STATA
TECHNICAL
BULLETIN

July 1993
STB-14

A publication to promote communication among Stata users

Editor

Sean Becketti
Stata Technical Bulletin
14619 West 83rd Terrace
Lenexa, Kansas 66215
913-888-5828
913-888-6708 FAX
stb@stata.com EMAIL

Associate Editors

J. Theodore Anagnoson, Cal. State Univ., LA
Richard DeLeon, San Francisco State Univ.
Paul Geiger, USC School of Medicine
Lawrence C. Hamilton, Univ. of New Hampshire
Joseph Hilbe, Arizona State University
Stewart West, Baylor College of Medicine

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

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

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

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

an1.1 STB categories and insert codes

Inserts in the STB are presently categorized as follows:

General Categories:

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

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.

an33 CRC has a new address...and a new name

Alexandria Humphreys, Stata Corporation, FAX 409-696-4601

On August 17, we are moving our offices from Santa Monica, California to College Station, Texas—the location of Texas A&M University. The company name is also changing to Stata Corporation (StataCorp) from Computing Resource Center. In summary, the new address and telephone numbers are

Stata Corporation
702 University Drive East
College Station, Texas 77840
800-STATAPC (800-782-8272)
800-248-8272 (for Canada)
409-696-4600
409-696-4601 (FAX)

Note the new local telephone and FAX numbers, 409-696-4600 and 409-696-4601. Also note that our toll-free, 800 numbers remain unchanged. Southern California residents who used to call our local number should now use the 800 number. Similarly, residents in the 409 area-code area of Texas who used to call our 800 number should now use the new local number.

In terms of the mechanics of the move, the movers will arrive Friday, August 13th to pack the Los Angeles office and deliver it the following Monday, August 16th. Both the Los Angeles and College Station offices will run in parallel the week of August 16th. By the following week, August 23rd, all operations will be transferred to the College Station office.

We will miss being near our friends in Southern California—Rand, UCLA, Cal State University, Los Angeles, and many others—but we are looking forward to close ties with Texas A&M University and having our own building located adjacent to the campus.

an34 Stata 3.1 is available now

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

I am pleased to announce that Stata 3.1 is now available. You will soon receive a *Stata News* that details all of the new features, but the following insert includes this information as well. Ordering information has been enclosed with this issue of the STB. If you are ordering by FAX, please use our new FAX number: 409-696-4601. Orders, whether placed verbally on the 800 line, by FAX, or by mail, will be filled in the order received from our Texas office beginning August 23rd.

an35	A first look at Stata 3.1

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

As the previous insert stated, Stata 3.1 is available now. I have been using a beta-test copy of Stata 3.1 for some months now, and, in this insert, I will give you a brief overview of the new features in this release.

Stata 3.1 significantly expands Stata's capabilities. In fact, I suggested to the staff at StataCorp that this release should be called Stata 4.0 to indicate just how many new features there are. In their typically conservative fashion, they ignored my suggestion, but by the end of this insert, you may lean towards my point of view.

What's really new

The biggest change and the highlight of Stata 3.1 is the new matrix-programming language (MPL). STB readers with incredibly good memories may remember that in *sts2* I suggested linking Stata to an MPL to handle vector autoregressions and other multiple-equation models that Stata could not satisfactorily estimate at that time.

The MPL makes it possible to estimate multiple-equation models, and Stata 3.1 includes new commands for a host of multiple equation and multivariate methods. There are many other new commands as well. I have grouped all these changes below into sections on the MPL, on statistics, on graphics, on data management, and on Stata programming. In closing, I comment on the implications of Stata 3.1 for the STB.

The matrix-programming language

Before the addition of the MPL, some statistical problems could not be solved with Stata alone. Many Stata users, myself included, combined Stata with other matrix software such as Gauss, Matlab, SAS's PROC IML, IMSL, EISPACK, or their own programs to solve these problems. While this approach worked, it was cumbersome and error-prone since data had to be transported between Stata and the other software.

Stata's new MPL eliminates this difficulty completely. In Stata 3.1, matrices and scalars are native data objects and an extensive set of matrix operations are included:

- The standard list of matrix operators is provided including matrix addition, subtraction, multiplication, Kronecker product, transposition, and matrix joining by rows or columns.

- Matrix functions are divided into two types: those that return matrices and those that return scalars. Matrix functions that return matrices include matrix inversion, Cholesky decomposition, sweep, diagonalization and extraction, singular value decomposition, and the calculation of eigenvalues and eigenvectors of symmetric matrices. A function for converting a covariance matrix into a correlation matrix is included as are functions for the creation of identity and zero matrices.

- Matrix functions that return scalars include trace, determinant, and functions that return the number of rows and columns of a matrix.

- Subscripting and element-by-element definition is allowed. Subscripting provides features for easy extraction of submatrices.

Rows and columns are named as well as numbered in Stata's matrix-programming language. While everyone realizes that matrix notation simplifies the description of complex calculations, it is not as obvious that, corresponding to every matrix operation, there is an analogous rule for carrying the names of the rows and columns forward. Stata 3.1 implements these rules so matrix results are correctly labeled no matter how long and complex the sequence of mathematical operations.

Stata also provides a command that, given a coefficient vector and corresponding covariance matrix, produces output that looks like Stata's standard estimation output. Standard errors are obtained from the covariance matrix and significance tests and confidence intervals are automatically constructed. User-programmed estimation results can be posted to Stata's internal storage areas so that they are indistinguishable from Stata's internally generated results. Estimation results can be redisplayed and `predict` and `test` may be used after estimation.

The matrix-programming language also provides an extensive set of routines to assist in maximum-likelihood estimation.

Statistics

The list of new statistical features is very long. In addition to multiple equation and multivariate methods, Stata 3.1 includes commands for estimating nonlinear models by least-squares and by maximum likelihood, commands for new linear and nonlinear regression models, and new commands for miscellaneous statistical procedures.

Multiple equation and multivariate methods

Stata 3.1 has commands to estimate several types of multiple-equation models. The new `eq` command is used to define the equations in these models.

- `mvreg` provides multivariate linear regression and fully integrates its results into Stata's existing `test` command, allowing a full range of cross-equation tests. Between-equation correlations are also reported and tested against zero (the null of independence in the normal case).
- `sureg` estimates multiple-equation models using Zellner's seemingly unrelated equations method. `sureg`'s results are also integrated into the `test` command. Between-equation correlations are reported along with Breusch–Pagan test of independence.
- `hotel` reports Hotelling's T-squared test of the hypothesis that the means of a set of variables are all zero or equal to the means of a second set of variables.
- `heckman` estimates sample-selection models using either Heckman's two-step procedure (with corrected standard errors) or full maximum-likelihood.
- `canon` estimates canonical correlations between two sets of variables and the loadings for these underlying linear combinations. `canon` also provides conditional standard errors for these loadings. Once the canonical correlations are estimated, the linear combinations can be formed using `predict`.
- `hadivmo` provides a technique useful in identifying multiple outliers in multivariate data.

Linear and nonlinear regression models

Stata 3.1 provides three new linear regression commands:

- `areg` estimates a linear regression after absorbing the effect of a set of mutually exclusive and exhaustive dummies. You might run a regression where each observation records the values of the variables for one of the 50 United States. `areg` will absorb the state effect and report the remainder of the regression, that is, it will produce the same result as using `regress` and adding 50 dummy variables, one for each state. The advantage of `areg` is that the dummy variables need never be created, are never included among the explanatory variables and so do not count against the `matsize` limit, and the coefficients and standard errors on the dummies are not reported. The coefficients and standard errors on the remaining variables, the variables of interest, are reported as usual. There is no limit to the number of implied dummy variables that can be absorbed by `areg`. If you want to absorb the family-specific effect of the 4,000 families in your sample, `areg` will do it for you.
- `cnsreg` estimates linear regressions with linear constraints on the coefficients. For instance, `cnsreg` will estimate a regression with the constraint that several of the coefficients sum to 1.
- `eivreg` estimates linear regressions with adjustments for errors in the variables. You provide an estimate of the reliability of the variable or variables measured with error, and adjusted regression results and standard errors are estimated.

Stata 3.1 adds the following nonlinear regression capabilities:

- `poisson`, an existing command for Poisson regression models, now uses improved initial values (Rodríguez 1993) resulting in faster and better convergence.
- `nbreg` and `gnbreg` estimate negative binomial regression models (a gamma mixture of Poisson random variables). `gnbreg` is a generalization of `nbreg` that allows the mixture parameter to be parameterized.
- `hereg` (Rogers 1992b) estimates maximum-likelihood exponential distribution (survival time) models with robust standard errors based on Huber's formula for individual-level data. Consistent standard errors are estimated even if the data are weighted or the residuals are not independently and identically distributed, such as in clustered samples.

- `bsqreg` provides quantile regression estimates with bootstrap standard errors. Rogers (1992a) reported that the method used in Stata 3.0 understates the true standard errors when the errors are heteroscedastic. `bsqreg` abandons the closed form solution for the standard errors, which is appropriate with homoscedastic errors, and uses the bootstrap to produce standard errors which perform well in all cases.

Least squares and maximum-likelihood estimation of nonlinear models

Linear or nonlinear, single or multi-equation models can be estimated in Stata 3.1 by nonlinear least squares or by maximum likelihood:

- `nl` (Royston 1992a) estimates nonlinear models—that is, models that are nonlinear in the parameters—by the method of least squares. The user specifies the criterion function. Precanned nonlinear estimation is provided for the logistic function (symmetric sigmoid shape, not to be confused with logistic regression) and the Gompertz function (nonsymmetric sigmoid shape) (Royston 1993a).
- `ml` estimates nonlinear models by the method of maximum likelihood. The user can specify just the likelihood function; the likelihood function and the gradient vector; or the likelihood function, the gradient vector, and the matrix of second derivatives. When only the likelihood function is specified, a unique linear-form option uses a more efficient, first–derivative-only solution method. This linear-form option is appropriate when the likelihood can be written as $f(\mathbf{X}\beta)$. In this case, `ml` also provides assistance in choosing starting values for the parameters.

Miscellaneous statistical commands

- `centile` reports percentiles and their confidence intervals.
- `pwcorr` displays correlation coefficients with pairwise, rather than casewise, deletion of missing values.
- `brier` reports the Yates, Sanders, and Murphy decomposition of the Brier mean probability score. This decomposition is used for evaluation prediction probabilities when observed outcomes take one of two values.
- `nptrend` (Stepniewska and Altman 1992) performs Cuzick's nonparametric test for trend across ordered groups. This test is an extension of the Wilcoxon rank-sum test. A correction for ties is incorporated.
- `runtest` performs a nonparametric test for serial independence (random order).
- The new `noadjust` option to the `ltable` command suppresses the actuarial adjustment for deaths and censored observations, corresponding to the standard Kaplan–Meier assumption. With this option, `ltable`'s results are consistent with `kapmeier`.
- In Stata 3.0, `logistic`, `logit`, `probit`, and most of Stata's other maximum-likelihood estimation commands report significance levels and confidence intervals based on a pseudo-t distribution. Monte Carlo results reported by Gould in this issue of the STB (1993b) suggest that using the normal asymptotic distribution of the estimates produces nominal significance levels closer to the actual levels. Some commands, such as `weibull`, still incorporate the pseudo-t assumption. This assumption may be revised as more Monte Carlo results become available. For those models now using the normal asymptotic distribution, `test` now reports the χ^2 rather than the F statistic.

Graphics

Stata 3.1 incorporates relatively few changes to the graphics commands:

- The `twoway` and `histogram` graph styles now use value labels, if defined, to label axes.
- `hist` graphs histograms of categorical variables more conveniently than does `graph, histogram`. All bars are automatically labeled and the labels are centered below the bars.
- `cusum` (Royston 1993b) graphs the cumulative sum of a binary (0/1) variable against a continuous variable. This graph is useful in uncovering functional form in, say, a logistic regression model.
- `grmeanby` graphs the means or medians of a variable across groups. These means or medians can be weighted.

Data management

Several important new data management commands have been added to Stata in Version 3.1:

- `ipolate` linearly interpolates and extrapolates a variable to fill in missing observations. A lowess smooth can be used in place of linear interpolation, if desired.
- `ds` displays the variable names in a data set in a compact way. `ds` is a mnemonic for "describe short."
- `lookfor` finds variable names based on substrings that appear in the variable name or variable label. `lookfor` is more flexible than using Stata's wildcards. This command assists in finding the names of variables you may have forgotten.
- `codebook` examines variable names, labels, and values to generate a codebook describing the data set.
- `egen`'s new `group()` function generates a sequentially numbered grouping variable from one or more string or numeric variables. Some Stata commands allow only a single variable name in their `by()` option. Others require the variable(s) to be numeric. The `group()` function makes it easy to meet these requirements.
- `infile`'s new `lrecl()` directive makes it possible to read IBM mainframe-style data sets that have no carriage returns or other markers at the end of lines.
- `reshape` now allows dashes to be specified in the range of numbers of the `reshape values`.

Programming

If you are a regular reader of the STB, you probably skipped directly to this section. There are a number of additions to Stata's kit of programming tools. These additions should make it even easier to extend Stata by writing new `ado`-files.

- `pause` (Becketti 1993) specifies breakpoints in a program to aid in debugging.
- Six new system macros report the date, time, and information on the operating system and type of Stata that is currently running.
- `macro shift` now allows you to specify the number of shifts to perform.
- `macro define` has been renamed `global`. The phrase `macro define` is still understood by Stata 3.1, but you are encouraged to move to the new name. `global` is the more natural antonym for `local`, thus it produces more readable code.
- There is now a `display` extended function for both the `local` and `global` commands. This function makes it possible to place formatted strings into macros.
- There is now a `sortedby` extended function for the `local` and `global` commands. This function gives programs access to the current sort order of the data.
- `word count` and `word of` are extended functions that count the number of words in a string and extract the ith word, respectively.
- `scalar` provides a way to store scalars without using macros. Using Stata's scalars is faster and more accurate than performing the same operations with macros.
- Temporary data sets created by `tempfile` are now automatically erased at the conclusion of a program.
- `preserve` and `restore` protect the current data set from temporary changes during a program. Previously, a program had to use `tempfile` to create a backup copy of the current data set, then it had to restore the data set if the program was interrupted, say by the user pressing the *Break* key. `preserve` handles these functions automatically. `restore` restores the original data set as needed.
- `version` now stores the current version number in `result(1)`.
- `describe` now stores a 1 in `_result(7)` if the data set has changed since it was last saved.
- It is now safe to use `discard` and `program drop` within a program—programs that are currently executing will not be discarded.
- `for` repeats a Stata command for a sequence of variables.

Implications of Stata 3.1 for the STB

All the programs supplied with this issue of the STB are written for Stata 3.0. I will continue to accept submissions written for Version 3.0 for an indefinite period. Since Stata 3.1 will correctly execute any older program that begins with an appropriate `version` command, this policy is not a constraint. Beginning with the next issue of the STB, however, programs written for Stata 3.1 will be included. If you make use of the programs published in the STB—and I certainly hope you do—I recommend that you obtain a copy of Stata 3.1 soon.

References

Becketti, S. 1993. Program debugging command. *Stata Technical Bulletin* 13: 13–14.

Gould, W. W. 1992. Quantile regression with bootstrapped standard errors. *Stata Technical Bulletin* 9: 19–21.

——. 1993a. Graphs of means and medians by categorical variables. *Stata Technical Bulletin* 12: 13.

——. 1993b. Confidence intervals in probit and logit models. *Stata Technical Bulletin* this issue.

Gould, W. W. and A. S. Hadi. 1993. Identifying multivariate outliers. *Stata Technical Bulletin* 11: 28–32.

Rodríguez, G. 1993. An improvement to `poisson`. *Stata Technical Bulletin* 11:11–14.

Rogers, W. H. 1992a. Quantile regression standard errors *Stata Technical Bulletin* 9:16–19.

——. 1992b Huber exponential regression. *Stata Technical Bulletin* 9: 14.

Royston, J. P. 1992a. Nonlinear regression command. *Stata Technical Bulletin* 7: 11–18.

——. 1992b. Centile estimation command. *Stata Technical Bulletin* 8: 12–15.

——. 1993a. Standard nonlinear curve fits. *Stata Technical Bulletin* 11: 17.

——. 1993b. Cusum plots and tests for binary variables. *Stata Technical Bulletin* 12: 16–17.

Stepniewska, K. A. and D. G. Altman. 1992. Non-parametric test for trend across ordered groups. *Stata Technical Bulletin* 9: 21–22.

cc1	Stata chosen for the Medicare program

Charles Chapin, Health Services Advisory Group, FAX 602-241-0757

Many Stata users/STB subscribers may have noticed an increasing number of STB inserts related to health services, Medicare, survival models, hospital billing data, life-tables, etc. This is not surprising considering the recent changes in the Medicare program and that Stata is the statistical package recommended by the Health Care Financing Administration (HCFA) to support their efforts.

Background

As a result of amendments to the Social Security Act in 1982, HCFA was given responsibility for the assurance of the quality of medical care rendered to over 30 million Medicare beneficiaries. This activity is carried out at the local level via approximately 50 Peer Review Organizations (PROs).

Historically, PROs have monitored the appropriateness and quality of the care provided by physicians and hospitals through medical record review of individual encounters. However, HCFA and the PROs are embarking in a new direction that will lead to improved quality of care through statistical analysis and feedback/education. This undertaking, termed the Health Care Quality Improvement Initiative (HCQII), shifts PROs from a focus on individual case-by-case review to analyzing patterns of care and outcome.

Initially, the primary data being used for this endeavor is inpatient claims. In the future, however, it will be possible to conduct longitudinal studies linking an individual's billing data from every health care setting encountered. While paid claim data can supply patient demographics, diagnosis and procedure codes, and prior management information, they do not always adequately provide risk-adjustment or severity of illness information.

In order to compare alternative strategies for the management of particular conditions, clinical differences among patients subjected to different treatments must be taken into consideration. Therefore, in addition to capturing the complete continuum of care that a person receives, the paid claims data eventually will be supplemented by abstracting comprehensive clinical data from the medical record.

Pattern analysis

The feasibility of analyzing the vast amount of Medicare data in a microcomputer environment began in 1991, using Stata 2.1 and, at the time, state-of-the-art technology. The first workstations piloted in Arizona and Connecticut were 80486/33MHz, with

64 megabytes (Mb) of RAM and 5 gigabytes of hard disk storage. Although by most standards these are extremely large specifications, many of the raw files HCFA provided during the pilot project were over 200 Mb when uncompressed. Now this minimal hardware configuration and Stata will be a standard requirement in every PRO.

From the perspective of one of the pilot PROs, the enhancements made in version 3.0 of Stata greatly improved the PROs' ability to manipulate and analyze the large Medicare databases. Moreover, many inserts in the STB have been directly related to HCFA data, including the Bailey–Makeham Survival Models (Rogers 1991; Rogers 1993) and a faster `hpredict.exe` (Krakauer 1993). Perhaps the greatest advantage of using Stata on large data sets is its use of extended memory. Given 64 Mb on a 66 MHz machine, the Arizona PRO can read in over 500,000 observations into Stata and still maintain a respectable number of variables. Without being able to load such a large database into memory, it is hard to imagine the processing time that would be needed if it was necessary to read and write to the hard disk along the way.

In more typical applications, the number of observations that a PRO will work with will be significantly smaller, thereby allowing one the ability to add more variables, up to Stata's generous limit of 2,000. Hopefully, as PROs become more familiar with Stata programming, many ado-files and STB inserts will be developed that can be utilized by other disciplines and Stata users.

[Unix/Intercooled Stata allows a maximum of 2,047 variables. Regular Stata allows 254 variables—Ed.]

References

Krakauer, H. 1993. Faster hpredict.exe. *Stata Technical Bulletin* 12: 14.

Rogers, W. H. 1991. Bailey–Makeham survival model. *Stata Technical Bulletin* 2: 11–14.

——. 1993. Bailey–Makeham survival models. *Stata Technical Bulletin* 12: 14.

dm14	Converting Julian dates to Stata elapsed dates

Charles Chapin, Health Services Advisory Group, FAX 602-241-0757

`jtoe` converts Julian dates to Stata elapsed dates.

The syntax of `jtoe` is

`jtoe` { *year day* | *jdate* } [`if` *exp*] [`in` *range*] , `generate(`*varname*`)` [`noshort`]

Julian dates are dates encoded as the year and the day of the year. For example, the Julian date for January 31, 1993 is 1993:31. Similarly, the Julian date for February 1, 1993 is 1993:32. The day value for December 31 is 365 (366 in leap years). `jtoe` allows the Julian date to be stored in two variables (*year* and *day*) or in a single variable (*jdate*) where the year and day information is combined.

Options

`generate(`*varname*`)` specifies the name of the variable created by `jtoe` and containing the Stata elapsed dates. This "option" is not optional.

`noshort` forces `jtoe` to interpret two-digit years literally, for example, `57` as the year 57 A.D. The default is to interpret two-digit years as a shorthand for the 20th century, that is, `57` is normally interpreted as the year 1957. If any value of the year variable is greater than 99, however, all year values are interpreted literally.

Discussion

Dates are stored in Julian form in a number of data sets. The example with which I am most familiar is the Medicare billing data sent to Peer Review Organizations (PROs) by HCFA. (See *cc1* in this issue of the STB for a discussion of the use of Stata by PROs.) Stata has a reasonably complete complement of commands for encoding and translating dates (see [5d] dates or type `help dates`), but, for some reason, these commands do not handle Julian dates. `jtoe` solves this problem by converting Julian dates to Stata elapsed dates which can then be converted to any other desired form using Stata's existing date commands.

`jtoe` deviates from the conventions of Stata's other date commands by interpreting two-digit years as a shorthand for dates in the 20th century. For example, the value `57` is interpreted as the year 1957. I chose this convention because the data sets I work with typically encode years as two digits. This convention can be overridden by specifying the `noshort` option. Also, years are interpreted literally if any value of the year variable is greater than 99.

Another convenience feature of jtoe is its handling of the several ways of storing Julian dates. Most of the data sets I use store the Julian date as a single five- or seven-digit integer variable, with the first two or four digits reserved for the year and the last three digits for the day of that year. Days of the year less than 100 are zero-filled, for example, January 1, 1993 is encoded as 93001 (or 1993001). Some data sets with a single variable for the Julian date may separate the year and the day portions with a decimal point. In this method, for example, January 1, 1993 is stored as 93.001 (or 1993.001). In still other data sets, however, Julian dates are stored as two separate variables, one for the year and the other for the day of the year. If jtoe is followed by one variable name, it is assumed to contain the combination of the year and the day, either with or without a decimal place separating the year from the day. If jtoe is followed by two variable names, they are assumed to be the variables containing the year and day, respectively.

Example

The following example uses artificial data to demonstrate the use of jtoe. The data set contains six observations of Julian dates, stored in two variables. I create two single variable versions of the dates to demonstrate both syntaxes of jtoe.

```
. use je, clear
. describe
Contains data from je.dta
  Obs:     7 (max= 12250)
 Vars:     2 (max=    99)
Width:     4 (max=   200)
  1. yr            int    %8.0g
  2. dy            int    %8.0g
Sorted by:
. generate long lj = 1000*yr + dy
. generate float fj = yr + (dy/1000)
. list
          yr        dy          lj         fj
1.        52        10       52010      52.01
2.        76       186       76186     76.186
3.        84       266       84266     84.266
4.        92       365       92365     92.365
5.        92       366       92366     92.366
6.        92       367       92367     92.367
```

Note that lj, the integer version of the combined date, must be stored in a long. Now, let's prove that both syntaxes produce the same Stata elapsed dates.

```
. jtoe lj, generate(edate1)
. jtoe fj, generate(edate2)
. jtoe yr dy, generate(edate3)
. list edate*
        edate1      edate2      edate3
1.       -2913       -2913       -2913
2.        6029        6029        6029
3.        9031        9031        9031
4.       12052       12052       12052
5.       12053       12053       12053
6.       12054       12054       12054
```

As a final check, I use Stata's etomdy command to convert the dates to a more readable form.

```
. etomdy edate1, generate(month day year)
. list fj month day year
           fj     month       day      year
1.      52.01         1        10      1952
2.     76.186         7         4      1976
3.     84.266         9        22      1984
4.     92.365        12        30      1992
5.     92.366        12        31      1992
6.     92.367         1         1      1993
```

Note that the "illegal" Julian date in observation 6 is handled sensibly. Now, note that if any of the year values take more than two digits, all year values are interpreted literally, that is, as if the noshort option were specified.

```
. use je2, clear
. jtoe yr dy, generate(edate)
. etomdy edate, generate(month day year)
. list
          yr        dy       edate      year       day     month
  1.      84       266     -684929        84        22         9
  2.    1984       266        9031      1984        22         9
```

dm14.1 Converting Stata elapsed dates to Julian dates

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

In *dm14* above, Charles Chapin provides `jtoe`, an ado-file that converts Julian dates to Stata elapsed dates. For completeness, I have written `etoj`, an ado-file that does the reverse. The syntax is

`etoj` *edate* [`if` *exp*] [`in` *range*] `, generate(`{ *year day* | *jdate* }`)`

The variable *edate* contains Stata elapsed dates. As in `jtoe`, the `generate()` option is not optional; it indicates the names to give the Julian date variables. `etoj` imitates Chapin's `jtoe` in allowing the Julian date to be stored either in two variables (*year date*) or one variable (*jdate*). If one variable is specified, the year forms the integer part of the variable and the day is encoded as the three digits following the decimal point. Unlike `jtoe`, there is no `short` option; years in the 20th century are stored with four digits (e.g., 1957). If the short form is desired, subtract 1900 from *year* (or *jdate*) after using `jtoe`.

Reference

Chapin, C. 1993. Converting Julian dates to Stata elapsed dates. *Stata Technical Bulletin* 14: 3.

os10 A method for taking notes during a Stata session

Bill Rising, Kentucky Medical Review Organization, FAX 502-339-8641

Introduction

I frequently find it useful to take brief notes during a Stata session. When I am examining a new data set or trying out a new Stata program, I often discover problems or come up with ideas for changes. If I don't jot these ideas down immediately, I am likely to forget them.

If I write my notes down on paper, I run the risk of mislaying the paper. On the other hand, I don't want to interrupt my Stata session in order to make a notation in a computer file. It's true that, using the `shell` command (typically abbreviated as `!`), I can edit a file without exiting Stata. For example, in the middle of my session, I can type

```
. ! vi notefile
```

to edit the file `notefile` using the `vi` editor. There are two disadvantages to this procedure though. First, the process can be cumbersome if you use a slow computer or if your editor loads a lot of information when it starts up. Second, the Stata screen disappears when you invoke your editor. If you're trying to make a note of something that just happened, it's easier if the information is still in front of you.

Another solution is to use a windowing system on your computer. If you use Stata under Unix, you can easily open another window to edit in while Stata is still running in the original window. This approach does not always work as smoothly outside of the Unix environment. OS/2 supports multitasking, so this approach will work if you use OS/2. Microsoft Windows, on the other hand, is not a multitasking operating system, so only one window can be active at a time.

To solve this problem until the ideal operating system is available (or until they start giving away Sun computers), I have written two ado-files, `addnote` and `notefile`, that allow you to take brief notes without interrupting your Stata session.

Syntax

The syntax for `addnote` and `notefile` is

`addnote` *text-of-note*

`notefile` [[*path*] *filename*]

`addnote` appends the text that follows the command name to the current notefile. If no notefile has been specified yet, `addnote` creates a plain ASCII file called `notes.not` in the current directory and stores the comment in it. `notefile` changes the current notefile to the filename specified by the user. If no path information is given, the notefile is stored in the current directory. If no extension is specified for the filename, `notefile` adds the extension `.not`.

Example

The following Stata log demonstrates the use of `addnote` and `notefile`.

```
. notefile
No notefile has been specified.  The default is notes.not.
. *
. *  notes.not does not exist yet.
. *
. type notes.not
file notes.not not found
r(601);
. addnote This is the first line in notes.not.
. addnote And this is the second line.
. type notes.not
This is the first line in notes.not.
And this is the second line.
. notefile
The notefile is notes.not in the current directory.
. notefile new
. notefile
The notefile is new.not in the current directory.
. addnote This note goes in the new notefile.
. type new.not
This note goes in the new notefile.
. notefile notes
. addnote Now we are adding lines to notes.not again.
. type notes.not
This is the first line in notes.not.
And this is the second line.
Now we are adding lines to notes.not again.
```

Details

If the current notefile does not have a pathname attached to it, `addnote` and `notefile` assume it is stored in the current subdirectory. As a consequence, if you change subdirectories during your Stata session, `addnote` will add notes to a file with the current notefile name in the new subdirectory. If no such file exists, `addnote` will silently create it.

This behavior is not always a drawback. Many Stata users organize their different projects into different subdirectories. `addnote` is designed to store your notes in the current directory, by default. Thus your notes will usually be stored with the project to which they refer.

os11 A poor-man's 'windowing' environment for Stata

Paul Geiger, University of Southern California School of Medicine, EMAIL pgeiger@mizar.usc.edu

I have found a combination of programs that work with Stata to provide a powerful, flexible environment that doesn't require Microsoft Windows—in effect, I have assembled a kind of poor-man's 'windowing' environment.

The system consists of an AMD 386 40 MHz CPU (Advanced Micro Devices) and Cyrix coprocessor with 8 megabytes of memory and suitable hard disk space. Memory is managed with 386Max (Qualitas, Inc.). A disk caching program called Power Pak (PC-Kwik Corp.) speeds up processing and includes a print spooler, keyboard accelerator, screen review feature, and RAM disk. Control of such niceties as blanking the screen, managing the printer, etc. is at the fingertips. For simplicity, these utilities are run at their default settings which leaves 635,360 bytes of low memory available. The RAM disk is usually set to 1,024K; this setting influences the values reported below.

With the above helper programs installed, regular Stata can handle 21,871 observations (25,803 without the RAM disk), 99 variables (width 200). Shelling out of Stata leaves 258,992 bytes still available. I use the (now old) Norton Commander 3.0 and call it with the *F5* function key (set up when Stata starts and run from the RAM disk). The file finding features and simple editor of the Commander allow easy modification of ado-files during their composing and testing. Other editors such as the one supplied with DOS (I use DOS 5.0 at this writing), the freeware editor, `ted.com`, available on many bulletin boards (Ziffnet, for instance), or StupenDOS (PKWARE, Inc.) work well too.

Under the above conditions, Intercooled Stata handles 18,260 observations but the `shell` to DOS provides about 540K. With this much RAM, larger programs such as WordPerfect run well.

If separate, bare bones `CONFIG.SYS` and `AUTOEXEC.BAT` files are set up using the DOS high-memory manager and invoked with PC-Kwik's Kwikboot feature about 629K of low RAM are available. Intercooled Stata then can handle 32,496 observations but with the loss of the Power Pak features. Shelling to DOS still leaves 540K of low RAM to use.

A really convenient feature of Power Pak is the screen reviewer, PCKSCRN; a press of the space bar saves the screen in a buffer. The size of the file saved is variable. Although the default is 16K, I usually set it to 128K. This setting permits a lot of Stata commands and results to be reviewed on the screen or saved. The buffer can be flushed when desired. I find the screen buffer more convenient than Stata's `log` command when working with data and especially when writing and debugging ado-files. Stata's `#review` command also works nicely in concert with the screen handler. So far no troublesome conflicts among these programs have occurred.

qs5 How to create stacked bar charts of percentages

Felicia Knaul, Departmento Nacional de Plantacion, Republica de Columbia

Q. I am trying to graph a bar chart of four dummy variables on work and school status by age (6–18 years), stacked so that each bar sums to 100 percent. I have been unable to produce this graph so far. The four dummy variables are (1) the child works and attends school, (2) the child works and does not attend school, (3) work status is unreadable and the child attends school, and (4) work status is unreadable and the child does not attend school.

A. Producing a desired bar chart can be a bit tricky sometimes. Let's work through this problem with an artificial data set that resembles the one you've described. We've created a data set with five variables: `age` which takes values from 6–18; `Dws`, a dummy variable equal to one for children who work and go to school; `Dwns`, a dummy variable equal to one for children who work but do not go to school; `Dus`, a dummy variable equal to one for children whose work status is unreadable and who go to school; and `Duns`, a dummy variable equal to one for children whose work status is unreadable and who do not go to school. The data set is already sorted by age, so we can graph a stacked bar chart of the dummy variables by age.

```
. use example, clear
. describe
Contains data from example.dta
  Obs:   100 (max= 12249)
 Vars:     5 (max=    99)
Width:     6 (max=   200)
  1. age          int    %8.0g
  2. Dws          byte   %8.0g            Works, in school
  3. Dwns         byte   %8.0g            Works, not in school
  4. Dus          byte   %8.0g            Unreadable, in school
  5. Duns         byte   %8.0g            Unreadable, not in school
Sorted by:  age
. graph Dws Dwns Dus Duns, bar stack by(age) title(Work and school status by age)
```

This graph is displayed as Figure 1 below. Recall that Stata displays bars that measure the sums of the values of the variables listed. (See [3d] bar for a complete description of Stata's treatment of bar charts.) Thus the components of each stacked bar display the number of times each dummy variable is equal to one for each age. The total height of each bar displays the number of observations for each age.

This is not the desired result. We want each stacked bar to sum to 100 percent and each component to display the percentage of observations at that age for which each dummy variable is equal to one. To produce this graph, we have to create new variables whose sums in each age group are the desired proportions. The following code does the trick.

```
. gen Pws  = Dws
. gen Pwns = Dwns
. gen Pus  = Dus
. gen Puns = Duns
. label variable Pws "Works, in school"
. label variable Pwns "Works, not in school"
. label variable Pus "Unreadable, in school"
. label variable Puns "Unreadable, not in school"
. gen Ptot = Pws + Pwns + Pus + Puns
. qui by age: replace Ptot = sum(Ptot)
. qui by age: replace Pws  = 100*(Pws/Ptot[_N])
. qui by age: replace Pwns = 100*(Pwns/Ptot[_N])
. qui by age: replace Pus  = 100*(Pus/Ptot[_N])
. qui by age: replace Puns = 100*(Puns/Ptot[_N])
```

The P (for percent) variables are matched to the dummy variables. For each value of the age variable, the P variables sum to the desired percentages, thus the four variables Pws, Pwns, Pus, and Puns sum to 100 percent. The variable Ptot counts the number of observations, at each age, for which all four of the dummy variables are non-missing.

```
. graph Pws Pwns Pus Puns, stack bar by(age) title(Work and school status by age)
```

[This command produces the desired chart which is displayed below as Figure 2.—Ed.]

Figures

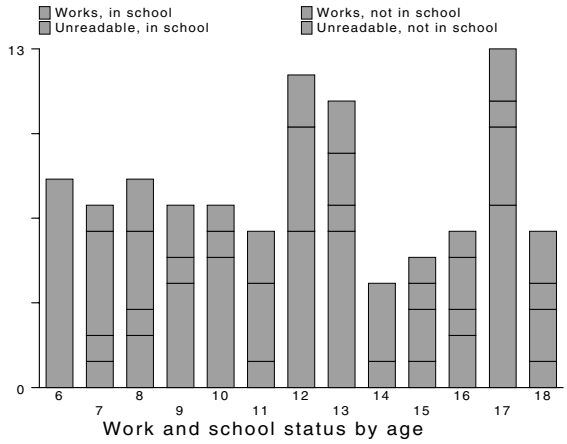

Figure 1

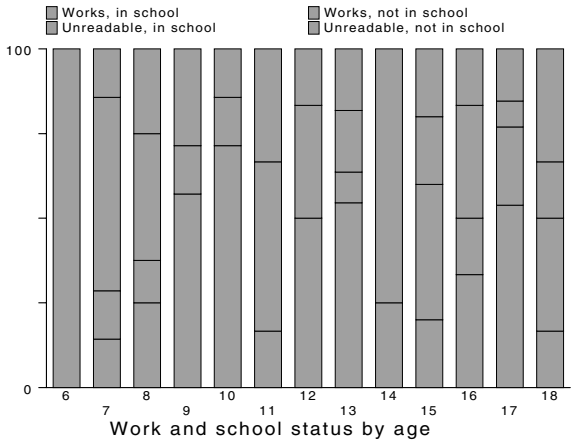

Figure 2

qs6 Query for chemists, physiologists, and pharmacologists

Paul Geiger, University of Southern California School of Medicine, EMAIL pgeiger@mizar.usc.edu

Are there any Stata users who would like to share notes or comments as well as ado-files specifically addressed to the use of Stata in solving problems and handling data in chemistry, biochemistry, physiology, or pharmacology? I believe Stata deserves wider recognition as a tool for the bench scientist manipulating (generally) small samples in such analyses as ELISA, RIA, EISA, chemical equilibria, enzyme kinetics, etc. Please tell your colleagues in departments besides economics, epidemiology, psychology, sociology (they probably already know) about the power of Stata. Contact me by EMAIL at pgeiger@mizar.usc.edu to exchange ideas and notes. You can reach me by mail at

Paul Geiger
Department of Pharmacology and Nutrition, KAM 110,
University of Southern California School of Medicine
1975 Zonal Avenue
Los Angeles, California 90033

qs7 Maximum-likelihood estimation of Gaussian mixtures

Isaias Hazarmabeth Salgado-Ugarte, University of Tokyo, FAX (011)-81-3-3812-0529

Multimodal frequency distributions often arise in biological data, with each mode representing a group of observations sharing the same age, sex, gonadal maturity stage, etc. In the fisheries sciences, the length (size) frequency distribution of fishes and other aquatic organisms is such a polymodal mixture with every mode representing (under certain conditions) a group of fish with the same age. Each age group can be adjusted by a Gaussian, log-normal, exponential, gamma, or Weibull distribution (MacDonald and Green 1988, 1993). Several methods to estimate the parameters—the means, standard deviations and proportions of the modes—have been proposed. Among them are

Graphical or semigraphical methods

Probability paper: Harding (1949); Cassie (1954)
Logarithmic transformation of Gaussian distributions: Tanaka (1962)
Logarithmic differences: Bhattacharya (1967); with some variations to automate it, Pauly and Caddy (1985)

Analytic methods

Moments: Pearson (1915)
Maximum likelihood (with variations): Rao (1948); Hald (1949); Hasselblad (1966); MacDonald and Pitcher (1979); Schnute and Fournier (1980); Akamine (1982, 1984, 1985); Liu, Pitcher, and al-Hossaini (1989); Fournier, Sibert, Majkowski, and Hampton (1990)

Computer programs

ELEFAN IV: Gayanilo, Soriano, and Pauly (1989) using Bhattacharya's method
LFSA: (Length-based Fish Stock Assessment. F. A. O.'s program), Sparre, et. al. (1989), using Bhattacharya's method
MIX: (Ichthus Data Systems) MacDonald and Green (1988)
MULTIFAN: (Otter Research) Fournier, et. al. (1990)

I know that Stata can be programmed to estimate models using maximum likelihood (see [6] maximize in the *Stata Reference Manual*). I am hoping that other Stata users have developed or are interested in developing an ado-file to estimate the Gaussian mixture model via maximum likelihood. Such an ado-file would be useful for a great number of Stata users in biological and industrial fields and those working on any problem concerning mixture distributions. I am willing to collaborate in the writing of such a program.

More details on Gaussian mixture models can be obtained from the references listed below. To further entice potential programmers to help me in this endeavor, the following tables present a worked-out example from the literature. Table 1 displays a well-known data set of length frequencies from Tanaka (1962). Table 2 displays six sets of parameter estimates from these data using six different estimation methods.

If you are interested in working on this problem, you may reach me at

Isaias Hazarmabeth Salgado-Ugarte
Department of Fisheries, Faculty of Agriculture
University of Tokyo
1–1–1 Yayoi, Bunkyo-Ku
Tokyo 113, JAPAN
FAX (011)-81-3-3812-0529

Table 1: Data

midpoint	frequency	midpoint	frequency	midpoint	frequency
7.5	7	17.5	448	27.5	114
8.5	79	18.5	512	28.5	64
9.5	509	19.5	719	29.5	22
10.5	2240	20.5	673	30.5	0
11.5	2341	21.5	445	31.5	2
12.5	623	22.5	341	32.5	2
13.5	476	23.5	310	33.5	0
14.5	1230	24.5	228	34.5	0
15.5	1439	25.5	168	35.5	1
16.5	921	26.5	140	36.5	0

Table 2: Estimates

Parameter	Estimation method	Components				
		1	2	3	4	5
Means	Buchanan-Wollaston	11.05	15.32	19.85	23.58	26.82
	Cassie	11.02	15.33	19.85	23.46	26.92
	Tanaka	10.99	15.26	19.84	23.50	26.82
	Bhattacharya	11.03	15.28	19.86	23.62	26.62
	Akamine	11.0	15.3	19.7	23.5	27.2
	MacDonald & Green	11.00	15.30	19.70	23.45	27.26
Standard deviations	Buchanan-Wollaston	.844	1.161	1.412	1.212	1.443
	Cassie	.76	1.15	1.32	1.29	1.54
	Tanaka	.8	1.2	1.4	1.2	1.4
	Bhattacharya	.81	1.13	1.60	1.07	1.47
	Akamine	.87	1.14	1.43	1.55	1.19
	MacDonald & Green	.83	1.10	1.39	1.62	1.12
Proportions	Buchanan-Wollaston	.4072	.3110	.1860	.0642	.0316
	Cassie	.4049	.3164	.1788	.0693	.0307
	Tanaka	.4007	.3194	.1873	.0598	.0328
	Bhattacharya	.4065	.3067	.2087	.0420	.0361
	Akamine	.411	.305	.183	.077	.024
	MacDonald & Green	.4106	.3056	.1787	.0827	.0224

References

Akamine, T. 1982. A BASIC program to analyze the polymodal frequency distribution into normal distributions. (in Japanese with English abstract), *Bull. Jap. Sea Reg. Fish. Res. Lab.* 33: 163–166.

——. 1984. The BASIC program to analyze the polymodal frequency distribution into normal distributions with Marquardt's Method. (in Japanese with English abstract), *Method. Bull. Jap. Sea Reg. Fish. Res. Lab.* 53: 53–60.

——. 1985. Consideration of the BASIC programs to analyze the polymodal frequency distribution into normal distributions. (in Japanese with English abstract), *Bull. Jap. Sea Reg. Fish. Res. Lab.* 35: 129–160.

Bhattacharya, C. G. 1967. A simple method of resolution of a distribution into Gaussian components. *Biometrics*. 23: 115–135.

Buchanan-Wollaston, H. G. and W. C. Hodgeson. 1929. A new method of treating frequency curves in fishery statistics, with some results. *J. Cons.* 4: 207–225.

Cassie, R. M. 1954. Some uses of probability paper for the graphical analysis of polymodal frequency distributions. *Aust. J. Mar. Freshw. Res.* 5: 513–522.

Fournier, D. A., J. R. Sibert, J. Majkowski, and J. Hampton. 1990. MULTIFAN, a likelihood-based method for estimating growth parameters and age composition from multiple length frequency data sets illustrated using data for Southern bluefin tuna (*Thunnus maccoyii*). *Can. J. Fish. Aquat. Sci.* 47: 301–317.

Gayanilo, F. C., Jr., M. Soriano, and D. Pauly. 1989. A draft guide to the complete ELEFAN. *ICLARM Software*. International Center for Living Aquatic Resources Management, Manila, Philippines, 2: 67.

Hald, A. 1949. Maximum likelihood estimation of the parameters of a normal distribution which is truncated at a known point. *Skand. Aktuarietidskr.* 32: 119–134.

Harding, J. F. 1949. The use of probability paper for the graphical analysis of polymodal frequency distributions. *J. Mar. biol. Ass. U. K.* 28: 141–153.

Hasselblad, V. 1966. Estimation of parameters for a mixture of normal distributions. *Technometrics*. 8: 431–444.

Liu, Q., T. Pitcher, and M. al-Hossaini. 1989. Ageing with fisheries length-frequency data, using information about growth. *J. Fish Biol.* 35: 169–177.

MacDonald, P. D. M. and P. E. J. Green. 1988. User's guide to program MIX: an interactive program for fitting mixtures of distributions. Ichthus Data Systems, Hamilton, Ontario, Canada.

MacDonald, P. D. M. and T. Pitcher. 1979. Age-groups from size-frequency data: a versatile and efficient method of analyzing distribution mixtures. *J. Fish. Res. Bd. Can.* 36: 987–1001.

Pauly, D. and J. F. Caddy. 1985. A modification of Bhattacharya's method for the analysis of mixtures of normal distributions. *FAO Fish. Circ.* 781: 16.

Pearson, K. 1915. On the problem of sexing osteometric material. *Biometrika*. 40: 479–487.

Rao, C. R. 1948. The utilization of multiple measurements in problems of biological classification. *J. R. Statist. Soc. B*. 10: 159–193.

Schnute, J. and D. Fournier. 1980. A new approach to length-frequency analysis: growth structure. *Can. J. Fish. Aquat. Sci.* 37: 1337–1351.

Tanaka, S. 1962. A method of analyzing a polymodal frequency distribution and its application to the length distribution of the porgy, *Taius tumifrons* (T. and S.). *J. Fish. Res. Bd. Can.* 19: 1143–1159.

sg16.2 GLM: A unified power-link based program including the negative binomial

Joseph Hilbe, Dept. of Sociology, Arizona State University, FAX 602-860-1446, EMAIL atjmh@asuvm.inre.asu.edu

This updated `glm` command unifies and substantially enhances the previous `glm` and `glmp` commands (Hilbe 1993a, 1993b) for using generalized linear models (GLM). A partial list of enhancements includes

1. unification of standard and power links;
2. negative binomial models, including linear and quadratic canonical specifications as well as full power-link parameterization (Note: the linear model is not a true GLM);
3. separate `gpredict` command to access predicted values and residuals;
4. standardized Pearson and deviance residuals for all distributions;
5. appropriate default scaling; full scaling options allowed;

6. display of both deviance and χ^2 statistics and dispersion values;
7. bug fixes from the previous version.

Structured upon what I believe to be a more pedagogically sound and computationally efficient algorithm, `glm` is based upon a power-link model. That is, unless one is employing the standard binomial links, or the canonical negative binomial, most models use the power-link algorithm as described in Hilbe (1993b) to calculate estimates and so forth. Minor changes have been made to the command line in order to better effect these enhancements.

Syntax

The syntax for `glm` is

```
glm depvar [cases] varlist [weight] [if exp] [in range] , f({gau|bin|poi|gam|ivg|nb})
l({l|p|c|pow}) p(power) g sc({d|c| |n}) k(#) kl(0|1) o(var) ex(var) eform le(#) it(#)
```

where `f` indicates the error or distribution family, `l` the link, `p` the power, and `sc` the scaling if other than the default. Note the new three-letter code for the inverse Gaussian. Also, there are now only four links; the logit, probit, and cloglog with the binomial and the power link for all distributions. Except for the Gaussian and binomial, canonical links do not require a power specification—but I recommend providing them regardless. Examples are given below which should assist understanding how models are to be called.

The `k(#)` option is used only with negative binomial models, with `k(1)` as the default. The value of `k` is thereby entered into the variance and deviance functions as an additional parameter. Typically, `k(#)` will have values ranging from .01 to 2. One normally uses the negative binomial with overdispersed Poisson models, although the deviance function has been adjusted so that it may be used with a Bernoulli response as well. The following mean/variance relationship may be used to access whether one should use a binomial, Poisson, or negative binomial model:

binomial	mean $>$ variance
Poisson	mean $=$ variance
negative binomial	mean $<$ variance

I have structured the default quadratic specification of the canonical negative binomial so that calling the option `kl(1)` allows the alternative linear parameterization. However, I have found that for most cases the quadratic is more robust. The latter is the re-weight specification provided for all negative binomial power-linked models. The log negative binomial appears optimal for most situations when dealing with Poisson-overdispersed count data.

Output includes χ^2 and scaled deviance summary statistics, their χ^2 significance, and corresponding dispersion values. Gaussian, gamma, and inverse Gaussian standard errors are default scaled by the deviance-based dispersion; however the user may request a χ^2 dispersion scale using the `sc(c)` option or `sc(n)` for no scaling. Binomial, Poisson, and negative binomial models default to a noscale. Scaling may be appropriate for Gaussian and binomial distributions if there is evidence of under or overdispersion. Bernoulli models require no scaling; the `k` value adjusts the dispersion for the negative binomial.

When modeling with negative binomial, one seeks to adjust the value of `k` so that the deviance dispersion is close to 1.0, the significance of the scaled deviance statistic is greater than .05, the estimates significantly enter the model, and the χ^2 dispersion is not too different than that of the deviance dispersion. When these conditions obtain, the model is well-fitted and well-specified. Generalized NB residual statistics have been created to assist the user in further assessing fit and for isolating influential cases or patterns of covariates.

The new `gpredict` command is used in a manner similar to that of `lpredict` following Stata's logistic command. Credit for the implementation of `gpredict` goes to Bill Rogers of StataCorp. However, I must take responsibility for the residuals themselves. All distributions allow the user to obtain the linear predictor (η), the fit (μ), and the following residuals: Pearson, standardized Pearson, deviance, standardized deviance, and hat. The binomial additionally allows the likelihood, Δ-Pearson, Δ-deviance, and Δ-beta residuals. Refer to the `gpredict` help file for additional details.

Summary

The user has a choice of the following distributions and example links:

`gau` = Gaussian; specified with power link

`l(pow) p(1)` = identity (standard OLS linear regression)
`l(pow)` = identity [`p(1)` is default power link]
`l(pow) p(0)` = lognormal

`bin` = binomial, either Bernoulli [0,1] or grouped. The `g` option must be specified to use the grouped version.

`l(l)` = logit link (canonical)
`l(p)` = probit link
`l(c)` = complementary log-log link
`l(pow) p(#)` = binomial power

`poi` = Poisson; log link default (canonical)

`l(pow) p(0)` = alternative canonical log link
`l(pow) p(1)` = identity link, also simply `l(pow)`

`gam` = gamma; inverse link default (canonical)

`l(pow) p(-1)` = alternative canonical inverse link
`l(pow) p(0)` = log link

`ivg` = inverse Gaussian; inverse quadratic link default (canonical)

`l(pow) p(-2)` = alternative canonical link (preferred)
`l(pow) p(0)` = log link

`nb` = negative binomial (Bernoulli or extended count/continuous)

`l(pow) p(0)` = log link (preferred method)

Examples

- Standard linear regression (Gaussian).

```
glm age start numb, f(gau) l(pow)
```

- Logistic regression, showing odds ratios: Bernoulli response.

```
glm kyph age start numb, f(bin) l(l) eform
```

- Complementary loglog, showing exponentiated coefficients, grouped data.

```
glm infec cases cd4 cd8, f(bin) l(c) eform g
```

- Poisson regression, canonical log link showing incidence rate ratios with exposure variable.

```
glm injuries XYZowned, f(poi) ex(n) eform
```

- Poisson regression, identity link.

```
glm age start numb, f(poi) l(pow) p(1)  [or]
glm age start numb, f(poi) l(pow)
```

- Gamma regression, canonical link with frequency weight variable.

```
glm age start [fw=numb], f(gam)
```

- Gamma regression, log link, noscale (noncensored exponential regression).

```
glm age start numb, f(gam) l(pow) p(0) sc(n)
```

- Negative binomial, log link with `k`=1.1.

```
glm age start numb, f(nb) l(pow) p(0) k(1.1)
```

References

Hilbe, J. 1993a. Generalized linear models. *Stata Technical Bulletin* 11:20–28.

——. 1993b. Generalized linear models using power links. *Stata Technical Bulletin* 12:15.

——. 1993c. Generalized linear models: software implementation and the structure of a general power-link based GLM Algorithm. Statistical Computing Section, Royal Statistical Society.

——. 1993d. Negative binomial regression: algorithms and the logic of its Use. C.O.R.E. seminar papers. Institute of Statistics, Catholic University of Louvain, Belgium.

McCullagh, P. and J. A. Nelder. 1989. *Generalized Linear Models*. 2d ed. New York: Chapman & Hall.

Nelder, J. 1993. Generalized linear models with negative binomial or beta-binomial errors. Unpublished manuscript.

sg18	An improved R^2

Patrick Royston, Royal Postgraduate Medical School, London, FAX (011)-44-81-740 3119
EMAIL proyston@rpms.ac.uk
Richard Goldstein, Qualitas Inc., Brighton, MA

The accompanying ado-file `brsq.ado` is designed to provide reliable R^2 (coefficient of determination) statistics to compare the fit of a model in which the response variable is untransformed with that of one in which the response is subjected to a power transformation. Details of the theory are given by Royston (1993). Here, the principle will be motivated by way of examples.

`brsq` *yvar* [*xvars*] [*weight*] [`if` *exp*] [`in` *range*] [, `boxcox(`*Lvar*|*L*`)`
`leave reweight tvars(`*tvars*`)` *other_regression_options*]

calculates deviances, R^2 and adjusted R^2 for two regression models. The models are *yvar* on *xvars* and $f(yvar)$ on *xvars*, where $f()$ is the normalized Box–Cox (power) transformation

$$f(yvar) = \frac{yvar^L - 1}{L\dot{y}^{L-1}} \qquad \text{if } L \neq 0,$$
$$= \dot{y}\ln(yvar) \qquad \text{if } L = 0,$$

and $\dot{y}$ is the geometric mean of *yvar*. If $L = 0$, apart from multiplication by the constant $\dot{y}$, the transformation is logarithmic.

The novel feature is that the values of R^2 are 'scale-corrected' to the scale of *yvar*. This allows for the effect of the transformation on the coefficient of determination for the transformed model and makes meaningful the comparison with R^2 for the untransformed model.

Options

`boxcox(`*Lvar*|*L*`)` is the Box–Cox power parameter. You may use a variable (*Lvar*) or a constant (*L*). Default: constant $L = 0$ (log transformation).

`leave` creates three new variables: `_fy` $= f(yvar)$ as above, and `_fyf` and `_sfy`, the fitted values and standard deviation, respectively, of `_fy` estimated from the regression analysis.

`reweight` computes the weights for the Box–Cox regression to allow for the power transformation, using the formula

$$reweight = weight(\dot{y}/\hat{y})^{2L-2}$$

where $\hat{y}$ is the fit from the (weighted) regression of *yvar* on *xvars*. The default (`noreweight`) forces both models to use the same weights (1, or as given by the *weight* variable). Thus specifying `reweight` uses a set of weights which should be correct for (untransformed) *yvar*, that is, should be proportional to the reciprocal of its variance, and transforms the set appropriately for use with the power transformed *yvar*. Use of `noreweight` compares two regression models *and* two systems of weights (variances) simultaneously.

`tvars(`*tvars*`)` allows you to specify a different set of regressors, *tvars*, for $f(yvar)$. There is in general no reason to expect the model for *yvar* to be satisfactory for $f(yvar)$ also.

other_regression_options are any of the standard options available with the `regress` command.

First example

Using the Stata example file `auto.dta`, suppose we wish to compare a model for `mpg` as a quadratic function of automobile weight with a similar model which uses log(`mpg`):

```
. use auto
(1978 Automobile Data)
. gen w = weight/1000 /* scales weight to more reasonable values */
. gen w2 = w^2
. brsq mpg w w2
An improved R-squared
---------------------
Model: Y-var = mpg, X-vars = w w2
                 Untransformed Y-var    Transformed Y-var
                 -------------------    -----------------
Log-likelihood        -193.125               41.683
Deviance-0             468.789                9.725
Deviance               386.250              -83.365
Deviance-Scaled        386.250              364.266
R-squared-Scaled        0.6722               0.7565
Adj-R-sq-Scaled         0.6630               0.7496
Residual-SD-Scaled      3.3587               2.8951
Note: same weights used in untransformed and transformed analyses
```

`Deviance-0` is the deviance (twice the negative log likelihood) for a model with just a constant and `Deviance` is that for the full model. `Deviance-scaled` and `Residual-SD-scaled` are the deviance and the residual SD adjusted to allow for the transformation of *yvar* (Royston 1993).

```
. brsq mpg w w2, reweight
 An improved R-squared
---------------------
Model: Y-var = mpg, X-vars = w w2
                 Untransformed Y-var    Transformed Y-var
                 -------------------    -----------------
Log-likelihood        -193.125               35.193
Deviance-0             468.789               14.678
Deviance               386.250              -70.386
Deviance-Scaled        386.250              377.245
R-squared-Scaled        0.6722               0.7098
Adj-R-sq-Scaled         0.6630               0.7016
Residual-SD-Scaled      3.3587               3.1604
Note: transformed weights used in transformed analysis
```

The scaled R^2 is highest for the transformed model without reweighting. Assuming that the quadratic model is an adequate fit both on the original and on the log scale, this implies that (a) the log transformation is useful (it appears to counter skewness in the distribution of `mpg`) and (b) the variance is more constant on the log scale than on the original scale.

Note that the unadjusted and adjusted values of `R-squared-scaled` for the untransformed model (0.6722 and 0.6630) are identical to the R^2 values given by the `regress` command. However (and this is the whole point of 'An improved R^2') the corresponding values for the transformed model without reweighting (0.7565 and 0.7496) *differ* from those obtained by regressing log(`mpg`) on `w` and `w2`. In this example, the latter values (0.7158 and 0.7078) are lower.

Second example

Scott and Wild (1991) gave an example comprising an artificial data set with just six (x, y) pairs: (0, 0.1), (3, 0.4), (8, 2), (13, 10), (16, 15), (20, 16). They used it to illustrate the problems of comparing models when the response variable is transformed. Using a flavour of R^2 known as R_1^2 (Kvålseth 1985), they concluded that "the value of R_1^2 for this model [log y on x] is $-.21$, indicating an absolutely terrible fit on the original scale." (In fact, the correct value of R_1^2 is -0.316, not -0.21, but this error obviously would not have altered their conclusion.)

The results of using `brsq` with this data set are as follows:

```
. use scott2
(Scott&Wild, TAS, 1991, 45:127-9)
. brsq y x
An improved R-squared
---------------------
Model: Y-var = y, X-vars = x
                 Untransformed Y-var    Transformed Y-var
                 -------------------    -----------------
Log-likelihood          -12.295               -4.209
Deviance-0               39.863               24.914
Deviance                 24.590                8.418
Deviance-Scaled          24.590               18.933
R-squared-Scaled         0.9216               0.9694
Adj-R-sq-Scaled          0.9020               0.9618
Residual-SD-Scaled       2.3001               1.4356
Note: same weights used in untransformed and transformed analyses
```

Scaled R^2 for the second model is 0.9694, rather different from $R_1^2 = -0.316$! What has happened? The log transformation seems to have simultaneously stabilized the variance and improved the fit, reducing the scaled deviance markedly (by 5.66, in fact). The improvement is reflected in scaled R^2, whereas the value of $R_1^2 = -.316$, which is a crude index of fit on the original scale, is hugely influenced by the apparently poor fit of the log model at $x = 20$. In the log scale, this point fits OK; if it is omitted from the analysis, R_1^2 jumps to 0.804, whereas scaled R^2 increases from 0.969 to 0.996.

If the `reweight` option is used, the log-transformed model is less successful:

```
. brsq y x, reweight
An improved R-squared
---------------------
Model: Y-var = y, X-vars = x
                 Untransformed Y-var    Transformed Y-var
                 -------------------    -----------------
Log-likelihood          -12.295               -6.006
Deviance-0               39.863               18.933
Deviance                 24.590               12.011
Deviance-Scaled          24.590               22.526
R-squared-Scaled         0.9216               0.9444
Adj-R-sq-Scaled          0.9020               0.9305
Residual-SD-Scaled       2.3001               1.9367
Note: transformed weights used in transformed analysis
```

Note that `R-squared-Scaled` for transformed *yvar* (0.9444) is lower than before (0.9694), implying that use of `reweight` has worsened the fit. The results confirm the impression that the variance of y as a function of x is more constant on the log scale than on the original scale. If the opposite had been true, that is, if the variance of y had been independent of x, we would have expected `reweight` to have improved the fit rather than to have worsened it, since `reweight` constructs weights which are equivalent on a log scale to those supplied by the user as appropriate for the untransformed scale of y.

For comparison, we show some of the output from applying Goldstein's (1992) `logsumm.ado` program to the Scott and Wild data:

```
. logsumm y x
Some Summary Statistics:
             |             Adjusted               Better    Standard
             |    Raw        Raw        Log     Adj´d Log  Adj´d Log
-------------------------------------------------------------------------
R-Square     |   0.9216     0.9283     0.9360     0.5040     0.1355
Adjusted R-SQ|   0.9020     0.9104     0.9200     0.3800    -0.0807
F-Value      |  47.00      51.79      58.53       4.06       0.63
RMSE         |   2.3001     0.6327     0.5977    14.3074     9.4229
CV (*100)    |  31.73      72.21      68.21      63.81     392.31
Kvalseth´s R1-type R-squared for some of the models:
                 0.9216                          -2.0349    -0.3164
```

In the above table, `RMSE` stands for 'root mean square error' (i.e. residual SD) and 'CV' is the coefficient of variation (residual SD divided by the mean of y). The first column shows the standard statistics for the regression of y on x. The second column shows summary statistics for the same model but this time adjusted by transforming to logs. The third column gives the values from the regression of $\log y$ on x. These are followed by two sets of adjusted statistics: `Better Adj´d Log`, a less biased

back-transformation than simple exponentiation, and `Standard Adj´d Log`, the 'standard', biased, back-transformation obtained by just exponentiating the predicted values from the log model.

Note that the `Better Adj´d Log` method still gives a lowish R^2 (0.5040), and that the value (-2.0349) of Kvålseth's R_1^2 is even more peculiar with this method than before. Also, the `RMSE` values from `logsumm` are useless when it comes to comparing model fits, whereas the values of `Residual-SD-scaled` from `brsq` make sense.

The conclusion from this example is that scaled-R^2 and adjusted scaled-R^2 give a more useful comparison of the fits of the two models than the `Better Adj´d Log` statistics.

References

Goldstein, R. 1992. Adjusted summary statistics for logarithmic regression. *Stata Technical Bulletin* 5: 17–21.

Kvålseth, T. O. 1985. Cautionary note about R^2. *The American Statistician* 39: 279–285.

Royston, P. 1993. A better R^2. *The American Statistician*, submitted for publication.

Scott, A. and Wild, C. 1991. Transformations and R^2. *The American Statistician* 45: 127–129.

snp5	The run test for random order

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

`runtest` tests whether observations in a sequence are serially independent—that is, whether they occur in a random order—by counting how many runs there are above and below a threshold. By default, the median is used as the threshold. A very small number of runs indicates positive serial correlation. A very large number of runs indicates negative serial correlation.

The syntax of `runtest` is

`runtest` *varname* [`in` *range*] [`, continuity drop mean split threshold(#)`]

Discussion

`runtest` performs a nonparametric test of the hypothesis that the observations of *varname* occur in a random order by counting how many runs there are above and below a threshold—by default, the median. If *varname* is positively serially correlated, it will tend to remain above or below its median for several observations in a row, that is, there will be relatively few runs. If, on the other hand, *varname* is negatively serially correlated, observations above the median will tend to be followed by observations below the median and vice versa, that is, there will be relatively many runs.

The median is not always the best choice for a threshold. If the `mean` option is specified, the mean is used instead of the median. To specify an arbitrary threshold, use the `threshold(#)` option. Since `runtest` divides the data into two states—above and below the threshold—it is appropriate for data which are already binary, for example: win or lose, live or die, rich or poor, etc. Such variables are often coded as zero (`0`) for one state and as one (`1`) for the other state. In this case, specify `threshold(0)`, since, by default, `runtest` separates the observations into those that are greater than the threshold and those that are less than *or equal* to the threshold.

As with almost all nonparametric procedures, the treatment of ties complicates the test. For the run test, an observation that is exactly equal to the threshold value is a tie. There are three possible treatments. By default, such an observation is treated as though it is below the threshold. If the `drop` option is specified, such an observation is dropped and the total number of observations is adjusted. If the `split` option is specified, ties are randomly assigned to the above-threshold and below-threshold groups. If this procedure is chosen, repeating the test with the same data may give a different answer each time.

`runtest` begins by calculating the number of observations below the threshold, n_0, the number of observations above the threshold, n_1, the total number of observations, $N = n_0 + n_1$, and the number of runs, r. These statistics are always reported so the exact tables of critical values in Swed and Eisenhart (1943) may be consulted if necessary.

For larger samples, the number of runs is asymptotically normal. The expected number of runs under the null is

$$\mu_r = \frac{2n_0n_1}{N} + 1$$

the variance is

$$\sigma_r^2 = \frac{2n_0n_1\left(2n_0n_1 - N\right)}{N^2\left(N-1\right)}$$

and the normal approximation test statistic is

$$\widehat{z} = \frac{r - \mu_r}{\sigma_r}$$

Options

`continuity` specifies a continuity correction that may be helpful in small samples. However, if there are fewer than ten observations either above or below the threshold, the tables in Swed and Eisenhart (1943) provide more reliable critical values. By default, no continuity correction is used.

`drop` directs the program to ignore any values of *varname* that are equal to the threshold value when counting runs and tabulating observations. By default, `runtest` counts a value as above the threshold when it is strictly above the threshold and as below the threshold when it is less than or equal to the threshold.

`mean` directs the program to tabulate runs above and below the mean rather than the median.

`split` directs the program to randomly split values of *varname* that are equal to the threshold. In other words, when *varname* is equal to threshold, a "coin" is flipped. If it comes up heads, the value is counted as above the threshold. If it comes up tails, the value is counted as below the threshold.

`threshold(#)` specifies an arbitrary threshold to use in counting runs. For example, if *varname* has already been coded as a 0/1 variable, the median generally will not be a meaningful separating value.

Example: Testing for serial correlation

We can use `runtest` to check regression residuals for serial correlation.

```
. list
         year    resid
  1.     1975     2.95
  2.     1976     1.08
  3.     1977    -1.07
  4.     1978    -2.69
  5.     1979     0.64
  6.     1980     0.75
  7.     1981     2.20
  8.     1982     2.12
  9.     1983    -0.40
 10.     1984    -2.37
 11.     1985    -2.31
 12.     1986    -1.32
 13.     1987    -0.80
 14.     1988    -1.39
 15.     1989     0.99
 16.     1990     1.56
. graph resid year, c(l) yli(0) ylab xlab title(Regression residuals)
```

(See Figure 1)

The graph gives the impression that these residuals are positively correlated. Excursions above or below zero—the natural threshold for regression residuals—tend to last for several observations. `runtest` can evaluate the statistical significance of this impression.

```
. runtest resid, thresh(0)
 N(resid <= 0) = 8
 N(resid >  0) = 8
         obs = 16
    N(runs) = 5
         z  = -2.06
      P>|z| = .03
```

There are five runs in these sixteen observations. Using the normal approximation to the true distribution of the number of runs, the five runs in this series are fewer than would be expected if the residuals were serially independent. The p-value is 0.03, indicating a significant test at the 5 percent level. This p-value is for the two-sided test. (Note the `P>|z|` in the listing.) If the alternative hypothesis is positive serial correlation, rather than any deviation from randomness, then the p-value is approximately 0.015. However, with so few observations, the normal approximation may be inaccurate. The tables compiled by Swed and Eisenhart give five runs as the 5 percent critical value for a one-tailed test.

`runtest` is a nonparametric test. It ignores the magnitudes of the observations and notes only whether the values are above or below the threshold. We can demonstrate this feature by reducing the information about the regression residuals in this example to a 0/1 variable that indicates only whether a residual is positive or negative.

```
. generate byte sign = resid>0
. runtest sign, thresh(0)
 N(sign <= 0) = 8
 N(sign >  0) = 8
         obs = 16
    N(runs) = 5
         z  = -2.06
      P>|z| = .03
```

As expected, `runtest` produces the same answer as before.

Example: A two-sample test for equality of distribution

The run test can also be used to test the null hypothesis that two samples are drawn from the same underlying distribution. The run test is sensitive to differences in the shapes, as well as the locations, of the empirical distributions.

Suppose, for example, that two different additives are added to the oil in 10 different cars during an oil change. The cars are run until a viscosity test determines that another oil change is needed, and the number of miles traveled between oil changes is recorded. The data are

```
. list
     additive     miles
  1.        1      4024
  2.        1      4756
  3.        1      7993
  4.        1      5025
  5.        1      4188
  6.        2      3007
  7.        2      1988
  8.        2      1051
  9.        2      4478
 10.        2      4232
```

To test whether the additives generate different distributions of miles between oil changes, sort the data by `miles`, then use `runtest` to see whether the marker for each additive occurs in random order:

```
. sort miles
. list
     additive     miles
  1.        2      1051
  2.        2      1988
  3.        2      3007
  4.        1      4024
  5.        1      4188
  6.        2      4232
  7.        2      4478
  8.        1      4756
  9.        1      5025
 10.        1      7993
. runtest additive, thresh(1)
 N(additive <= 1) = 5
 N(additive >  1) = 5
         obs = 10
    N(runs) = 4
         z  = -1.34
      P>|z| = .18
```

In this example, the additives do not produce statistically different results.

Example: The runs up-and-down test

A test that is related to the run test is the runs up-and-down test. In the latter test, the data are classified not as to whether they lie above or below a threshold, but as to whether they are steadily increasing or decreasing. Thus an unbroken string of increases in the variable of interest is counted as one run, as is an unbroken string of decreases. According to Madansky (1988), the run test is superior to the runs up-and-down test for detecting trends in the data, but the runs up-and-down test is superior for detecting autocorrelation.

`runtest` can be used to perform a runs up-and-down test. Using the regression residuals from the example above, we can perform a `runtest` on their first differences:

```
. generate D.resid = resid - resid[_n-1]
. list
         year    resid  D.resid
  1.     1975     2.95        .
  2.     1976     1.08    -1.87
  3.     1977    -1.07    -2.15
  4.     1978    -2.69    -1.62
  5.     1979     0.64     3.33
  6.     1980     0.75     0.10
  7.     1981     2.20     1.46
  8.     1982     2.12    -0.08
  9.     1983    -0.40    -2.52
 10.     1984    -2.37    -1.97
 11.     1985    -2.31     0.06
 12.     1986    -1.32     0.99
 13.     1987    -0.80     0.52
 14.     1988    -1.39    -0.59
 15.     1989     0.99     2.38
 16.     1990     1.56     0.57
. runtest D.resid, thresh(0)
 N(D.resid <= 0) = 7
 N(D.resid >  0) = 8
         obs = 15
    N(runs) = 6
         z  = -1.32
      P>|z| = .18
```

Edgington (1961) has compiled a table of the small sample distribution of the runs up-and-down statistic, and this table is reprinted in Madansky (1988). For large samples, the z-statistic reported by `runtest` is incorrect for the runs up-and-down test. Let N be the number of observations (15 in this example) and r be the number of runs (6). The expected number of runs in the runs up-and-down test is

$$\mu_r = \frac{2N-1}{3}$$

the variance is

$$\sigma_r^2 = \frac{16N-29}{90}$$

and the correct z-statistic is

$$\widehat{z} = \frac{r-\mu_r}{\sigma_r}$$

Technical note: The treatment of missing values

`runtest` will tolerate missing values at the beginning or end of a series, as in the example above. However, `runtest` will exit with an error if there are any missing observations in the interior of the series (in the portion covered by the `in` *range* modifier). If you wish to perform the test anyway, simply `drop` the missing observations before calling `runtest`.

Saved Results

`runtest` saves in the `S_#` macros:

`S_1`	number of observations	`S_5`	expected number of runs
`S_2`	number of runs	`S_6`	variance of the number of runs
`S_3`	number below the threshold	`S_7`	z-statistic
`S_4`	number above the threshold	`S_8`	p-value of z

References

Edgington, E. S. 1961. Probability table for number of runs of signs of first differences in ordered series. *Journal of the American Statistical Association*. 56: 156–159.

Madansky A. 1988. *Prescriptions for Working Statisticians*. New York: Springer-Verlag.

Swed, F. S. and C. Eisenhart. 1943. Tables for testing randomness of grouping in a sequence of alternatives. *Annals of Mathematical Statistics*. 14: 83–86.

Figure

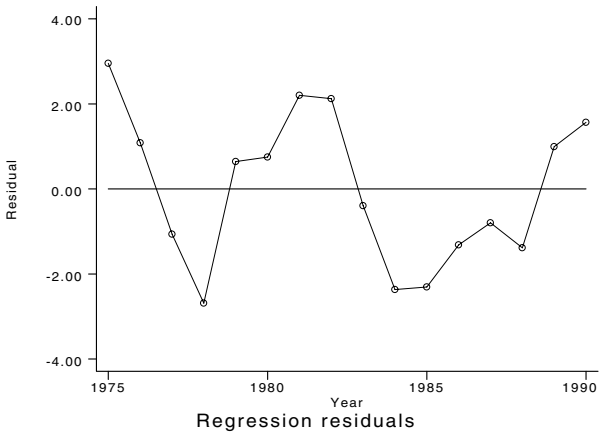

Figure 1

ssi4	Confidence intervals in logit and probit models

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

Stata 3.0 calculates significance levels and confidence intervals for coefficients in logit and probit models using the t distribution even though all that is known about such coefficients is that they are asymptotically normally distributed. That is, given an estimated coefficient b and standard error s, Stata assumes the ratio $r = b/s$ is distributed $t(n-k)$, where n is the number of observations and k the number of estimated parameters including the intercept.

For instance, given estimates $b = 2$ with $s = 1$ in a 2-parameter model estimated on 50 observations, the ratio $r = 2/1 = 2$ is reported as a t with significance level `tprob(50-2,2)` = .0512. The 95% confidence interval is obtained as $b \pm t_{.95}s$ where $t_{.95}$ = `invt(50-2,.95)` = 2.011, resulting in the range −.011 to 4.011. The alternative normal-based statistics would treat r as $\mathrm{N}(0,1)$ resulting in significance level `-2*normprob(-2)` = .0455 and confidence interval $b \pm z_{.95}s$ where $z_{.95}$ = `invnorm(1-(1-.95)/2)` = 1.96, or the range .04 to 3.96.

There is no formal justification for the substitution of t for the normal. One does feel, however, that confidence intervals should be wider in finite samples and one does know, for instance, that linear regression estimates are also asymptotically normally distributed and that, in finite samples, the distribution is t.

Stata is not alone in that feeling. Richard Goldstein once did a tally of statistical packages by whether they use the normal or t in logit models and obtained roughly a 50/50 split. Stata is in good company in that, at least, BMDP also uses the t distribution.

Good company, however, is not adequate justification. David Hosmer recently sent the following fax to us:

> By comparing Stata output to other packages I recently discovered that Stata is not computing the p-values and confidence intervals for estimated coefficients and odds ratios correctly in the logistic and logit procedures. These commands are using the t-distribution with degrees of freedom equal to $n-p$, where p is the number of estimated parameters in the model. This is of course correct for normal errors models but is incorrect for other member of the GLM family. In non-normal errors models one should use the standard normal distribution as results about the distributions are obtained from central limit theorem type arguments and not ratios of random variables which generate the t-distribution.

Upon receipt of this fax, I set about establishing that Stata's use of the t was better in finite samples. My results suggest the opposite.

Begin by considering the model $z = \alpha + \beta_1 x$ with $\alpha = .2$ and $\beta_1 = .4$ and letting x taking on values between 0 and 1. I concocted a 25-observation data set based on these assumptions:

```
. set obs 25
. gen x = _n/25
. gen z = .2 + .4*x
```

In the case of probit, the probability of observing a positive outcome is $F(z)$, where $F()$ is the normal distribution function. Thus, the probability can be generated as:

```
. gen p = normprob(z)
```

In the case of logit, the probability of observing a positive outcome is $e^z/(1+e^z)$, so alternatively, the probability could be generated as

```
. gen p = exp(z)/(1+exp(z))
```

In either case, one can now generate a pseudo-random data set drawn from this population

```
. gen y = p>=uniform()
```

and then estimate the appropriate probit or logit model:

```
. probit y x
—or—
. logit y x
```

Reported in the results will be the 95% confidence intervals for the coefficients. One can thus write down whether the true coefficients are inside or outside of this interval and then one can repeat the process:

```
. replace y = p>=uniform()
. probit y x
—or—
. logit y x
```

This is what I did, repeating the experiment 20,000 times for each model. The results were

	probit		logit	
	t	z	t	z
α	96.96	95.77	97.87	96.62
β_1	96.74	95.48	97.61	96.26

I am surprised at the results: in this experiment with probits and logits estimated on 25-observation data sets, the t intervals are too wide and so, as a matter of fact, are the normal intervals. (An exact binomial 95% confidence intervals for the percent accepted is 94.695% to 95.3%; even the probability of accepting 95.48%—the smallest number in the table—or more of the 20,000 samples has probability .000752.)

I then repeated this experiment with a three-parameter model: $z = \alpha + \beta_1 x_1 + \beta_2 x_2$. The results are

	probit		logit	
	t	z	t	z
α	97.36	95.99	97.82	96.51
β_1	97.30	95.72	97.51	96.24
β_2	98.33	97.83	97.88	96.50

Finally, to verify that asymptotics do eventually take hold, I repeated both experiments using data sets with 100 observations:

	probit		logit	
	t	z	t	z
α	95.61	95.34	95.56	95.35
β_1	95.49	95.15	95.46	95.16
α	95.08	94.82	95.19	94.95
β_1	95.15	94.79	95.29	94.98
β_2	95.18	94.92	95.57	95.31

In small samples, asymptotic confidence intervals for z and t are wider than necessary, leading to conservative tests. When the sample size is 100, both z and t have close to 95% acceptance probability, but z is still a little better than t.

The bottom line is that Hosmer is right: Stata should use the normal, not the t, for significance levels and confidence intervals. Widening the confidence intervals—as is done in linear regression—is not necessary for probit and logit models. (In theory, this result should not surprise you. The widening of the confidence intervals in linear regression is not really due to the finite sample size, it is due to the substitution of an estimate of the variance of the residuals for the true variance. In probit and logit, the variance is not estimated because it is unidentified. Instead the variance is normalized to 1.)

In the next release of Stata, normal significance levels and confidence intervals are substituted for the t-based statistics currently reported.

The programs and log of execution for the results reported above are provided on the STB diskette.