only the outermost hull, for example. The default value is 500.

Graphing previously calculated hulls

The syntax of `condraw` is

`condraw` *varlist* `, grno(#) hull(#)` [`dr(`*var1*`) group(`*var2*`) vid(`*var3*`)`]

`condraw` produces graphs using the hull numbers generated by `conhull`. In order to do this it creates new observations and new variables. The graphs that are produced can be modified using the Stata graph options.

Options

`dr(`*var1*`)` names a marker variable that allows the points in the graph to be closed. The default is `dr`. As this is not a temporary variable, a "`dr already defined`" error message will appear if you use the default and your data set contains a variable named `dr`.

`grno(#)` specifies which class of the categorical variable identified by `group()` is to be used in the graph. The default is to use all classes. If more than one, but fewer than all, classes are desired, the user should use the default option and then eliminate the undesired classes when regraphing the data.

`group(`*var2*`)` is the name of a categorical variable for which convex hulls were generated. The default is to assume the entire sample was used to generate the hulls.

`hull(#)` specifies the hull to be graphed. The default is to graph the outermost hull.

`vid(`*var3*`)` names the new variables needed for graphing the hulls. The default is "`hv*`". For example,

```
condraw x y, g(smoke) h(3)
```

graphs the third outermost hull for the levels of the categorical variable `smoke`. If `smoke` has three levels, three new variables are created: `hv1`, `hv2`, and `hv3`. As these are not temporary variables, an "`hv1 already defined`" error message will appear if you use the default and your data set contains a variable named `hv1`.

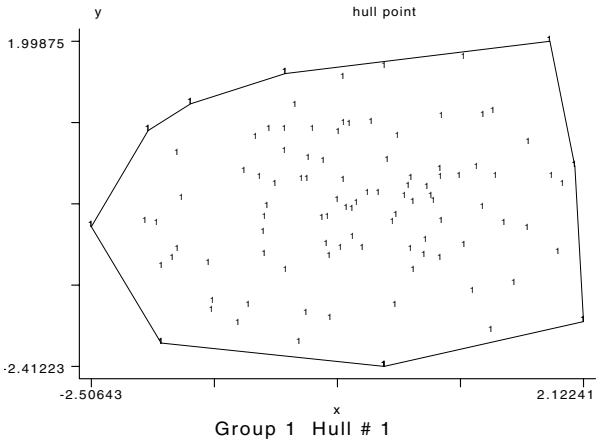

Figure 1

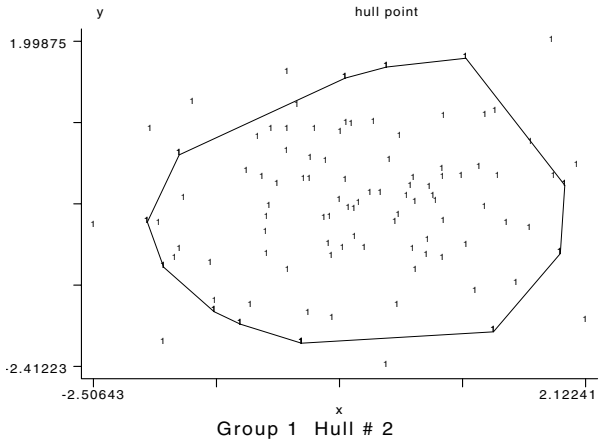

Figure 2

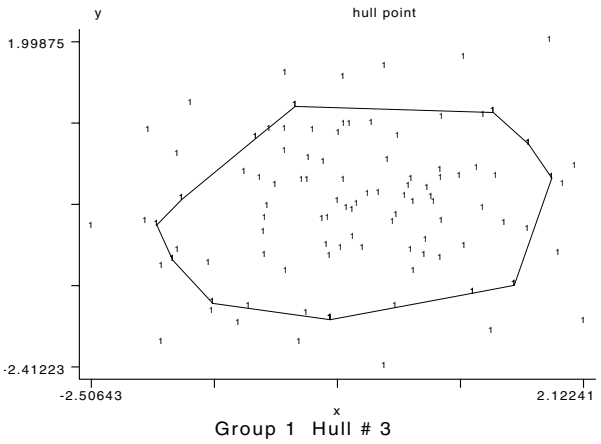

Figure 3

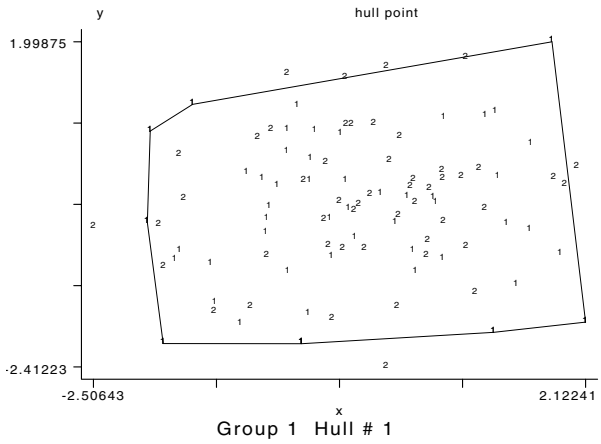

Figure 4

Discussion

Analysis of spatial data often requires identifying the points that comprise the convex hull of a scatter of points in two-dimensional space. `conhull` and `condraw` perform this task using an algorithm proposed by R. A. Jarvis (1973).

The operation of `conhull` and `condraw` can be illustrated with the following example:

```
. drop _all
. set obs 100
obs was 0, now 100
. generate x=invnorm(uniform())
. generate y=invnorm(uniform())
. generate byte group=1
. replace group=2 if _n>50
(50 real changes made)
. conhull x y, hull(3)
```

(three graphs appear, see Figures 1–3)

```
The hull number for each point is saved as conhull
. conhull x y, hull(3) g(group) c(hulls)
```

(six graphs appear, see Figures 4–9)

```
The hull number for each point is saved as hulls
. condraw x y hulls,g(group) h(2)
```

(graph appears, see Figure 10)

```
Variables:  hv1 hv2 were created
You can redraw the graph with the command:
graph y hv1 hv2 x, c(.ll) s([group][group])
You can now change the graph options, add titles, save, etc.
To use the same data set to generate a different graph or to eliminate
all the created variables use the following sequence of commands
(making sure to follow the order given):
. drop if dr==1
. drop dr hv*
. drop if dr==1
(2 observations deleted)
. drop dr hv*
```

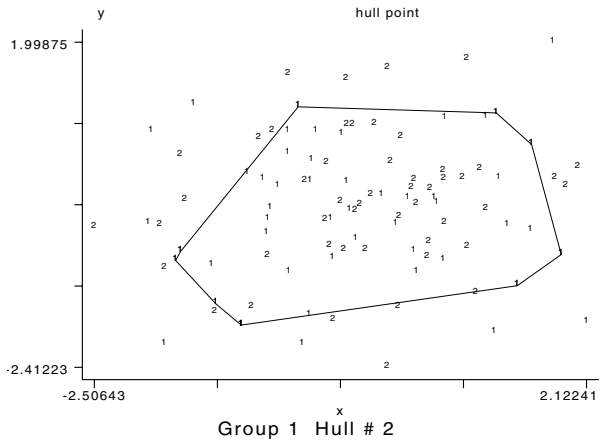

Figure 5

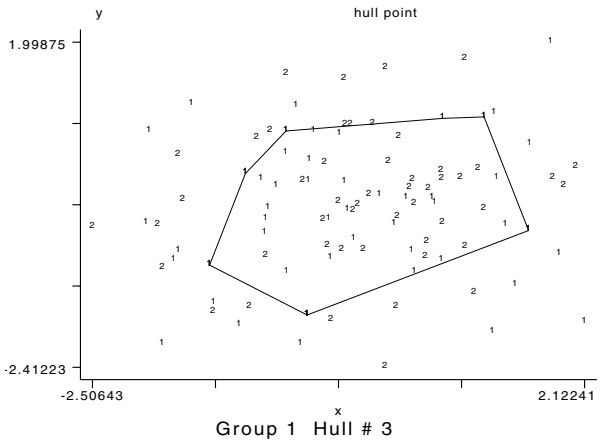

Figure 6

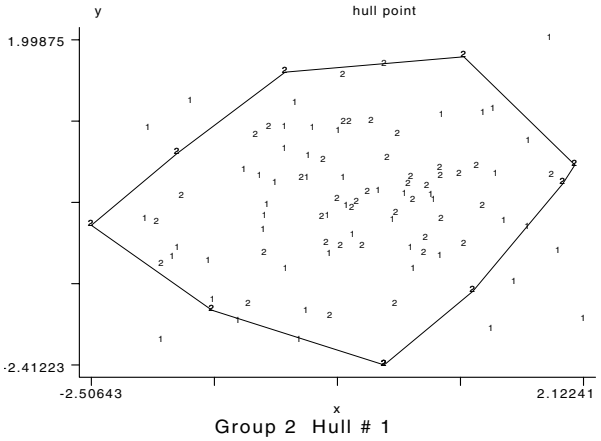

Figure 7

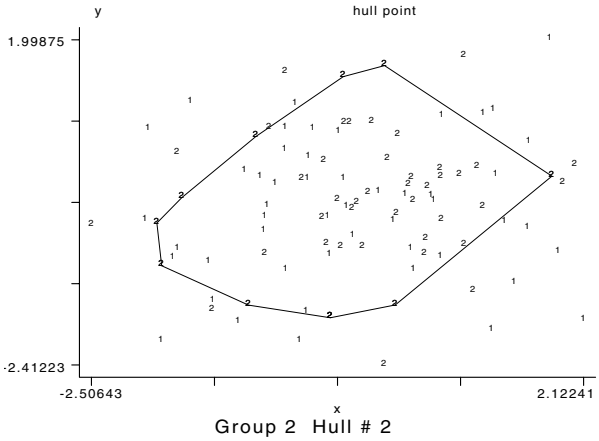

Figure 8

One possible use of the `conhull` program is with bootstrapped data to assess the stability of solutions in correspondence analysis or other scaling techniques (Greenacre 1984, Ringrose 1992). For example, Greenacre provides the following scores for four categories of smoking in a fictitious firm:

Table 1. Scores by smoking category.

	dimension 1	dimension 2
None	0.393	−0.029
Light	−0.098	0.141
Medium	−0.195	0.007
Heavy	−0.293	−0.197

`smoke.dta` contains the points generated by 100 bootstrap replications of Greenacre's Table 3.1.

```
. use smoke,clear
(bootstrap smoking data)
. describe
Contains data from smoke.dta
  Obs:   400 (max=  1076)                 bootstrap smoking data
 Vars:     4 (max=    99)
Width:    11 (max=   200)
  1. x            float  %9.0g             dimension 1 score
  2. y            float  %9.0g             dimension 2 score
  3. conhull      int    %8.0g             convex hull number
  4. smoke        byte   %9.0g    smoke    smoking class
Sorted by:  conhull  smoke
. condraw x y conhull, hull(1) group(smoke)
 (graph appears, see Figure~11)
Variables:  hv1 hv2 hv3 hv4 were created
You can redraw the graph with the command:
graph y hv1 hv2 hv3 hv4 x, c(.llll) s([smoke][smoke][smoke][smoke])
You can now change the graph options, add titles, save, etc.
To use the same data set to generate a different graph or to eliminate
all the created variables use the following sequence of commands
(making sure to follow the order given):
. drop if dr==1
. drop dr hv*
```

The graph shows that all the categories overlap with the exception of "`none`" and "`heavy`". Using the value labels makes identification of the hulls easy, but produces a messy looking graph. We can clean things up and exit the program with the following commands:

```
graph y hv* x,c(.llll) s(....)
```
(graph appears, see Figure˜12)
```
drop if dr==1
(4 observations deleted)
drop dr hv*
```

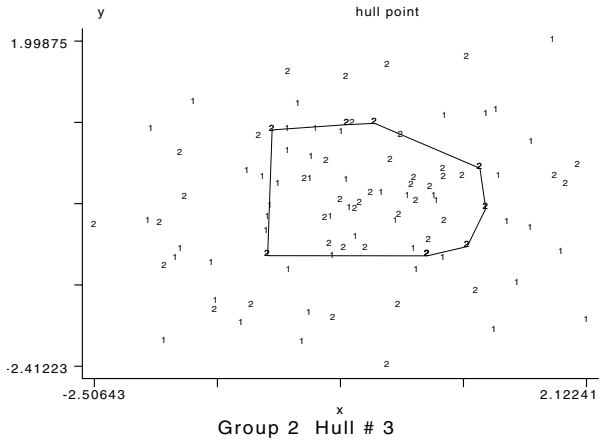

Figure 9

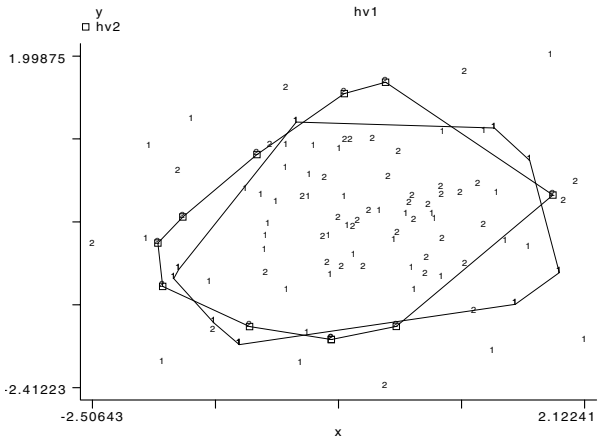

Figure 10

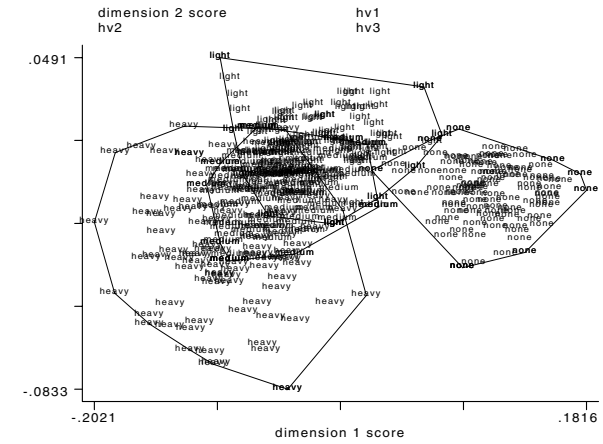

Figure 11

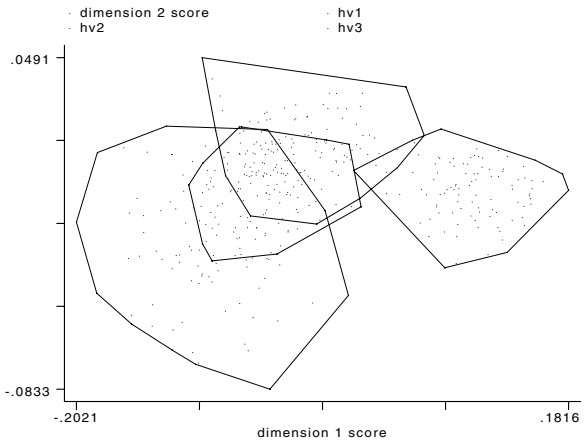

Figure 12

References

Greenacre, Michael J. 1984. *Theory and Applications of Correspondence Analysis*. New York: Academic Press.

Jarvis, R. A. 1973. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters* 2: 18–21.

Ringrose, T. J. 1992. Bootstrapping and correspondence analysis in archaeology. *Journal of Archaeological Science* 19: 615–629.

ip7	A utility for debugging Stata programs

Timothy J. Schmidt, Federal Reserve Bank of Kansas City, FAX 816-881-2199

`findbug` is a utility that assists in the debugging of user-written Stata programs. If the user's program contains any run-time errors, `findbug` displays the offending line and several surrounding program lines, along with Stata's error message and an ordered roster of all programs called before the error occurred. This information allows the user to quickly identify the type of error, the program in which it occurred, and its location within that program.

The syntax is

`findbug` [`-a`] *command*

where *command* invokes the user's Stata program, including any options, qualifiers, or other legal syntax. The Unix-style option `-a` directs `findbug` to include in its report an ordered roster of all programs called prior to the error. `findbug` uses this style of option to distinguish it from any Stata-style options that *command* may include.

Introduction

Users who program a lot in any computer language or software package know that writing the program code is often not the most time-consuming task. The most trying (and often frustrating) work is debugging a program. There are two broad levels of program debugging. What I will call *level I* debugging is concerned with "run-time" errors, that is syntax and other errors that prevent the program from executing to completion. *Level II* debugging is concerned with semantic errors, that is, errors that prevent the program from producing the desired result.

Level II debugging is generally the more difficult type of debugging, and its success is largely dependent upon good program design. Stata provides the `pause` command ([6a] pause) to help with level II debugging. `pause` allows the user to interact with a program in the midst of execution, thereby assisting in the identification of design problems that prevent the program from doing what the user intended. While `findbug` can be used in conjunction with `pause`, `findbug` is intended to assist users in level I debugging, that is, in identifying and eliminating run-time errors.

Problems with trace

Stata already offers the `trace` command ([6a] program) to assist in level I debugging. `trace` allows the user to inspect the flow of a program as it executes, stopping when Stata encounters a run-time error. However, users who program a lot with Stata know that `trace` has a couple of shortcomings.

One difficulty is that the trace of a program can rapidly become very long and unwieldy, making it time consuming to locate errors. This is especially true of programs that are very long, contain a lot of loops, display a lot of information to the screen, or invoke other Stata programs that are implemented as ado-files. The user has three possible ways to use `trace`. Assuming `more` ([6a] more) is set to 0, the user can patiently proceed through the trace one screen at a time until the error is encountered; this method can be quite tedious for long traces. Alternatively, the user may `set more 1` and let the trace rush by on the screen, hoping that the error occurred sufficiently late in the program that the offending line and error message do not scroll off the screen as subsequent code is displayed until the program's end; this method can be unreliable. Finally, the user may direct the trace to a log file ([5u] log) and subsequently examine it in a text editor; this method can be cumbersome and requires manually searching through an often lengthy file.

A second difficulty in using `trace` occurs when debugging programs that invoke other Stata programs implemented as ado-files (external commands). It is not uncommon for programs to incorporate several levels of these nested program calls. For example, the user's program may call `hpfilter` (another user-provided external command (Schmidt 1994)), which in turn calls `_crcnuse` (a Stata-provided external command). When an external command is invoked either directly or indirectly from the user's program, its execution becomes incorporated into the trace. This can make it very difficult for the user to identify the program in which the error occurred.

findbug: an alternative to trace

In essence, `findbug` facilitates the process of debugging Stata programs by overcoming the limitations of `trace`. `findbug` directs a trace of the user's program to a log file and scans the log file for errors that Stata found. If an error was found, `findbug` first displays 12 lines to the screen—the six or seven lines of program code that preceded the error, the line containing the error, Stata's error message or messages, and the three lines of program code that follow the error. `findbug` also displays Stata's error message again (so the user can rapidly spot it) and, optionally, a complete roster of all external programs (Stata- or user-provided ado-files) called either directly or indirectly from the user's program. If no errors were found in the user's program, `findbug` so reports.

Compared to the conventional use of `trace`, the advantages of `findbug` are several. First, `findbug` makes it unnecessary for the user to painstakingly scroll through the trace and locate the error. Second, `findbug` is relatively fast. If an error exists in the user's program, `findbug` locates it rapidly by scanning the log of the trace with a program compiled from source code written in C. Third, `findbug` efficiently presents the user with all of the pertinent information to fix a program error. It provides more information than a conventional trace by displaying the ordered sequence of all external programs called directly or indirectly in the user's program. This information, along with the display of the section of program code containing the error, should allow the user to quickly identify the program in which the error occurred and its location within that program. Fourth, `findbug` stores results for further reference by the user. The log file of the trace is saved in `_findbug.log` in the user's current directory. If an error was found and the user specified the -a option with `findbug`, the sequence of external program calls is saved in a system macro, `S_adoseq`.

Users of `findbug` should be mindful of one caution. The purpose of `findbug` is to efficiently locate and display any run-time errors that Stata finds in the user's program. However, Stata will not report errors that are contained within `capture` blocks ([6a] capture). Therefore, the user should always eliminate `capture` blocks before debugging a program. An efficient way of doing this is to replace all `capture` blocks with `quietly` blocks or `noisily` blocks ([6a] quietly). This method keeps the user's block structures intact, making it easy to change them back to `capture` blocks once all of the errors have been removed.

Example

The following example illustrates the use of `findbug`. Suppose the user is in the process of writing an ado-file to read an ASCII-format data file into Stata, save it as a Stata-format data set, reorganize the data by percentiles, and save the resulting data set under another name. After writing the code, the user names this ado-file "`lng2wide.ado`" and tries running it in Stata by reading the ASCII-format file mtr.asc.

```
. lng2wide mtr
no room to add more variables
no room to add more variables
r(900);
```

In the midst of executing `lng2wide.ado`, Stata halts the program and reports that there is no room to add more variables. The user can look up return code 900 in Stata's manuals to get more information about the error.

However, in order to proceed with debugging, the user has to know where in the program the error occurred. In this admittedly simple example, it may be obvious to the user where the error occurred. However, if the location of the error is not obvious, the user was formerly compelled to `set trace on` and manually page through the program run. This would be rather time consuming since, in this example, `lng2wide.ado` generates a trace that is over 12,000 lines long (`lng2wide.ado` has lots of loops and calls several Stata-provided commands that are implemented as ado-files). This is a job for `findbug`.

```
. findbug -a lng2wide mtr
```

Stata begins to execute `lng2wide.ado` and the trace of the program run passes by on the screen. At the end of the trace, `findbug` clears the screen and displays the following output:

```
- label variable `var´`pct1´ "`pct1´th percentile"
- label variable `var´`pct2´ "`pct2´th percentile"
- label variable `var´`pct3´ "`pct3´th percentile"
- label variable `var´`pct4´ "`pct4´th percentile"
- label variable `var´`pct5´ "`pct5´th percentile"
- if `i´ > 1 {
- merge using `varlist´wide
no room to add more variables
no room to add more variables
  drop _merge
  }
  capture save `varlist´wide, replace
no room to add more variables
Ado-files called (most recent to first):
_crcrsfl _crcrsfl _crcrsmc _crcrsw reshape lng2wide
```

By displaying the program lines surrounding the error and repeating Stata's error message, `findbug` makes it easy to identify the error and its location. In addition, the user can follow the sequence of program execution by examining the ordered roster of program calls. Note that `findbug` is smart—it will even report ado-file calls that are nested several levels within the user's program code. In this example, the user's program, `lng2wide`, calls Stata's `reshape` command ([5d] reshape). `reshape` is implemented as an ado-file so it appears in `findbug`'s ado-file sequence. `findbug` doesn't stop there. `reshape` calls three other Stata-provided programs (`_crcrsfl`, `_crcrsmc`, and `_crcrsw`) and so `findbug` reports those in the order in which they were called.

While `findbug`'s tenacity in identifying all nested program calls may be unnecessary to spot the problem in this simple example, this feature can prove to be very useful in debugging complex, multi-layered programs. In cases where knowledge of the ado-file sequence is unnecessary, the user can suppress that part of `findbug`'s report by calling `findbug` without the `-a` option. `findbug` executes faster when the `-a` option is not specified.

A final note

At the present time, `findbug` is only available for DOS and Sun versions of Stata. I am currently unable to provide a version for the Apple Macintosh because the heart of `findbug` is an executable file compiled from source code I wrote in C, and I do not have access to a C compiler for the Mac. However, if there is enough interest in `findbug`, I will happily seek out a Mac and release a Mac-compatible version in a future issue of the STB. In any case, C source is provided on the STB diskette.

Reference

Schmidt, T. 1994. sts5: Detrending with the Hodrick–Prescott filter. *Stata Technical Bulletin* 17: 22–24.

sg29 | Tabulation of observed/expected ratios and confidence intervals

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

This insert presents `smr`, a program that displays the ratios of observed and expected counts together with confidence intervals for the ratios. `smr` stands for *S*tandardized *M*ortality *R*atio, reflecting the author's interest in the statistical analysis of medical and epidemiological data. And, in fact, epidemiologists are likely to find `smr` particularly useful. Nonetheless, `smr` can be used by anyone interested in computing confidence intervals for ratios of observed and expected counts where the observed counts can be assumed to be distributed as Poisson random variables and the expected counts are known without error.

Syntax

`smr` *obsvar expvar* [`if` *exp*] [`in` *range*] [`,` `icd(`*varname*`)`
`level(#)` `rowlab(`*varname*`)` `sumonly` `total`]

— or —

`smri` $\#_o$ $\#_e$ [`,` `level(#)`]

Description

For each observation, `smr` computes the ratio of *obsvar* over *expvar* together with a confidence interval based on assuming *obsvar* is a Poisson count and the expected value given in *expvar* is known without error. A one-sided confidence interval is provided if the observed count is zero. Ratios that are significant are marked with one asterisk for p-values less than .05, with two asterisks for p-values less than .01, and with three asterisks for p-values less than .001. None of the p-values and confidence intervals are adjusted for the implicit multiple comparisons. p-values are twice the one-sided Poisson probabilities of observing a more extreme count.

`smri` is the immediate form of `smr`; see [4] immediate.

Options

`level` specifies the significance level in percent for the confidence interval.

`icd` is a synonym for `rowlab`. The option `icd` stands for *I*nternational *C*lassification of *D*isease, reflecting the author's interest in the statistical analysis of medical and epidemiological data.

`rowlab(`*varname*`)` specifies a variable that is used to label the rows of the table. *varname* contains row labels for the observed and expected counts. It can be a numeric or a string variable. If it is a noninteger number, it is advisable to read and store the variable as a `double` rather than a `float` to avoid unattractive display. `rowlab` will be ignored if `icd` is specified.

`total` adds the total observed count together with its expectation, the ratio O/E, and its confidence interval to the usual output. `sumonly` specifies that only the totals should be calculated and displayed.

Remarks

Standardized mortality ratios are commonly used in epidemiology to summarize the results of cohort studies. The expected number of events (incidences of a particular condition) is calculated from national vital statistics based on the length of follow-up in each age×calendar-period category. Observed incidences in the cohort (population under study) are compared to these expected incidences to see whether they are unusually large or small. Classic examples of such cohorts include British doctors (studied for the health effects of smoking) and workers in particular factories (studied for various exposures, such as asbestos and radiation).

Under the assumption that individuals are independent (which excludes study of contagious diseases) and have probability of developing a specific condition in a given month that depends only on those factors used to calculate the expected incidence (e.g., age, sex, and calendar period), the total observed count of individuals with the condition is a Poisson random variable. (This follows from results about independent Poisson processes.) Since the expected incidences are derived from census-type statistics, it is often reasonable to treat them as known (that is, free from both stochastic variation and measurement error). Thus, the observed count O is distributed Poisson with mean E, the expected count.

The SMR is just the ratio of the observed to the expected count which may be multiplied by 100 (as in the enclosed ado-file) and is thus a measure of the relative rates after adjusting for age, sex, and calendar period. Exact confidence limits are obtained from the upper (μ_U) and lower (μ_L) limits for the mean of the Poisson variable O; i.e., $\text{SMR}_L = \mu_L/E$ and $\text{SMR}_U = \mu_U/E$. Since the computation of exact Poisson limits is fast in Stata, the program uses this routine even when O is large and the limits could be well-approximated using normal formulae.

The Poisson distribution is often used when events occur randomly in time (in queuing theory, for example) and to describe experiments in which the observed variable is a count. Possible applications of this ado-file outside of epidemiology might include quality control, emission of radioactive particles, and sampling fish populations. The use of SMRs in epidemiology is standard and is explained in many texts. A thorough treatment of the topic can be found in Chapter 2 of Breslow and Day (1987). The review article of Keiding (1987) describes the use of SMRs going back to 1786.

Examples

To illustrate the use of `smr`, we use data on the incidence of second primary cancers in a population all of whom had melanoma (a type of skin cancer). The cohort consists of all individuals in a particular country diagnosed with melanoma during the study period. The observation time is from diagnosis until death, second cancer diagnosis, or the end of the study whichever occurred first. First, we calculate the ratios and confidence intervals for selected cancers for the males in the study.

```
. use eye_demo
. smr m_o m_e if icd<155, icd(icdstr)
                                                  -- Poisson  Exact --
icdstr      | Male Obs   Male Exp       O/E (%)   [95% Conf. Interval]
------------+-----------------------------------------------------------
Lip         |        2    1.4365      139.2            17      503
Tongue      |        0    0.2300        0.0             0     1603+
Salivary    |        0    0.2346        0.0             0     1572+
Mouth       |        0    0.4836        0.0             0      763+
Pharynx     |        0    0.5404        0.0             0      682+
Oesophagus  |        2    1.4304      139.8            17      505
Stomach     |       11    8.6345      127.4            64      228
Sm. intest  |        0    0.3177        0.0             0     1161+
Colon       |        9    8.6651      103.9            47      197
Rectum      |       11    7.3252      150.2            75      269
(+)   one-tail, 97.5% confidence interval
```

It may be of interest to see whether this group of males exhibits an unusual incidence of *any* type of malignant neoplasm (cancer). We use the `sumonly` option to display the information for total incidence and to suppress the individual rows of the table.

```
. smr m_o m_e if icd<999, sumonly
                                                    -- Poisson  Exact --
            | Male Obs   Male Exp       O/E (%)   [95% Conf. Interval]
------------+-------------------------------------------------------------
Total       |      136  111.1966      122.3*          103      145
(*)   Twice 1-sided p<.05
```

As we noted above in the discussion of options, noninteger row labels may display unattractively unless they are read and stored as doubles rather than floats. To illustrate that phenomenon, we stored the ICD information both as a float and a double.

```
. describe icd*
  1. icd          double %10.0g
 13. icd_f        float  %9.0g
 14. icdstr       str19  %19s
. smr a_o a_e in 11/13, icd(icd) total
                                                    -- Poisson  Exact --
icd         | All Obs    All Exp      O/E (%)     [95% Conf. Interval]
------------+-------------------------------------------------------------
155         |        8    1.7665      452.9***        196      892
155.1       |        2    2.4450       81.8            10      295
156         |        4    1.1928      335.3            91      858
------------+-------------------------------------------------------------
Total       |       14    5.4043      259.1**         142      435
(**)  Twice 1-sided p<.01
(***) Twice 1-sided p<.001
. smr a_o a_e in 12, icd(icd_f)
                                                    -- Poisson  Exact --
icd_f       | All Obs    All Exp      O/E (%)     [95% Conf. Interval]
------------+-------------------------------------------------------------
155.10001   |        2    2.4450       81.8            10      295
```

References

Keiding, N. 1987. The method of the expected number of deaths 1786–1886–1986. *International Statistical Review* 55: 1–20.

Breslow, N. E. and N. E. Day. 1987. *Statistical Methods in Cancer Research: Volume II—The Design and Analysis of Cohort Studies*. Lyon: International Agency for Research on Cancer.

sg30	Measures of inequality in Stata

Edward Whitehouse, OECD, Paris, FAX (011)-33-1-45-24-78-52

A large literature has investigated methods of summarizing the degree of inequality in a distribution (see Morris and Preston 1986, Nygard and Sandstrom 1981, Lambert 1989, and the references cited below). The most common application in economics is to the inequality of income; the example of income is used here. The aim of this insert is briefly to describe a number of inequality measures and their implementation in Stata.

The Lorenz curve

The measurement problem is best illustrated with reference to a Lorenz curve (Figure 1) which plots the cumulative proportion of the population, from the poorest upward, against the share of total income they hold. The 45-degree line, drawn for comparison, would be the Lorenz curve for a distribution of incomes that were equal.

If the Lorenz curve of one distribution lies at some point outside (lower than) the Lorenz curve of a second distribution and at no point inside, then the first distribution can be regarded as more unequal than the second. But such a measure of inequality reserves judgment when Lorenz curves cross. A large number of indices of inequality have been proposed to compare pairs of distributions of whose Lorenz curves may cross.

Relative mean deviation

One of the simplest measures of inequality, which shows the extent to which individual incomes, y_i, in a population of size n differ from the mean, $\overline{y}$, is the relative mean deviation (RMD) defined by

$$I_{\text{RMD}} = \frac{1}{2\overline{y}n} \sum_{i=1}^{n} |y_i - \overline{y}|$$

The RMD shows the proportion of income that would need to be transferred from those above mean income to those below mean income to achieve equality. The major fault of the RMD is that it is insensitive to transfers between people on the same side of the mean. It therefore violates the so-called Pigou–Dalton condition (Pigou 1932, Dalton 1920), that a fruitful index of inequality should decrease whenever income is transferred from a richer to a poorer individual.

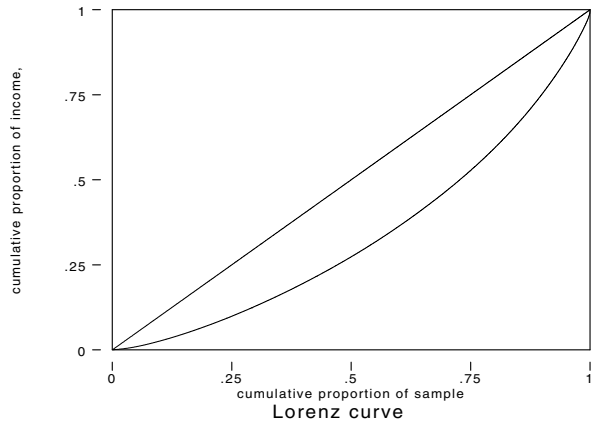

Figure 1

Coefficient of variation

The variance

$$I_{\text{VAR}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})^2$$

is a simple measure that meets the Pigou–Dalton condition, but it has the undesirable property that measured inequality varies with the level of the mean. The variance may be scaled by the mean to give the coefficient of variation (CV) which avoids the problem of mean-dependence:

$$I_{\text{CV}} = \sqrt{I_{\text{VAR}}/\overline{y}}$$

Standard deviation of logarithms

Foster and Shorrocks (1985), among others, have suggested an alternative to the Pigou–Dalton condition which rates income transfers among those with low incomes as more important than transfers between high income individuals. The CV, for example, is affected most by those with high incomes. They propose the standard deviation of logarithms (SDL) as an alternative even though this measure violates the Pigou–Dalton condition. The SDL is defined as

$$I_{\text{SDL}} = \frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log \overline{y}_G)^2$$

where $\overline{y}_G$ is the geometric mean.

Gini coefficient

The Gini coefficient has a simple interpretation since it is based on the Lorenz curve. The Gini coefficient is defined as the ratio of twice the area between the Lorenz curve and the line of absolute equality (the 45-degree line) to the area of the box as a whole,

$$I_{\text{GINI}} = \frac{2}{n^2\overline{y}}\sum_{i=1}^{n} i(y_i - \overline{y})$$

where the y_i are arranged in ascending order. A large number of Lorenz-based measures have been suggested including that of Mehran (1976),

$$I_{\text{MEHRAN}} = \frac{3}{n^3\overline{y}} \sum_{i=1}^{n} i(2n+1-i)(y_i - \overline{y})$$

and that of Piesch (1975),

$$I_{\text{PIESCH}} = \frac{3}{2n^3\overline{y}} \sum_{i=1}^{n} i(i-1)(y_i - \overline{y})$$

The Gini coefficient is a weighted average of the Mehran and Piesch indices, and all three Lorenz-based measures satisfy the Pigou–Dalton condition and lie in the range zero to unity. The Mehran measure is most affected by those on low incomes, the Piesch by the incomes of the richest.

A more general Lorenz-based index, suggested by Donaldson and Weymark (1980, 1983) is the class of "relative S-Ginis", which take the form

$$I_{\text{RELSG}} = \frac{1}{\overline{y}} \left\{ \overline{y} - \frac{1}{n^\delta} \sum_{i=1}^{n} \left[(n-i+1)^\delta - (n-1)^\delta \right] y_i \right\}, \qquad \delta \geq 1$$

where δ is a parameter representing sensitivity to those on low incomes. For $\delta = 1$, the index is always zero; $\delta = 2$ gives the Gini coefficient; $\delta = 3$ gives the Mehran; and, as $\delta \to \infty$, it tends to the "maxi-min" function usually attributed to Rawls (1971) which depends purely on the income of the individual with the lowest income.

Kakwani

The Gini coefficient is most sensitive to income transfers near to the mean (Atkinson 1970, Kakwani 1980). An alternative measure, derived from the length of the Lorenz curve rather than the area bounded by the curve, is more sensitive to income transfers at the extremes. This measure is defined as (Kakwani 1980, Piesch 1975)

$$I_{\text{KAK}} = \frac{1}{2-\sqrt{2}} \left[\left(\frac{1}{n\overline{y}} \sum_{i=1}^{n} \sqrt{y_i^2 + \overline{y}^2} \right) - \sqrt{2} \right]$$

Atkinson

Atkinson (1970) proposes the following measure

$$I_{\text{ATK}}(\epsilon) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i}{\overline{y}} \right)^{(1-\epsilon)} \right)^{1/(1-\epsilon)}$$

$$I_{\text{ATK}}(1) = 1 - \prod_{i=1}^{n} \left(\frac{y_i}{\overline{y}} \right)^{1/n}$$

where ϵ is a parameter showing aversion to inequality. This measure is derived from a restricted form of social welfare function, reflecting the relationship between the current mean income and the equally-distributed income that would generate the same level of welfare. With the parameter $\epsilon = 1$, the index is one minus the ratio of the geometric to the arithmetic mean, and with $\epsilon = 2$ it is one minus the ratio of the harmonic to the arithmetic mean, inequality measures used by Champernowne (1973, 1974). As the parameters of both Atkinson and the relative S-Gini indices increase, both measures focus more on the position of those with the lowest incomes.

Entropy measures

Theil (1969, 1972) presents two measures drawn from information theory

$$I_{\text{THEIL}} = \frac{1}{n} \sum_{i=1}^{n} \frac{y_i}{\overline{y}} \log \frac{y_i}{\overline{y}}$$

$$I_{\text{MLD}} = \frac{1}{n}\sum_{i=1}^{n} \log \frac{\overline{y}}{y_i}$$

where the second of these entropy measures is the mean logarithmic deviation.

Implementation in Stata

The distribution diskette contains four ado-files that calculate the inequality measures discussed above.

`lorenz` *varname* [`if` *exp*] [`in` *range*] [*weight*] [, `offset` *graph_options*]

graphs the Lorenz curve. In this and the three following commands, `fweights` are allowed.

`inequal` *varname* [`if` *exp*] [`in` *range*] [*weight*]

computes a selection of the inequality measures described above: the relative mean deviation, coefficient of variation, standard deviation of logs, Gini, Mehran, Piesch, Kakwani, Theil entropy, and mean log deviation indices. The Atkinson and relative S-Ginis are left to separate commands, since each requires a parameter to be specified.

`atkinson` *varname* [`if` *exp*] [`in` *range*] [*weight*] [, `epsilon(`*parameter-list*`)`]

computes the Atkinson inequality index using the inequality aversion parameter(s) specified in *parameter-list*.

`relsgini` *varname* [`if` *exp*] [`in` *range*] [*weight*] [, `delta(`*parameter-list*`)`]

computes the Donaldson–Weymark relative S-Gini using the distributional sensitivity parameter(s) specified in *parameter-list*.

References

Atkinson, A. 1970. On the measurement of inequality. *Journal of Economic Theory* 2: 244–263.

Champernowne, D. 1973. *The Distribution of Income between Persons*. Cambridge: Cambridge University Press.

——. 1974. A comparison of measures of inequality of income distributions. *Economic Journal* 84: 784–816.

Dalton, H. 1920. The measurement of inequality in incomes. *Economic Journal* 30: 348–361.

Donaldson, D. and J. Weymark. 1980. A single-parameter generalisation of the Gini indices of inequality. *Journal of Economic Theory* 22: 67–86.

——. 1983. Ethically flexible Gini indices for income distributions in the continuum. *Journal of Economic Theory* 29: 353–358.

Foster, J. and A. Shorrocks. 1985. Transfer sensitive inequality measures. University of Essex Economics Discussion Paper no. 264.

Kakwani, N. 1980. *Income Inequality and Poverty*. New York: Oxford University Press.

Lambert, P. 1989. *The Distribution and Redistribution of Income: A Mathematical Analysis*. Cambridge, MA: Basil Blackwell.

Mehran, F. 1976. Linear measures of income inequality. *Econometrica* 44: 801–809.

Morris, N. and I. Preston. 1986. Inequality, poverty and the redistribution of income. *Bulletin of Economic Research* 38: 277–344.

Nygard, F. and A. Sandstrom. 1981. *Measuring Income Inequality*. Stockholm: Almqvist and Wicksell.

Piesch, W. 1975. *Statistische Konzent Rations Masse*. Tubingen: J. C. B. Mohr.

Pigou, A. 1932. *Economics of Welfare*. London: Macmillan.

Rawls J. 1971. *Theory of Justice*. Cambridge, MA: Harvard University Press.

sg31 | Measures of Diversity: Absolute and Relative

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

Statistics measuring diversity or inequality can be absolute or relative. An example of an absolute measure is the standard deviation, which measures, in the units of the original variable, the dispersion or inequality of measurements in a single distribution. Relative measures of diversity relate the dispersion to something else. For example, the coefficient of variation (CV) scales the standard deviation by the mean of the distribution. The Gini coefficient, another relative measure, compares the inequality of a particular distribution to an ideal distribution. (In the context of income distributions, the typical application of the Gini coefficient, this ideal distribution is one where every individual has the same income.)

This insert presents `rspread`, an ado-file that calculates eight different measures of diversity, some absolute and some relative. In addition, `rspread` will optionally display and save a graph of the *Lorenz curve*, a graphical measure of inequality related to the Gini coefficient.

rspread: Eight measures of dispersion or inequality

The syntax for `rspread` is

`rspread` *varlist* [`if` *exp*] [`in` *range*] [, `graph`]

For each variable in the *varlist*, `rspread` calculates eight measures of dispersion or inequality. Several of these measures are essentially robust alternatives to the variance or the standard deviation. If the `graph` option is specified, a Lorenz curve will also be displayed for each variable. The graphs of the Lorenz curves are automatically saved. The curve for the first variable is saved as `lorenz1.gph`, the curve for the second variable is saved as `lorenz2.gph`, and so on. Any previous files with these names are overwritten.

The current version of `rspread` does not allow weights, which can be a drawback with grouped data. You can use the `expand` command ([5d] expand) to overcome this limitation if the data are frequency-weighted, or you can modify `rspread` to accommodate weights.

The eight measures of diversity produced by `rspread` are

1. mean deviation about the mean (MD),
2. mean deviation about the median (AD),
3. mean difference (MeanDif),
4. coefficient of variation (CV),
5. coefficient of dispersion (CD),
6. Gini coefficient,
7. standard error of the mean,
8. maximum percentage deviation from average.

The MD is the average distance between the observations and their average value μ. This measure is also called the mean absolute deviation (MAD). The MD differs from the variance, which is the average squared (rather than average absolute) difference from the mean.

The AD is similar to the MD, but it measures the average distance from the median rather than from the mean.

The mean difference is the average of the differences between all pairs of numbers, i and j, where $i \neq j$. It is equal to $2\mu \times$ (Gini coefficient). One advantage of this statistic is that the distances are not measured from a central value (such as μ) and thus it is "an intrinsic measure of the average distance between the observations" (Gastwirth 1988, 1: 20).

The preceding three statistics are all measures of spread, similar in spirit to the variance.

The coefficient of variation (CV) is equal to the standard deviation divided by the mean, while the coefficient of dispersion (CD) is the ratio of the mean deviation about the median to the median. If one wants the Herfindahl index (a measure of concentration), it is approximately equal to $(1/n)((CV^2) + 1)$.

The Gini coefficient, equal to one-half the ratio of the mean difference to the mean, is a measure of relative inequality or relative variation (as are CV and CD). The Lorenz curve displays this concept graphically.

The preceding three statistics (CV, CD, Gini) are measures of relative variation or inequality, while the first three are measures of absolute variation. Sometimes what is wanted is not the entire Gini curve, but only a particular point on the curve, such as the percent of the population living in the smallest 50 percent of a set of political districts ("the minimum population percentage required to elect a majority"). The relevant point for any cutoff can be obtained by using the `tabulate` command to display a frequency-weighted tabulation of the variable: Find the row for the desired percentile (for example, the 50-th percentile) and read the cumulative frequency in the far right column. An easy way to find the appropriate row is to `summarize` the unweighted variable using the `detail` option and then to use row closest to the desired percentile (see the example below). The `detail` option of the `summarize` command provides the following percentiles: 1, 5, 10, 25, 50, 75, 90, 99. In the example below, the 50 percent of the districts with the smallest populations contain 43.36 percent of the total population.

The standard error of the mean is equal to the standard deviation divided by the square root of n and is the standard deviation of a set of means from, usually hypothetical, repeated samplings. It is used as an estimate of the accuracy of the sample mean as an estimator of the population mean.

The maximum percentage deviation from average shows the biggest percentage difference between a value and the average value. This statistic is primarily useful if you are using a substantive criterion, for instance, "no value should be more than a predetermined percentage from the mean".

Examples using Gastwirth's data

```
. use gastwrth

. rspread pop00 pop60, graph
Measures of Absolute and Relative Dispersion (or Inequality):
        | Mean Dev. about                                                  Max.
variable|    Mean  Median  MeanDif      CV       CD     Gini  SEMean    % Dev.
--------|-----------------------------------------------------------------------
pop00   |  2422.1  2395.9   3389.8  0.1982   0.1673   0.1147  509.65   35.92%
pop60   |   28772   26239    36616  0.5528   0.5809   0.3054  5769.6  120.11%
```

(two graphs appear, see Figures~1 and~2)

```
. summarize pop00, detail
                    Population in 1900
-------------------------------------------------------------
      Percentiles      Smallest
 1%         9466           9466
 5%        10814          10814
10%        10830          10830       Obs                  33
25%        12629          10830       Sum of Wgt.          33
50%        14318                      Mean           14771.27
                        Largest       Std. Dev.       2927.72
75%        16938          18846
90%        18846          19604       Variance        8571543
95%        19654          19654       Skewness       .0649131
99%        19992          19992       Kurtosis       2.022125

. tabulate pop00 [fw=pop00]
  Population|
     in 1900|      Freq.     Percent        Cum.
------------+-----------------------------------
        9466 |       9466        1.94        1.94
       10814 |      10814        2.22        4.16
       10830 |      21660        4.44        8.60
       10992 |      10992        2.25       10.86
       11251 |      11251        2.31       13.17
       11627 |      11627        2.39       15.55
       11677 |      11677        2.40       17.95
       12629 |      12629        2.59       20.54
       12662 |      12662        2.60       23.14
       13320 |      13320        2.73       25.87
       14114 |      14114        2.90       28.76
       14130 |      28260        5.80       34.56
       14269 |      14269        2.93       37.49
       14293 |      14293        2.93       40.42
       14318 |      14318        2.94       43.36
       14566 |      14566        2.99       46.35
       14986 |      14986        3.07       49.42
       15178 |      15178        3.11       52.53
       16645 |      16645        3.41       55.95
       16656 |      33312        6.83       62.78
       16892 |      16892        3.47       66.25
       16938 |      50814       10.42       76.67
       17007 |      17007        3.49       80.16
       18604 |      18604        3.82       83.98
       18846 |      18846        3.87       87.84
       19604 |      19604        4.02       91.87
       19654 |      19654        4.03       95.90
       19992 |      19992        4.10      100.00
------------+-----------------------------------
       Total |     487452      100.00
```

References

Allison, P. D. 1978. Measures of inequality. *American Sociological Review* 43: 865–880. Also see comment by G. Jasso and reply by Allison. 1979. same journal 44: 867–872.

Chandra, M. and N. D. Singpurwalla. 1981. Relationships between some notions which are common to reliability theory and economics. *Mathematics of Operations Research* 6: 113–121.

Gastwirth, J. L. 1988. *Statistical Reasoning in Law and Public Policy* 1: Statistical Concepts and Issues of Fairness. Boston: Academic Press.

Lerman, R. I. and S. Yitzhaki. 1984. A note on the calculation and interpretation of the Gini index. *Economics Letters* 15: 363–368.

Schechtman, E. and S. Yitzhaki. 1987. A measure of association based on Gini's mean difference. *Communications in Statistics–Theory and Methodology* 16: 207–231.

Figures

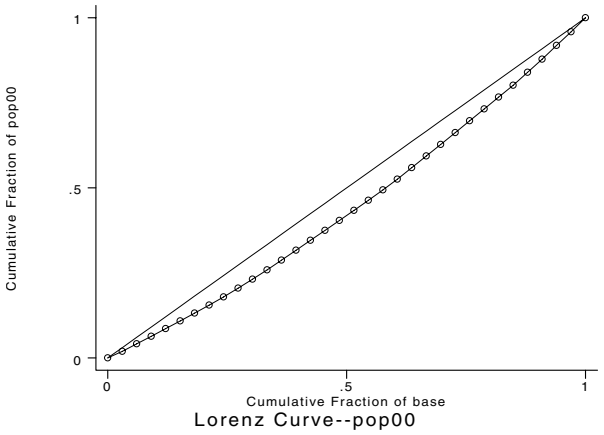

Figure 1

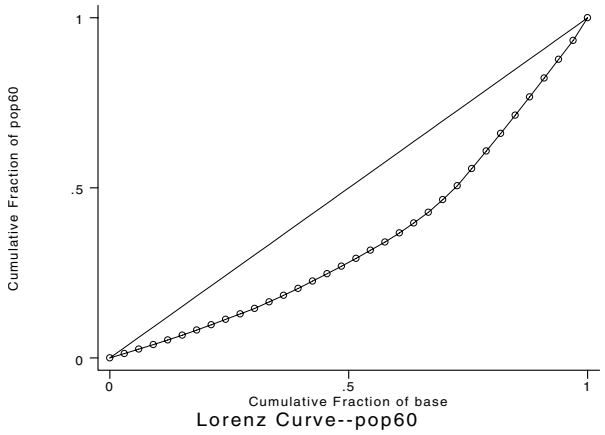

Figure 2

sqv10	Expanded multinomial comparisons

William H. Rogers, Stata Corporation, FAX 409-696-4601

This insert presents `mcross`, an ado-file that conveniently summarizes the comparisons implicit in a multinomial regression model. Multinomial regression results are normalized so one category—one level or outcome of the dependent variable—is the base category, that is one category is compared to all the other categories. You can specify the base category with the `basecat` option to `mlogit`, or you can let the program pick a default, which is the category with the largest sample.

As an example, we use the `mlogit` command to estimate the relationship between headroom (`hdroom`) and the repair record (`rep78`) for the vehicles in the automobile data set supplied with Stata. In this example, the category "`rep78==3`" is chosen as the base category.

```
. use auto
(1978 Automobile Data)
. mlogit rep78 hdroom
Iteration 0:  Log Likelihood =-93.692061
Iteration 1:  Log Likelihood =-88.804257
Iteration 2:  Log Likelihood = -87.53085
Iteration 3:  Log Likelihood =-87.100524
Iteration 4:  Log Likelihood =-86.993118
Iteration 5:  Log Likelihood =-86.986018
Iteration 6:  Log Likelihood =-86.985979
```

```
Multinomial regression                                Number of obs =     69
                                                      chi2(4)       =  13.41
                                                      Prob > chi2   = 0.0094
Log Likelihood = -86.985979                           Pseudo R2     = 0.0716
------------------------------------------------------------------------------
       rep78 |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
-------------+----------------------------------------------------------------
1            |
      hdroom |  -4.863244   2.591125    -1.877   0.061      -9.941756    .2152679
       _cons |   8.046749   4.798254     1.677   0.094      -1.357656    17.45115
-------------+----------------------------------------------------------------
2            |
      hdroom |   .3140054   .4926157     0.637   0.524      -.6515036    1.279514
       _cons |  -2.348883      1.689    -1.391   0.164      -5.659263     .961497
-------------+----------------------------------------------------------------
4            |
      hdroom |  -.2970307   .3723065    -0.798   0.425      -1.026738    .4326766
       _cons |   .4007078   1.172262     0.342   0.732      -1.896883    2.698299
-------------+----------------------------------------------------------------
5            |
      hdroom |  -1.051861   .5110166    -2.058   0.040      -2.053436   -.0502873
       _cons |   1.982806   1.423049     1.393   0.164      -.8063179    4.771931
------------------------------------------------------------------------------
(Outcome rep78==3 is the comparison group)
```

The contrasts of non-base categories can be calculated from these results, but the process is tedious. For instance, to test the contrast for the effect of headroom between categories "5" and "2", we can resort to the specialized syntax Stata uses for multi-equation models:

```
. display = [5]hdroom-[2]hdroom
-1.3658669
. test [5]hdroom=[2]hdroom
 ( 1) - [2]hdroom + [5]hdroom = 0.0
           chi2(  1) =    4.50
         Prob > chi2 =    0.0338
```

Alternatively, we can rerun the `mlogit` command and explicitly specify a different base category. Depending on the size of the dataset, the number of explanatory variables, and the number of categories, this process can quickly become unwieldy.

`mcross` simplifies this process by calculating and displaying the contrasts for each pair of categories. For this example, `mcross` produces the following output:

```
. mcross
------------------------------------------------------------------------------
       rep78 |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
-------------+----------------------------------------------------------------
2-1
      hdroom |   5.177249   2.622833     1.974   0.048       .0365902    10.31791
       _cons |  -10.39563   4.999165    -2.079   0.038      -20.19381    -.597449
-------------+----------------------------------------------------------------
4-1
      hdroom |   4.566213   2.592365     1.761   0.078      -.5147298    9.647156
       _cons |  -7.646041    4.80551    -1.591   0.112      -17.06467    1.772586
-------------+----------------------------------------------------------------
5-1
      hdroom |   3.811382   2.586553     1.474   0.141      -1.258169    8.880934
       _cons |  -6.063942   4.776895    -1.269   0.204      -15.42648    3.298599
-------------+----------------------------------------------------------------
4-2
      hdroom |  -.6110361   .5333629    -1.146   0.252      -1.656408     .434336
       _cons |   2.749591   1.785937     1.540   0.124      -.7507814    6.249963
-------------+----------------------------------------------------------------
5-2
      hdroom |  -1.365867   .6435894    -2.122   0.034      -2.627279   -.1044549
       _cons |   4.331689   1.976218     2.192   0.028       .4583732    8.205005
-------------+----------------------------------------------------------------
5-4
      hdroom |  -.7548308   .5374141    -1.405   0.160      -1.808143    .2984815
       _cons |   1.582099   1.494637     1.059   0.290      -1.347336    4.511534
------------------------------------------------------------------------------
```

Note that `mcross` works just as easily when there are multiple regressors.

The syntax for `mcross` is

`mcross [ , level(#) rrr ]`

As in other estimation commands, the `level()` option specifies the level of the confidence intervals. The `rrr` option indicates that relative risk ratios are to be displayed in place of regression coefficients.

Methods and Formulas

Consider two categories with coefficient vector estimates $\widehat{\beta}_1$ and $\widehat{\beta}_2$, normalized against a third category. The contrast of $\widehat{\beta}_1$ and $\widehat{\beta}_2$ has variance

$$V(\widehat{\beta}_1 - \widehat{\beta}_2) = V(\widehat{\beta}_1) + V(\widehat{\beta}_2) - E[(\widehat{\beta}_1 - \beta_1)(\widehat{\beta}_2 - \beta_2)' - (\widehat{\beta}_2 - \beta_2)(\widehat{\beta}_1 - \beta_1)']$$

Note that the covariance matrix is not symmetric; thus, its transpose must also be considered.

The base category and estimation results from the original `mlogit` are preserved.

Acknowledgment

The design of the `mcross` command was suggested and supported by the center and the staff of the Carolina Population Center, University of North Carolina at Chapel Hill.

sts7.5 A library of time series programs for Stata (Update)

Sean Becketti, Stata Technical Bulletin, FAX 914-533-2902

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

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

New features

This version of the time series library incorporates an improved version of Ken Heinecke's program to calculate Hansen's test for parameter instability (Heinecke 1994b). Hansen's procedure provides a locally most powerful test of the null hypothesis that the parameters of a time series regression are constant (both the coefficients and the variance of the error term) against the alternative hypothesis that the parameters follow a martingale. This alternative is very general: It accommodates parameters that change at unknown times and parameters that follow a random walk. Thus, Hansen's test is more general than the Quandt and Chow tests and locally more powerful than the CUSUM and CUSUM of squares tests.

The syntax of `hansen` is

`hansen` *varlist* [`if` *exp*] [`in` *range*] [, `regress` *tsfit-options*]

See Heinecke (1994b) for a complete description of Hansen's test.

This improved version of `hansen` increases the number of observations that can be accommodated. Cumbersome matrix calculations are required to calculate Hansen's test statistic. Unfortunately, it is not possible to use Stata's very efficient `matrix accum` command to perform these calculations. Instead, Heinecke's original program used `mkmat`, a program that loads variables from the current data set into a matrix, to carry out some of the calculations (Heinecke 1994a). However, storing the raw variables in a matrix limits the number of observations to the `matsize`, the maximum dimension of a Stata matrix. Heinecke has recoded `hansen` so it no longer uses `mkmat`, and thus, this limitation on problem size has been eliminated.

Both the new and the improved versions of `hansen` require `tsfit` and other programs from the time series library. In light of this requirement, Heinecke has graciously donated `hansen` to the library, where it joins programs previously donated by Hakkio (1994a, 1994b). I'd like to take this opportunity to thank Hakkio and Heinecke for their contributions and also the members of the Economic Research department of the Federal Reserve Bank of Kansas City for testing many of the programs in the library. Their critiques have greatly improved the quality and usability of these programs. I regret that I have not yet had time to incorporate all of their excellent suggestions.

References

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

Hakkio, C. 1994a. ip5: A temporary solution to a problem with temporary variable names. *Stata Technical Bulletin* 17: 8–10.

——. 1994b. sts6: Approximate p-valuesfor unit root and cointegration tests. *Stata Technical Bulletin* 17: 25–28.

Hansen, B. E. 1992. Testing for parameter instability in linear models. *Journal of Policy Modeling* 14: 517–533.

Heinecke, K. 1994a. ip6: Storing variables in vectors and matrices. *Stata Technical Bulletin* 20: 8–9.

——. 1994b. sts8: Hansen's test for parameter instability. *Stata Technical Bulletin* 20: 26–32.

zz3.7 Computerized index for the STB replaced in Stata 4.0

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

The capabilities of the interim `stb` command—introduced in Gould (1993)—have been subsumed in Stata 4.0's new `lookup` command. Whereas past inserts in this series have served to update the `stb` command's database, this insert explains how to use `lookup` as a replacement for `stb`.

`lookup` serves as an index into Stata's `help` system. Most importantly,

1. `lookup` performs keyword searches.
2. `lookup` indexes not only the on-line help, but the manuals and the STB as well.
3. `lookup`'s database is automatically updated when you install the official (crc) updates. STB subscribers always have the most recent information (with a one-issue lag) on-line.

Using lookup

As ordinarily used, `lookup`'s most important feature is that it performs keyword searches. You describe the topic using English or statistical terms—you do not need to know the name of the relevant Stata command. For instance, if you wish to find out about the Kolmogorov–Smirnov equality of distributions test, you can type

```
. lookup Kolmogorov-Smirnov equality of distribution test
[5s]    ksmirnov . . . . . Kolmogorov-Smirnov equality of distributions test
        (help ksmirnov)
```

The printed documentation is found in [5s] ksmirnov. On-line help can be obtained by typing '`help ksmirnov`'. It is not necessary to look up "Kolmogorov–Smirnov equality of distribution test"; typing '`lookup kolmogorov smirnov`', '`lookup kolmogorov`' or '`lookup smirnov`' is sufficient.

You can find all the Stata commands relevant to testing the equality of distributions by typing

```
. lookup equality of distribution tests
[5s]    bitest . . . . . . . . . . . . . . . . . . . Binomial probability test
        (help bitest)
[5s]    hotel  . . . . . . . . .  Hotelling´s T-squared generalized means test
        (help hotel)
[5s]    ksmirnov . . . . . Kolmogorov-Smirnov equality of distributions test
        (help ksmirnov)
[5s]    kwallis  . . . . . .  Kruskal-Wallis equality of populations rank test
        (help kwallis)
[5s]    signrank . . . . . . . . . . . . . . . . . . . . . . Sign and rank tests
        (help signrank)
[5s]    survival . . . . . . . . . . . . . . . . . . . . . . Survival analysis
        (help survival)
[5s]    tabulate . . . . . . . . . . .  One- and two-way tables of frequencies
        (help tabulate)
[5s]    ttest  . . . . . . . . . . . . . . . . . . . . . . Mean comparison tests
        (help ttest)
```

`lookup` is not limited to indexing the manuals; if there are relevant entries from the STB, they too are listed. For instance,

```
. lookup normality tests
[5s]    ksmirnov . . . . . . Kolmogorov-Smirnov equality of distributions test
        (help ksmirnov)
[5s]    sdtest . . . . . . . . . . . . . . . . . . . Variance comparison tests
        (help sdtest)
[5s]    sktest . . . . . . . . . . .  Skewness and kurtosis test for normality
        (help sktest)
[5s]    swilk  . . . . .  Shapiro-Wilk and Shapiro-Francia tests for normality
        (help swilk)
STB-5   sg3.7  . . . . . . . . . . . . . . Final summary of tests of normality
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Gould
        1/92    STB Reprints Vol 1, pages 114--115
STB-4   sg3.6  . . . . . . .  Response to sg3.3: comment on tests of normality
        (help swilk) . . . . . . . . . . . . . . . . . . . . . . . . P. Royston
        11/91   STB Reprints Vol 1, pages 112--114
STB-3   sg3.5  . . . . . . .  Comment on sg3.4 and an improved D´Agostino test
        (help sktestdc if installed) . . . . . . . . . . . . . . . P. Royston
        9/91    STB Reprints Vol 1, pages 110--112
STB-3   sg3.4  . . . . . . . . . . . . . . . . . . Summary of tests of normality
        . . . . . . . . . . . . . . . . . . . . . . . W. Gould and W. H. Rogers
        9/91    STB Reprints Vol 1, pages 106--110
STB-3   sg3.3  . . . . . . . . . . . . . . . . . . Comment on tests of normality
        . . . . . . . . R. B. D´Agostino, A. J. Belanger, R. B. D´Agostino Jr.
        9/91    STB Reprints Vol 1, pages 105--106
STB-3   sg3.2  . . . . . . . . . . . .  Shapiro-Wilk and Shapiro-Francia tests
        (help swilk) . . . . . . . . . . . . . . . . . . . . . . . . P. Royston
        9/91    STB Reprints Vol 1, page 105
STB-2   sg3.1  . . . . . . . . . . . . . .  Tests for departure from normality
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . P. Royston
        7/91    STB Reprints Vol 1, pages 101--104
STB-1   sg3  . . . . . . . . . . . . .  Skewness and kurtosis tests of normality
        (help sktestd if installed)  . . . . . . . . . . . . . . . . W. Gould
        5/91    STB Reprints Vol 1, pages 99--101
STB-2   srd2 . . . . . . . . . . . . . . . . . . Test for multivariate normality
        (help multnorm if installed) . . . . . . . . . . . . . . . R. Goldstein
        7/91    STB Reprints Vol 1, page 175
        graph for examining multivariate normality via
        diagnostics from a standard linear regression
```

Special features

For most users, `lookup`'s keyword search capabilities will be all they need, but `lookup` has added features that are of special help to STB subscribers in locating past inserts. `lookup`'s important options in this regard are `stb`, `entry`, `author`, and `historical`.

The `stb` option limits the search to STB inserts.

The `entry` option shifts the search from keywords to entry codes. The entry code of this insert, for instance, is "STB-23 zz3.7". The alternative `author` option shifts the search to author names.

Regardless of the other options you specify, the `historical` option expands the search to include all STB inserts. In order to make `lookup` as helpful as possible to users searching for current features, we have marked some past inserts as being of historical interest only. An insert is declared historical when the features and discussion offered are incorporated in Stata. In the Stata 4.0 distribution, our use of the historical marker is spotty; even if you do not specify the `historical` option, you will see inserts that ought to be marked historical. Over the next year, we will be marking more and more of the past inserts as historical.

In any case, here is how you use the options.

To perform a keyword search on the STB only, include the `stb` option:

```
. lookup normality tests, stb
STB-5   sg3.7  . . . . . . . . . . . . . . Final summary of tests of normality
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . W. Gould
        1/92    STB Reprints Vol 1, pages 114--115
STB-4   sg3.6  . . . . . . .  Response to sg3.3: comment on tests of normality
        (help swilk) . . . . . . . . . . . . . . . . . . . . . . . . P. Royston
```

```
        11/91   STB Reprints Vol 1, pages 112--114
```

(*output omitted*)

To obtain a table of contents for, say, STB-21, look up `stb-21` (capitalization does not matter) and specify the `entry` option:

```
. lookup stb-21, stb entry historical

STB-21  sbe11  . . . . . . . . . . . . . . . . . . . .  Direct standardization
        (help dstndiz if installed)  . . . . . . T. McGuire and J. A. Harrison
        9/94
        predates dstdize

STB-21  sed8 . . . . . .  Finding significant gaps in univariate distributions
        (help wgap if installed) . . . . . . . . . . . . . . . .  R. Goldstein
        9/94
        command that measures gaps in univariate distributions

STB-21  sg26 . . . . . .  Fractional polynomials to model curved relationships
        (help fp if installed) . . . . . . . . . . . P. Royston and D. G. Altman
        9/94
        intermediate method between polynomial and nonlinear models for
        parameterization.  Aim is to keep the advantages of conventional
        polynomials while eliminating (most of) the disadvantages
```

(*output omitted*)

Here, our use of the `historical` option was important. Had we not included it, the *sbe11* insert would not have been listed because the offered feature is now part of Stata.

The `entry` option can also be used to look up related inserts. You can get a list of all inserts in the *sg* series (general statistics), across all issues, by typing

```
. lookup sg, entry stb historical
```

(*output omitted*)

If you left off the `historical` option, the list would not be complete, but it probably would be more useful since it would include only inserts not yet incorporated into Stata.

You can look up all inserts related to sg22 (that is, sg22, sg22.1, etc.) by typing

```
. lookup sg22, entry stb historical
```

(*output omitted*)

You can obtain a list of all inserts written, say, by Gould, by typing

```
. lookup gould, stb author historical
```

(*output omitted*)

In general, keyword searches are the most useful, but sometimes one remembers odd things. If what you remember is that there was this useful insert by Rogers, type

```
. lookup rogers, stb author
```

(*output omitted*)

Finding it, if the insert number is sg8.1 and you are curious what else exists in the sequence, type

```
. lookup sg8, stb entry
```

(*output omitted*)

Remember to add the `historical` option if you want all the inserts.

References

Gould, W. 1993. zz3: Computerized index for the STB. *Stata Technical Bulletin* 16: 27–32.

STB categories and insert codes

Inserts in the STB are presently categorized as follows:

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

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

International Stata Distributors

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

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

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

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

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

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

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