Computing Murphy–Topel-corrected variances in a heckprobit model with endogeneity

Article *in* Stata Journal · January 2010

Source: RePEc

CITATIONS
8

READS
423

3 authors, including:

Juan Muro
University of Alcalá
43 PUBLICATIONS **192** CITATIONS
SEE PROFILE

Cristina Suárez
University of Alcalá
29 PUBLICATIONS **141** CITATIONS
SEE PROFILE

The Stata Journal

Volume 10 Number 2 2010

A Stata Press publication
StataCorp LP
College Station, Texas

The Stata Journal

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

For more information on the *Stata Journal*, including information for authors, see the web page

http://www.stata-journal.com

The *Stata Journal* is indexed and abstracted in the following:

- CompuMath Citation Index®
- Current Contents/Social and Behavioral Sciences®
- RePEc: Research Papers in Economics
- Science Citation Index Expanded (also known as SciSearch®)
- Social Sciences Citation Index®

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press. Stata, Mata, NetCourse, and Stata Press are registered trademarks of StataCorp LP.

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates

The listed subscription rates include both a printed and an electronic copy unless otherwise mentioned.

Subscriptions mailed to U.S. and Canadian addresses:

3-year subscription	$195
2-year subscription	$135
1-year subscription	$ 69
1-year student subscription	$ 42
1-year university library subscription	$ 89
1-year institutional subscription	$195

Subscriptions mailed to other countries:

3-year subscription	$285
2-year subscription	$195
1-year subscription	$ 99
3-year subscription (electronic only)	$185
1-year student subscription	$ 69
1-year university library subscription	$119
1-year institutional subscription	$225

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

Volume 10 Number 2 2010

The Stata Journal

The Stata Journal (2010)
10, Number 2, pp. 165–199

Resampling variance estimation for complex survey data

Stanislav Kolenikov
University of Missouri
Columbia, MO
kolenikovs@missouri.edu

Abstract. In this article, I discuss the main approaches to resampling variance estimation in complex survey data: balanced repeated replication, the jackknife, and the bootstrap. Balanced repeated replication and the jackknife are implemented in the Stata `svy` suite. The bootstrap for complex survey data is implemented by the `bsweights` command. I describe this command and provide working examples.

Editors' note. This article was submitted and accepted before the new `svy bootstrap` prefix was made available in the Stata 11.1 update. The variance estimation method implemented in the new `svy bootstrap` prefix is equivalent to the one in `bs4rw`. The only real difference is syntax. For example,

```
. bs4rw, rw(bw*): logistic highbp height weight age female [pw=finalwgt]
```

is equivalent to

```
. svyset [pw=finalwgt], vce(bootstrap) bsrweight(bw*)
. svy: logistic highbp height weight age female
```

Similarly, the example using mean bootstrap replicate weights,

```
. local mean2fay = 1-sqrt(1/10)
. svyset [pw=finalwgt], vce(brr) brrweight(bw*) fay(`mean2fay´)
. svy: logistic highbp height weight age female
```

is equivalent to

```
. svyset [pw=finalwgt], vce(bootstrap) bsrweight(bw*) bsn(10)
. svy: logistic highbp height weight age female
```

The weights created by the `bsweights` command discussed in this article are equally applicable with the `bs4rw` command and with the new `vce(bootstrap)` and `bsrweight()` options of `svy` and `svyset`.

Keywords: st0187, bsweights, balanced repeated replication, balanced bootstrap, bootstrap, complex survey data, Hadamard matrix, half-samples, jackknife, resampling, weighted bootstrap, mean bootstrap

1 Complex survey data

Researchers who study large-scale health, behavioral, and economic processes often have to deal with datasets collected via complex survey designs. To achieve balance between

costs and statistical accuracy, data collection in large-scale surveys is organized using specialized sampling techniques: stratification, clustering, multiple stages of selection, unequal probabilities of selection, and sampling with or without replacement, to name a few. The estimation procedures must be adapted to these complex survey design features.

In stratified samples, the population is split into nonoverlapping parts, called strata, before any of the sampling steps are taken. Stratification criteria might be geography and urbanicity in areal samples, land use in natural-resource surveys, or industry and size of the firm in establishment surveys. Survey-sample designers use stratification to improve efficiency, protect against badly unbalanced samples, and optimize the total cost of the survey. Stratification also allows the user to conduct straightforward statistical analysis within strata and implement different sampling techniques in different strata.

Cluster samples allow the user to reduce costs in situations where it is impossible or impractical to obtain the complete list of ultimate observation units. Instead, the sample designer obtains a much shorter list of clusters, or primary sampling units (PSUs). Once the required number of PSUs is sampled, more detailed lists are collected for these PSUs. The procedure of taking clusters of units may be repeated at several levels, resulting in multiple stages of selection. Large-scale human-population surveys usually feature between two and four stages of selection.

Sampling weights are necessary to ensure unbiased estimation. In their basic form, sampling weights are inverse probabilities of selection, but additional modifications of sampling weights are often performed. Poststratification uses the existing census or population information, such as the number of people in a group defined by age category, gender, and race. The weights are modified so that the weighted totals of poststratification variables match the population totals. To compute nonresponse adjustments, the weight is further increased by the inverse probability of response. This probability can be estimated 1) by the ratio of the weighted totals of the sampled and responded units within nonresponse adjustment cells or 2) by response propensity obtained from logistic regression of the response indicator on the available unit characteristics.

When simple random sampling is performed without replacement, there are efficiency gains summarized by finite population corrections: $\text{FPC} = 1 - f$, where $f = n/N$ is the sampling fraction, n is the sample size, and N is the population size. If units are selected with varying probabilities, joint selection probabilities need to be used to compute standard errors of (design-consistent) estimators. Hence, the effects of nonnegligible sampling fractions and associated efficiency gains, if any, are reflected in the joint selection probabilities. Because of their special relevance to simple random sampling designs only, and also because the population sizes are either not known or protected for privacy reasons, finite population corrections are rarely available in large-scale public-use datasets.

What happens if the complex survey design features are ignored in statistical analysis? If stratification or finite population corrections are ignored, the standard errors will be conservative (too large), the confidence intervals will be too long, and their coverage will exceed the nominal 95% level. While the positive bias of the standard errors leads to

a loss of power, it can generally be considered acceptable. If clustering is ignored, then the standard errors will be too small and reported results will be claimed significant too often. If sampling weights are ignored, then the sampling distributions of unweighted statistics underrepresent the values of the random variables associated with low selection probabilities and overrepresent the values associated with high selection probabilities. As a result, unweighted statistics are biased for population parameters they estimate. The effects of clustering and unequal weights are detrimental for statistical inference and so analysts and researchers need to account for them.

For a very complex survey design, exact accounting for all its features is extremely cumbersome. At the data analysis stage, approximations are often made to yield a usable estimation formula. The most common approximate design is stratified two-stage sampling with replacement (S2SWR). In this design, the population is divided into strata. From each stratum, a sample of PSUs is taken with replacement, and from each PSU, samples of ultimate units are taken. If the same PSU was sampled more than once, then independent samples within this PSU are taken and marked in the dataset as distinct. Unequal probabilities of selection may be used. The S2SWR approximation allows for relatively simple computations of point estimates (using weights only) and variances (using weights, stratification, and PSU information only).

An example of the S2SWR design is the Second National Health and Nutrition Examination Survey (NHANES II) data, which is used throughout the *Survey Data Reference Manual* ([SVY]).

```
. use http://www.stata-press.com/data/r11/nhanes2
. svyset
      pweight: finalwgt
          VCE: linearized
  Single unit: missing
     Strata 1: strata
         SU 1: psu
        FPC 1: <zero>
```

Here the design is specified as two stages. In the first stage, the stratification variable is `strata` and the PSU/cluster-level variable is `psu`. The second-stage units are assumed to be individual observations. The sampling-weight variable is `finalwgt`. There are no finite population corrections available for this dataset. By default, the variances of the parameter estimates will be computed using the linearization method (`VCE: linearized`). The statement `Single unit: missing` specifies that the variances will be reported as missing when only one PSU is available in some strata. In this dataset, PSUs are numbered 1 and 2 in each stratum, and in certain data manipulations, it is crucial to keep track of both strata and PSU identifiers.

Let us run a benchmark analysis of high blood pressure with individual covariates. This example is also provided as example 2 in [SVY] **svy estimation**.

```
. svy: logistic highbp height weight age female
(running logistic on estimation sample)
Survey: Logistic regression
Number of strata   =        31                 Number of obs      =      10351
Number of PSUs     =        62                 Population size    =  117157513
                                               Design df          =         31
                                               F(   4,     28)    =     178.69
                                               Prob > F           =     0.0000
```

highbp	Odds Ratio	Linearized Std. Err.	t	P>|t|	[95% Conf.	Interval]
height	.9688567	.0056822	-5.39	0.000	.9573369	.9805151
weight	1.052489	.0032829	16.40	0.000	1.045814	1.059205
age	1.050473	.0024816	20.84	0.000	1.045424	1.055547
female	.7250087	.0641188	-3.64	0.001	.605353	.8683158

Compared with typical output of non-`svy` commands, basic design information is added. In the upper left block, the number of PSUs and the number of strata are shown. The difference of the two is the design degrees of freedom, which is the greatest number of explanatory variables that can be used in a regression model. The design degrees of freedom is reported in the upper right block. The population size is estimated by the sum of weights. The column of standard errors has a heading indicating the type of standard errors used (linearized). Other components of the output are the same as in nonsurvey estimation. See [U] **20 Estimation and postestimation commands**.

To conclude this section, I will mention a few references on survey statistics. A popular introductory textbook is Lohr (2010). More formal treatment is given in classic monographs of Kish (1995) and Cochran (1977). The utmost level of mathematical detail can be found in Särndal, Swensson, and Wretman (1992) and Thompson (1997). Great intermediate literature with extensive conceptual explanations are Korn and Graubard (1995) and Lehtonen and Pahkinen (2004). Advanced topics are discussed in the collected volumes of Skinner, Holt, and Smith (1989) and Chambers and Skinner (2003). References that combine conceptual explanations with detailed software examples include Lumley (2010), which covers R, and Heeringa, West, and Berglund (2010), which covers Stata.

2 Variance estimation in complex surveys

Survey statistics has developed inference paradigms that are quite different from the mainstream "let the data be independent and identically distributed (i.i.d.) from a distribution, $f_\theta(x)$, characterized by a vector of parameters, θ." In its purest form of design-based inference, the units in the population, as well as their characteristics (measured variables), are assumed fixed. The randomness in the sample-based statistics (e.g., totals, ratios, means, regression coefficients) comes only from randomization performed at the sample selection stage. The design-based distributions are obtained by enumerating all samples possible under a given design scheme and associating the numeric values of the statistics of interest with the probabilities of the samples they

are based on. Other paradigms, such as model-based (Binder and Roberts 2003) and model-assisted (Särndal, Swensson, and Wretman 1992) inference, work with estimators that have good properties under a strict design-based paradigm and under a working model assumed by the survey statistician.

In this article, I discuss estimation of the design variances, i.e., the variances of the distributions generated by repeated sampling using a given design. In many practical situations, the design-based variance estimates are doubly robust: they are consistent when either the design is correctly described (typically, with replacement at the first stage) or the model is approximately correct (e.g., it is a linear or generalized linear model with a correctly specified mean structure). They are also robust to different specifications of the variance parameters in a model.

Variance estimation in complex surveys serves two goals. First, applied researchers need standard errors to test their hypotheses of substantive interest and construct (Wald) tests and confidence intervals. Second, sample designers use variance estimates to gauge performance of existing designs and choose design parameters for future surveys of similar populations.

There are several variance estimation methods commonly used with complex survey data. We shall talk about direct variance estimation and linearization in this section, and then turn to resampling methods in the next section.

Consider the following estimate of the total for stratified samples:

$$t_{\text{str}}(x) = \sum_{h=1}^{L} \sum_{i=1}^{n_h} t_{hi}$$

where

$$t_{hi} = \sum_{j \in \text{PSU}_{hi}} w_{hij} x_{hij}$$

is the estimate of the total based on the ith PSU in the hth stratum, and other symbols are defined in the appendix. In the S2SWR design, the variance of $t_{\text{str}}(x)$ can be directly estimated by

$$v_{\text{str}}\left\{t_{\text{str}}(x)\right\} = \sum_{h=1}^{L} \frac{n_h}{n_h - 1} \sum_{i=1}^{n_h} (t_{hi} - \bar{t}_h)^2, \quad \bar{t}_h = \frac{1}{n_h} \sum_{i=1}^{n_h} t_{hi} \tag{1}$$

This is analogous to (1) in [SVY] **variance estimation**, except that finite population corrections are ignored because sampling is assumed to be with replacement in the S2SWR design.

When the statistic of interest is a function of moments (e.g., means, ratios, correlations, regression coefficients), the customary method of variance estimation is Taylor series expansion, or linearization, also known in theoretical statistics and econometrics as the delta method. Let $\theta = f(T_1, \ldots, T_K)$ be a smooth function of the totals $T_1, \ldots, T_K$, and let $\widehat{\theta} = f(t_1, \ldots, t_K)$ be an estimate of θ, where $t_1, \ldots, t_K$ are sample-based estimates of the corresponding totals. Then the linearization variance estimator is

$$v_L\left(\widehat{\theta}\right) \approx \widehat{\mathrm{MSE}}\left(\widehat{\theta}\right) \approx v\left\{\left(\sum_k \frac{\partial f}{\partial \theta_k}\Big|_{\theta_k=T_k}\right) t_k\right\} \approx v\left\{\left(\sum_k \frac{\partial f}{\partial \theta_k}\Big|_{\theta_k=t_k}\right) t_k\right\}$$

Here the first approximation is to ignore a possible bias of $\widehat{\theta}$; the second approximation is the linearization itself; and the third approximation is to evaluate the necessary derivatives at the sample values $(t_1, \ldots, t_K)$. An important special case is the sample mean

$$\overline{x}_r = \frac{\sum_{hij} w_{hij} x_{hij}}{\sum_{hij} w_{hij}}$$

Subindex r stands for ratio estimator: $\overline{x}_r$ is the ratio $t(x)/t(1)$. Its variance is estimated by

$$v_r(\overline{x}_r) = \sum_h \frac{n_h}{n_h - 1} \sum_i (d_{hi} - \overline{d}_h)^2 \tag{2}$$

where

$$d_{hi} = \frac{1}{N_h} \sum_j w_{hij}(x_{hi} - \overline{x}_r), \quad \overline{d}_h = \frac{1}{n_h} \sum_i d_{hi}$$

The linearization method is also applicable when $\widehat{\theta}$ is derived as a solution to a set of estimating equations

$$t\left\{\psi(x,\theta)\right\} = \sum_{hij} w_{hij} \psi(x_{hij}, \theta) = 0 \tag{3}$$

The most common examples are generalized linear models (Binder 1983) and other models based on quasilikelihoods (Skinner 1989), with $\psi(x,\theta)$ being the score equations. The linearization variance estimator has a sandwich structure:

$$v_L\left(\widehat{\theta}\right) \approx \left[\nabla_\theta t\left\{\psi(x,\theta)\right\}\right]^{-1} v\left[t\left\{\psi(x,\theta)\right\}\right]\left[\nabla_\theta t\left\{\psi(x,\theta)\right\}\right]^{-1} \tag{4}$$

This expression should be contrasted with the traditional variance estimator in fully parametric likelihood inference: the inverse of the Hessian matrix, $-\left[\nabla_\theta t\left\{\psi(x,\theta)\right\}\right]^{-1}$. In complex survey data analysis, the latter is not a consistent estimator of the variance of $\widehat{\theta}$.

Variance estimation based on linearization is the default estimation method for survey data in Stata. The required derivatives are computed analytically for several commonly used models listed in [SVY] **svy estimation** and are computed numerically for other models using `_robust`. The use of the same mechanics for both survey inference and more traditional robust variance estimation, leading to the Huber–White (Huber 1967; White 1982) sandwich variance estimator, is not surprising. In fact, this is the reflection of the aforementioned double robustness property of the design variances and their estimators.

For parameters and statistics that are not smooth functions of the underlying distributions—such as quantiles, extreme order statistics, or certain poverty and income inequality indicators—the linearization variance estimator is not applicable.

When the survey data are released for public use, confidentiality of the respondents must be protected. Geographic information is provided in a coarse form, incomes are top-coded, small racial groups are conglomerated, etc. Variance estimation with (1) or (2) requires that stratum and PSU identifiers h and i are known for each observation. In (4), stratum and PSU identifiers are needed to compute $v\{t(\psi)\}$, which is done by (1) or (2). If the data provider decides that releasing strata and PSU information poses the threat that individual subjects could be identified, alternative variance estimation methods must be used.

3 Resampling methods

The three major resampling, or replication, methods used in complex survey inference (Rust and Rao 1996; Shao 1996, 2003) are balanced repeated replication (BRR), the jackknife, and the bootstrap. In each of these methods, multiple replicates of the original data are created. In the rth replicate, some PSUs are omitted (i.e., all sample elements in PSU_{hi} are removed) and some are retained (and may be included multiple times, as in the bootstrap). The parameter estimate of interest, $\widehat{\theta}_m^{(r)}$, is computed using the same estimation procedure as that for the original data. Subindex m stands for the estimation method. The resulting estimator of variance is generally defined by

$$v_m\left(\widehat{\theta}\right) = \frac{A}{R}\sum_{r=1}^{R}\left\{\widehat{\theta}_m^{(r)} - \widetilde{\theta}\right\}^2 \tag{5}$$

Here R is the number of replicates and A is a scaling constant chosen to ensure that in the linear case, v_m coincides with the known estimator (1). The measure of the central tendency $\widetilde{\theta}$ can be the mean of resampled values

$$\widetilde{\theta} = \frac{1}{R}\sum_{r=1}^{R}\widehat{\theta}_m^{(r)} \tag{6}$$

resulting in the variance version of the estimator or can be the original estimate

$$\widetilde{\theta} = \widehat{\theta} \tag{7}$$

resulting in the mean squared error (MSE) version of the estimator.

In most cases, $v_m\left(\widehat{\theta}\right)/v_L\left(\widehat{\theta}\right) \to 1$ in probability as $n = \sum_h n_h \to \infty$. In other words, in large samples all replication estimators are close to one another and to the linearization estimator. Shao (1996) suggests that the choice of the estimator should be based on computational rather than statistical grounds.

While formal interpretation of resampling methods is that the sample is re-created for each replicate r, a more practical implementation is achieved by varying the sampling weights. For instance, if a sampling unit is removed in a given replicate, it can simply be given a weight of zero. The weights of other units in the same stratum need to be increased to ensure that the totals are unbiased for each replicate. A set of replicate weights, $w_{hij}^{(r)}$, $r = 1, \ldots, R$, is created and distributed with the public release dataset. Variance estimation proceeds by running the same command $1 + R$ times (where the first run is to obtain the point estimates based on the original weights), substituting the replicate weights in place of the original ones, computing the estimates of interest, and combining the results using (5).

For the methods below, I describe the mechanics, the implied replicate weights, and the number of replicates each method requires. I compare the properties of variance estimation procedures in section 3.4.

3.1 Balanced repeated replication

BRR was introduced by McCarthy (1969) for the class of designs in which $n_h = 2$ for all strata. In each replicate, one of the two PSUs is omitted, and the other one is retained and replicated twice to ensure that the totals are on the right scale. Because exactly half of the PSUs are used, the replicates are also referred to as half-samples. The replicate weights are

$$w_{hij}^{(r)} = \begin{cases} 2w_{hij}, & \text{PSU}_{hi} \text{ is retained} \\ 0, & \text{PSU}_{hi} \text{ is omitted} \end{cases} \tag{8}$$

These weights are used to compute $\widehat{\theta}_{\text{BRR}}^{(r)}$. The BRR variance estimator is obtained from (5) with $A = 1$:

$$v_{\text{BRR}}\left(\widehat{\theta}\right) = \frac{1}{R}\sum_{r=1}^{R}\left\{\widehat{\theta}_{\text{BRR}}^{(r)} - \widehat{\theta}\right\}^2 \tag{9}$$

The number of all possible half-samples is 2^L. If all half-samples are used, $v_{\text{BRR}} = v_L = v_r$ in the linear case. McCarthy (1969) showed that this equality holds with a much smaller number of replicates, $L \leq R \leq L + 3$, for resampling designs that satisfy certain balance conditions. To wit, R must be a multiple of 4; each PSU must be used $R/2$ times; and each pair of units from different strata must be used $R/4$ times. Efficient BRR designs are based on Hadamard matrices (Hedayat, Sloane, and Stufken 1999), which are square matrices with entries ± 1 and rows that are mutually orthogonal. It has been conjectured that a Hadamard matrix of order $4k$ exists for every positive integer k. Several matrices are given in Sloane (2004), and the smallest order for which no Hadamard matrix is known is $4k = 668$. BRR designs can be generated from $R \times R$ Hadamard matrix H as follows. If the (h, r)th entry, H_{hr}, of the matrix is $+1$, use the first PSU from stratum h for replicate r; otherwise, use the second PSU:

$$w_{h1j}^{(r)} = (1 + H_{hr})w_{hij}, \quad w_{h2j}^{(r)} = (1 - H_{hr})w_{hij}$$

There are several modifications of the BRR method. For a given pattern of included and excluded PSUs in the rth replicate, a complementary half-sample is obtained by reversing the doubled and excluded units:

$$w_{h1j}^{(rc)} = (1 - H_{hr})w_{hij}, \quad w_{h2j}^{(rc)} = (1 + H_{hr})w_{hij}$$

These weights are used to compute the complementary half-sample estimate $\widehat{\theta}_{\text{BRR}}^{(rc)}$. Then additional variance estimates are

$$v_{\text{BRR2}}\left(\widehat{\theta}\right) \equiv v_{\text{BRR-D}}\left(\widehat{\theta}\right) = \frac{1}{4R}\sum_{r=1}^{R}\left\{\widehat{\theta}_{\text{BRR}}^{(r)} - \widehat{\theta}_{\text{BRR}}^{(rc)}\right\}^2$$

and

$$v_{\text{BRR3}}\left(\widehat{\theta}\right) \equiv v_{\text{BRR-S}}\left(\widehat{\theta}\right) = \frac{1}{2R}\sum_{r=1}^{R}\left\{\widehat{\theta}_{\text{BRR}}^{(r)} - \widetilde{\theta}\right\}^2 + \left\{\widehat{\theta}_{\text{BRR}}^{(rc)} - \widetilde{\theta}\right\}^2$$

where subindices BRR-D and BRR-S stand for the difference and the sum, respectively.

Fay's modification of BRR (Judkins 1990) is to increase the weight of one PSU by a factor of $2 - k$ and decrease the weight of the other PSU by a factor of k for some $0 \le k < 1$:

$$w_{hij}^{(r)} = \begin{cases} (2-k)w_{hij}, & \text{PSU}_{hi} \text{ is retained} \\ kw_{hij}, & \text{PSU}_{hi} \text{ is omitted} \end{cases}$$

The value of $k = 0$ gives the original BRR procedure. The correct scaling factor is $A = 1/(1-k)^2$. If Fay BRR weights are supplied in a public-release data file, the value of k (or A) must be provided to the data user.

BRR can be used to correct for small-sample biases (Rao and Wu 1985). A bias-corrected estimate is

$$\widehat{\theta}_{\text{BRR}} = 2\widehat{\theta} - \frac{1}{R}\sum_{r}\widehat{\theta}^{(r)}$$

or, if complementary half-samples are used,

$$\widehat{\theta}_{\text{BRRc}} = 2\widehat{\theta} - \frac{1}{2R}\sum_{r}\left\{\widehat{\theta}^{(r)} + \widehat{\theta}^{(rc)}\right\}$$

Stata implements the original v_{BRR} (9) with `svy brr`. By default, the variance formulation (6) is used, and the MSE formulation (7) may be requested with `svy brr, mse`. Fay's modification is available with the `fay(#)` option. The BRR replicate weights may be specified via `svyset, brrweight(`*varlist*`)`. If no BRR replicate weights are given, the user must provide a Hadamard matrix with `hadamard(`*matrix*`)`.

An example of a dataset with replicate weights is provided as example 1 of [SVY] **svy brr**:

```
. use http://www.stata-press.com/data/r11/nhanes2brr
. svyset
      pweight: finalwgt
          VCE: brr
          MSE: off
    brrweight: brr_1 brr_2 brr_3 brr_4 brr_5 brr_6 brr_7 brr_8 brr_9 brr_10
               brr_11 brr_12 brr_13 brr_14 brr_15 brr_16 brr_17 brr_18 brr_19
               brr_20 brr_21 brr_22 brr_23 brr_24 brr_25 brr_26 brr_27 brr_28
               brr_29 brr_30 brr_31 brr_32
  Single unit: missing
     Strata 1: <one>
         SU 1: <observations>
        FPC 1: <zero>
```

Note that the default estimation method is `VCE: brr`. Hence, typing `svy:` *command* will invoke variance estimation by BRR:

```
. svy: logistic highbp height weight age female
(running logistic on estimation sample)
BRR replications (32)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
................................
Survey: Logistic regression                     Number of obs      =      10351
                                                Population size    =  117157513
                                                Replications       =         32
                                                Design df          =         31
                                                F(   4,     28)    =     174.52
                                                Prob > F           =     0.0000
------------------------------------------------------------------------------
             |                BRR
      highbp | Odds Ratio   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      height |   .9688567   .0056915    -5.39   0.000     .9573181    .9805344
      weight |   1.052489   .0032886    16.37   0.000     1.045803    1.059217
         age |   1.050473   .0024619    21.01   0.000     1.045464    1.055506
      female |   .7250087   .0650444    -3.58   0.001     .6037789    .8705797
------------------------------------------------------------------------------
```

Most of the design information has been stripped from the data, which is often desirable for confidentiality protection. The estimation output only shows the design degrees of freedom and number of replicates. The point estimates are the same as before, whereas the standard errors and the corresponding confidence intervals are slightly different.

Generalizations of BRR to designs with more than two PSUs per stratum have been proposed (Gurney and Jewett 1975; Gupta and Nigam 1987; Wu 1991; Sitter 1993) but did not find widespread use in practice because of excessive mathematical complexity and limited availability of the orthogonal arrays necessary to construct the BRR schemes.

3.2 The jackknife

While BRR is a replication method unique to complex surveys, the jackknife has been widely used in mainstream statistics (Shao and Tu 1995). In its simplest form for an i.i.d. sample of size n, the rth replicate is obtained by removing the rth observation, and hence the number of replicates is $R = n$. The appropriate scaling factor in (5) is $A = n - 1$.

In complex survey data, the removed units are PSUs, and the number of replicates is the total number of PSUs, $R = n = n_1 + \cdots + n_L$. If PSU k in stratum g is removed in the rth replicate, the replicate weights are

$$w_{hij}^{(gk)} = \begin{cases} 0, & h = g, i = k \\ \frac{n_g}{n_g - 1} w_{hij}, & h = g, i \neq k \\ w_{hij}, & h \neq g \end{cases} \tag{10}$$

The jackknife variance estimators can be defined in a number of ways. Let $\widehat{\theta}^{(hi)}$ be the estimate obtained with unit h, i removed. Then two jackknife variance estimators are defined as follows:

$$v_{J1} = \sum_h \frac{n_h - 1}{n_h} \sum_i \left\{ \widehat{\theta}^{(hi)} - \widehat{\theta}^h \right\}^2$$

$$v_{J2} = \sum_h \frac{n_h - 1}{n_h} \sum_i \left\{ \widehat{\theta}^{(hi)} - \widehat{\theta} \right\}^2$$

The scaling factor needs to be applied within each stratum to produce correct totals and consistent variance estimates. Rao and Wu (1985) provide four additional jackknife estimators that have virtually the same properties and are rarely used in practice.

Like BRR, the jackknife can also be used to correct for small-sample biases of $\widehat{\theta}$ with a bias-corrected estimate

$$\widehat{\theta}_J = (n + 1 - L)\widehat{\theta} - \sum_h (n_h - 1)\widehat{\theta}^h$$

Stata implements the jackknife variance estimation with `svy jackknife`. Either the original design information (strata and PSU) or resampling weights (specified via `svyset, jkrweight(`*varlist*`)`) should be present in the dataset. The default estimator is v_{J1}, and the `mse` option invokes estimator v_{J2}.

(*Continued on next page*)

```
. use http://www.stata-press.com/data/r11/nhanes2
. svy jackknife: logistic highbp height weight age female
(running logistic on estimation sample)
Jackknife replications (62)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
............
Survey: Logistic regression
Number of strata   =        31                  Number of obs      =      10351
Number of PSUs     =        62                  Population size    =  117157513
                                                Replications       =         62
                                                Design df          =         31
                                                F(   4,     28)    =     178.59
                                                Prob > F           =     0.0000
```

highbp	Odds Ratio	Jackknife Std. Err.	t	P>|t|	[95% Conf.	Interval]
height	.9688567	.005684	-5.39	0.000	.9573332	.9805189
weight	1.052489	.0032837	16.40	0.000	1.045813	1.059207
age	1.050473	.0024823	20.84	0.000	1.045422	1.055548
female	.7250087	.0641366	-3.64	0.001	.6053228	.8683592

Like linearization, the jackknife is inconsistent for nonsmooth statistics. This problem can be ameliorated in designs with a large number of PSUs per stratum. For a carefully chosen $k > 1$, delete-k jackknife (Shao and Tu 1995; Wolter 2007) removes k PSUs from the same stratum to form the jackknife replicates. Consistency of the method is a complicated interplay between smoothness of the statistics of interest and parameter k. The complete delete-k jackknife requires $R = \sum_h \binom{n_h}{k} \sim O\left\{(n/L)^k\right\}$ replications, notably increasing the computational burden.

Computational burden can also be excessive when the number of PSUs in the complex survey design is several hundreds or even thousands. Some of the list-based establishment surveys may have such a structure. To reduce the number of replications, PSUs are randomly grouped, with groups acting as quasi-PSUs in the new design. The method is known as the random-groups method (Wolter 2007) or delete-a-group jackknife (Kott 2001). The method has some potential pitfalls, including inconsistency of the variance estimates (Shao 1996) and odd dependencies on the number of groups used (Valliant, Brick, and Dever 2008).

3.3 The bootstrap

Inference in parametric statistical procedures is based on sampling distributions of parameter estimates and test statistics. These distributions can often be derived by transformations of the underlying random variables or by asymptotic arguments. The bootstrap provides an alternative paradigm: it mimics the original sampling procedure to obtain approximations to the sampling distributions of the statistics of interest. The bootstrap samples are taken from a distribution that is close, in some suitable sense, to the unknown population distribution. A typical choice is the empirical distribution of the data.

Let the sample data $x_1, \ldots, x_n$ be i.i.d. from distribution F characterized by parameter $\theta = T(F)$. The empirical distribution function of the data is F_n, and the associated parameter estimate is $\widehat{\theta}_n = T(F_n)$. The bootstrap takes a simple random sample with replacement $(x_1^*, \ldots, x_m^*)$ of size m from $x_1, \ldots, x_n$. The empirical distribution function of the bootstrap sample is F_m^*, and the associated parameter estimate is $\widehat{\theta}_m^* = T(F_m^*)$. The bootstrap distribution of $\widehat{\theta}_m^*$ is obtained by taking different bootstrap samples and computing $\widehat{\theta}_m^*$ for each of them.

The plug-in principle of the bootstrap, illustrated in figure 1 on the next page, states that relation of the bootstrap values $\widehat{\theta}_m^*$ to $\widehat{\theta}_n$ is approximately the same as that of $\widehat{\theta}_n$ to the unknown parameter θ. Typically, but not necessarily, $m = n$. If this is the case, the bootstrap estimates of the moments and the distribution function of $\widehat{\theta}_n$ are

$$\begin{aligned}
\text{Bias}\left(\widehat{\theta}_n\right) &= E\left(\widehat{\theta}_n - \theta\right) \doteq E^*\left(\widehat{\theta}_n^* - \widehat{\theta}_n\right) \\
V\left(\widehat{\theta}_n\right) &= E\left[\left\{\widehat{\theta}_n - E\left(\widehat{\theta}_n\right)\right\}^2\right] \doteq E^*\left[\left\{\widehat{\theta}_n^* - E\left(\widehat{\theta}_n^*\right)\right\}^2\right] \\
\text{MSE}\left(\widehat{\theta}_n\right) &= E\left\{\left(\widehat{\theta}_n - \theta\right)^2\right\} \doteq E^*\left\{\left(\widehat{\theta}_n^* - \theta_n\right)^2\right\} \\
\text{cdf}_{\theta_n}(x) &= \text{Prob}\left(\widehat{\theta}_n - \theta < x\right) \doteq \text{Prob}^*\left(\widehat{\theta}_n^* - \widehat{\theta}_n < x\right)
\end{aligned} \tag{11}$$

where the starred quantities are taken with respect to the bootstrap distribution. The particular strength of the bootstrap is the last equation of (11). The bootstrap accounts for asymmetry of the sampling distributions and gives better one-sided confidence-interval coverage than the confidence intervals based on asymptotic normality (Efron and Tibshirani 1993; Shao and Tu 1995).

The theory of the bootstrap is based on the complete enumeration of all possible samples of size m from the distribution F_n. In practice, instead of the complete bootstrap, a large number R of random samples with replacement from the original data is taken, the statistics of interest are computed for each bootstrap sample, and Monte Carlo distributions of the resulting statistics are used to conduct inference. Two approximations are thus taken. The sampling distributions of the statistics of interest are approximated by the complete bootstrap distributions, and the complete bootstrap distributions are in turn approximated by simulation. Conceptually, the number of samples R should be large enough so that the simulation error is small (Efron and Tibshirani 1993, sec. 6.4). In practice, the number of the bootstrap replicates R is often restricted by computational burden and is usually taken to be between 100 and 1,000.

$$\begin{array}{ccccccc}
F & \overset{\text{sample}}{\longrightarrow} & F_n & \overset{\text{bootstrap}}{\longrightarrow} & (F_n^*) & \overset{\text{simulate}}{\longrightarrow} & F_n^{(*b)} \\
\downarrow T & & \downarrow T & & \downarrow T & & \downarrow T \\
\theta & \overset{\text{inference}}{\longleftrightarrow} & \widehat{\theta}_n & \overset{\text{bootstrap}}{\longleftrightarrow} & \left(\widehat{\theta}_n^*\right) & \approx & \widehat{\theta}_n^{(*b)}
\end{array}$$

Figure 1. The bootstrap principle

Special bootstrap schemes exist to control the error induced by simulation. The package `bsweights` (section 4) implements one such scheme, the balanced bootstrap. The motivation for the balanced bootstrap is that for statistics with known means and variance estimates (such as the sample mean), an attempt should be made to match the moments of the bootstrap distribution with the known ones. This is achieved by carefully tracking the number of times the units appear in the bootstrap samples. The first-order balance is achieved when each unit is included into the bootstrap samples the same number of times (Davison, Hinkley, and Schechtman 1986). The first-order balance removes simulation noise from the mean of the bootstrap distribution and hence from the estimates of bias. The second-order balance is achieved when each pair of units is included the same number of times (Graham et al. 1990). The second-order balance removes simulation noise from the estimates of variance.

Unfortunately, the bootstrap does not solve every problem in statistical inference. Examples where the bootstrap fails are abundant (Canty et al. 2006; Shao and Tu 1995, sec. 3.6). Complex survey data is one such example. More sophisticated resampling schemes or estimation procedures have to be employed in such situations.

We start developing the bootstrap for complex survey data by considering the following naïve bootstrap scheme (NBS). To construct the rth replicate, take a simple random sample with replacement of n_h units from the original data in stratum h; repeat independently across strata; estimate the parameter of interest, $\widehat{\theta}^{(*r)}$; repeat R times; and estimate the variance using (5). Rao and Wu (1988) demonstrated that even in the simple case of the stratified mean (2), the variance of the complete NBS distribution is

$$V^*_{\text{NBS}}(\overline{x}^*) = \sum_h \frac{W_h^2}{n_h} \frac{n_h - 1}{n_h} s_h^2$$

instead of

$$v_r(\overline{x}) = \sum_h \frac{W_h^2}{n_h} s_h^2$$

If the number of PSUs per stratum is small (and this number is often as low as $n_h = 2$), the bootstrap estimator is biased and inconsistent. The issue also exists in the bootstrap for i.i.d. data but is of lesser importance because the bias disappears as $n \to \infty$.

To rectify the bias of the NBS estimate, Rao and Wu (1988) proposed the following rescaling bootstrap (RBS) procedure. A simple random sample with replacement of m_h out of n_h units is taken, and internally scaled pseudovalues

$$\widetilde{x}_{hi}^{(r)} = \overline{x}_h + m_h^{1/2}(n_h - 1)^{-1/2} \left\{ x_{hi}^{(*r)} - \overline{x}_h \right\} \tag{12}$$

are used in computing the variances of the moments and their functions. Here $x_{hi}^{(*r)}, i = 1, \ldots, m_h$, is the rth sample taken from the hth stratum, and $\overline{x}_h = \sum_i x_{hi}/n_h$ is the estimated stratum mean. Rao, Wu, and Yue (1992) extended the RBS method to cover the case of estimating equations (3). They considered replicate weights implied by RBS and demonstrated that the necessary internal scaling can be achieved by the replicate weights

$$w_{hij}^{(*r)} = \left\{1 - m_h^{1/2}(n_h-1)^{-1/2} + m_h^{1/2}(n_h-1)^{-1/2}\frac{n_h}{m_h}m_{hi}^{(*r)}\right\} w_{hij} \tag{13}$$

where $m_{hi}^{(*r)}$ is the bootstrap frequency of unit hi, that is, the number of times PSU_{hi} was used in forming the rth bootstrap replicate. This method of internal scaling is implemented in the `bsweights` package described in section 4.

How should the bootstrap sample size m_h be chosen? First, note that internal scaling is not needed if $m_h = n_h - 1$. McCarthy and Snowden (1985) proposed this choice under the name of the bootstrap with replacement (BWR). Second, Rao and Wu (1988) provided theoretical arguments showing that when the strata variances are known, the choice $m_h = (n_h-2)^2/(n_h-1) \approx n_h-3$ for $n_h > 3$ allows matching of the third-order moments of the bootstrap distribution with those of the theoretical sampling distribution. Simulation evidence indicated that bootstrap estimators with $m_h = n_h - 3$ are unstable for moderate sample sizes $n_h = 5$ and unknown strata variances (Kovar, Rao, and Wu 1988). Finally, note that when PSU_{hi} is omitted from the rth replicate, the replicate weight is $w_{hij}^{(*r)} = \left\{1 - m_h^{1/2}(n_h-1)^{-1/2}\right\} w_{hij}$. If $m_h > n_h - 1$, this weight will become negative, which may lead to violations of natural ranges for parameters such as quantiles, distribution functions, variances, or correlations.

Given the above considerations, $m_h = n_h - 1$ appears to be a good choice that ensures efficiency of the bootstrap estimators without producing any artifacts like range restriction violations.

How should the number of replications R be chosen? When the data are i.i.d., we argued that this number should be chosen to make Monte Carlo variability of the bootstrap variance estimates sufficiently small. For complex surveys, it is also desirable that the number of replicates is at least as large as the design degrees of freedom, $n - L$. The design degrees of freedom is the largest possible rank of the covariance matrix of the coefficient estimates. The choice $R < n - L$ will not allow the bootstrap to provide this highest possible rank. The degrees of freedom may not be an issue if $n - L$ is a sufficiently large number (e.g., exceeds 100).

To increase the number of replicates, and hence the stability of the estimates, the mean bootstrap (Yung 1997; Yeo, Mantel, and Liu 1999) can be used. To compute the rth replicate weight variable, the bootstrap frequencies are averaged across a series of K subsequent replicates, and the average frequency

$$\overline{m}_{hi}^{(*r)} = \frac{1}{K}\sum_{k=(r-1)K+1}^{rK} m_{hi}^{(*k)}$$

is used instead of $m_{hi}^{(*r)}$ to scale weights according to (13). The total number of bootstrap replications in this method is the product RK, while the number of replicate weight variables is R. The scaling factor A in (5) needs to be set equal to K to ensure consistency. There are some similarities between this scheme and Fay's modification of BRR. The motivation for both schemes is confidentiality protection: PSUs should

never receive zero replicate weights. Also, the scaling factor A needs to be increased to compensate for smaller variability of the replicate weights.

3.4 Comparison of estimators and relations between them

The linearization, jackknife, bootstrap, and (where applicable) BRR variance estimators are estimating the same quantity, the variance $V\left(\widehat{\theta}\right) = \sigma^2$ of statistic $\widehat{\theta}$. Can we identify conditions under which some estimators perform better than others? (As noted by Eltinge (1996), different goals of variance estimation may lead to different estimators being preferred.) A number of comparisons, both theoretical and empirical (by simulation), have been made in the literature.

In the special case of linear statistics of moments such as totals, the variance estimators coincide: $v_L = v_J = v_{\text{BRR}} = v_{\text{BOOT}}$, covering all versions of the jackknife and BRR estimators, and rescaling (but not naïve) bootstrap estimators.

Consistency of various versions of v_{BRR} and v_J, as well as v_L, was established by Krewski and Rao (1981) for smooth functions under the important setting of a bounded number of PSUs per stratum and number of strata $L \to \infty$. They also found that in terms of two-sided confidence-interval coverage, BRR was the best-performing method, followed by the jackknife, and then by linearization. In terms of stability, i.e., the mean squared error $E\left\{(v_m - \sigma^2)^2\right\}$, the ordering was reversed.

Rao and Wu (1985, 1988) demonstrated that different jackknife and BRR estimators are very close to one another and that the jackknife is closest to the linearization estimator, followed by the bootstrap and BRR estimators. Also, v_{BRR1} tends to be farther from v_L than v_{BRR2} or v_{BRR3}. There is no preferred estimator in terms of bias: Rao and Wu (1985) found conditions under which each of v_L, v_{J2}, and various versions of BRR estimators had the smallest biases. This is an interesting observation, because the linearization estimator v_L is usually considered the "golden standard" (if it is applicable for a given estimation problem). Valliant (1996) also discussed several situations in which the jackknife estimator exhibited better model-based properties than did the linearization estimator in ratio estimation and in poststratification.

Balanced bootstraps have a lot in common with BRR. If $n_h = 2$, the bootstrap scheme that avoids internal rescaling has the bootstrap sample size $m_h = n_h - 1 = 1$; that is, the bootstrap samples are random half-samples. The second-order balance conditions dictate that the number of times units from different strata are resampled together is the same for all pairs of units. These conditions are identical to the BRR balance conditions. Hence, BRR can be viewed as the second-order balanced bootstrap resampling scheme. Nigam and Rao (1996) proposed the second-order balanced bootstrap schemes for more general designs with $m_h = n_h$ equal to an even number or a prime power.

Overall, the jackknife and linearization methods tend to exhibit similar performance. They are more stable for smooth functions but inconsistent for nonsmooth functions. The method that is applicable for all statistics is the bootstrap (and its kin, BRR, for designs with $n_h = 2$). Additionally, the bootstrap can provide more-accurate one-sided

confidence intervals and better balance of the tail probabilities of two-sided confidence intervals. However, this versatility comes at the price of lesser stability and longer confidence intervals.

4 The bsweights command

4.1 Syntax

`bsweights` *prefix*, `reps(#) n(#)` [`replace average(#) balanced dots calibrate(`*call_to_weight_calibration_routine*`) verbose nosvy seed(#) float double`]

The *call_to_weight_calibration_routine* is

[`do`] *calibration_program* [*arg1*] `@` [*arg2*]

4.2 Options

`reps(#)` specifies the number of bootstrap replications to be taken and the number of weight variables to be generated. `reps()` is required.

`n(#)` specifies how the number of PSUs m_h per stratum be handled. If a positive number is specified, it is interpreted as the number of units per stratum m_h that will be used. If a nonpositive number is specified, then $m_h = n_h - |\#|$. Specifying `n(0)` leads to the number of units resampled being equal to the original number of units n_h. `n()` is required.

`replace` requests that the weight variables be created anew. Use with caution; it will `drop` the existing *prefix** variables!

`average(#)` implements the mean bootstrap. The bootstrap frequency counts are averaged across the given number of replications. If the bootstrap weights are created using this option, you should specify the `vfactor()` option of `bs4rw`. The total number of replications is the product of `reps()` and `average()`.

`balanced` requests the balanced bootstrap (Graham et al. 1990). See *Remarks* below.

`dots` provides additional output.

`calibrate(`*call_to_weight_calibration_routine*`)` allows a call to another program to adjust the weights for poststratification and nonresponse. See *Remarks* below.

`verbose` provides output from the weights calibration commands.

`nosvy` explicitly states that the data are not of the survey format.

`seed(#)` specifies the seed for the random-number generator. See [D] **generate**.

`float` specifies that the weight variables have float type. See [D] **data types**.

`double` specifies that the weight variables have double type. See [D] **data types**.

4.3 Remarks

Calibration

When rescaling the replicate weights, `bsweights` can make calls to *calibration_program* substituting the current replicate weight variable being processed for the symbol `@`. For instance, if you specify

```
. bsweights bsw, ... calibrate(do adjust @)
```

then `bsweights` will issue the consecutive commands

```
do adjust bsw1
do adjust bsw2
...
```

In turn, the do-file `adjust.do` might contain

```
args weight_var
...
replace `weight_var´ = ...
...
```

See examples 5 and 6 in section 5.1.

The weight adjustments are taking place after the internal scaling (13). It is the user's responsibility to provide correct treatment of the resampling weights in their calibration procedures. The `verbose` option provides output from calibration commands for debugging purposes.

For proper results, you must specify as inputs to `bsweights` the original probability weights rather than the final sampling weights. The same adjustment procedure that produces the publicly available weights from the probability weights should be applied to the bootstrap weights.

Balanced bootstraps

The balanced bootstrap in `bsweights` is implemented using permutation algorithm BB2 of Gleason (1988). Only the first-order balance is achieved with this algorithm. For stratified samples, the balanced bootstrap is conducted in each stratum independently.

For the bootstrap scheme to be first-order balanced, certain simple bookkeeping conditions must be satisfied. Namely, it is necessary that the total number of units recycled from stratum h across all bootstrap replicates be divisible by n_h for all strata h. It will be satisfied if R (or KR in the case of the mean bootstrap) is divisible by the

least common multiple of $n_1, \ldots, n_L$. The returned value `r(balanced)` shows whether the first-order balance was achieved.

Using the bootstrap weights

There are two ways to use the bootstrap weights generated by `bsweights`. In the examples below, I use the `bs4rw` command written by Jeff Pitblado of StataCorp. This is an analogue of the official `bootstrap` command that uses the replicate weights instead of actually resampling the data in Stata memory. To install `bs4rw`, type `findit bs4rw` and follow the instructions. Here are a few general comments about `bs4rw`.

The command called by `bs4rw` with different sets of weights must accept probability weights, `[pweight=`*exp*`]`, or importance weights, `[iweight=`*exp*`]`, as part of its syntax. This rules out many important commands, such as `correlate` and `xtmixed`.

The first call `bs4rw` makes is to find the point estimates. Therefore, the command specification must contain the original weights; otherwise, the point estimates reported in the output of `bs4rw` will be incorrect.

The bootstrap postestimation summaries (estimates of bias and various confidence intervals) are available with `estat bootstrap`; see [R] **bootstrap postestimation**.

An alternative way to use the replicate weights is outlined by Phillips (2004). As long as both bootstrap and BRR use the same replication variance estimation formula (5) with the same scaling factor A, you can trick Stata (or any other software that performs BRR estimation) into accepting the bootstrap weights as the BRR weights; see example 4 in section 5.1.

4.4 Saved results

Scalars

`r(balanced)`	`1` if the first-order balance was achieved, `0` otherwise

5 Examples

Here are several examples (using Stata example datasets) of how `bsweights` and `bs4rw` can be used.

5.1 Complex survey data

The complex survey data examples will be based on the aforementioned NHANES II data.

To demonstrate the flexibility of `bsweights`, we shall collapse some of the strata, producing a pseudodesign with seven pseudostrata. The numbers of PSUs in these strata are 4, 4, 8, 8, 12, 10, and 16. We also need to recode the PSU variable to make it run from 1 to 62.

```
. use http://www.stata-press.com/data/r11/nhanes2
. generate cstrata = floor(sqrt(2*strata-1))
. egen upsu = group(strata psu)
. svyset upsu [pw=finalwgt], strata(cstrata)
      pweight: finalwgt
          VCE: linearized
  Single unit: missing
     Strata 1: cstrata
         SU 1: upsu
        FPC 1: <zero>
. svy: logistic highbp height weight age female
(running logistic on estimation sample)
Survey: Logistic regression
Number of strata   =         7                  Number of obs      =      10351
Number of PSUs     =        62                  Population size    =  117157513
                                                Design df          =         55
                                                F(   4,     52)    =     205.17
                                                Prob > F           =     0.0000
------------------------------------------------------------------------------
             |             Linearized
      highbp | Odds Ratio   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      height |   .9688567   .0062847    -4.88   0.000     .9563433    .9815338
      weight |   1.052489   .0031645    17.01   0.000     1.046166     1.05885
         age |   1.050473   .0023319    22.18   0.000      1.04581    1.055157
      female |   .7250087    .073301    -3.18   0.002      .592036    .8878474
------------------------------------------------------------------------------
```

▷ Example 1: BWR scheme

In the first example, we shall create bootstrap weights with arbitrarily chosen $R = 100$ replications and the bootstrap sample size $m_h = n_h - 1$ most commonly used in practice:

```
. bsweights bw, reps(100) n(-1) seed(10101) dots
Running bsample 100  times ................................................
> ..........................................
Rescaling weights
................................................................................
> ..................
Warning: the first-order balance was not achieved
```

```
. bs4rw, rweights(bw*): logistic highbp height weight age female [pw=finalwgt]
(running logistic on estimation sample)
BS4Rweights replications (100)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
..................................................   100
Logistic regression                               Number of obs      =     10351
                                                  Replications       =       100
                                                  Wald chi2(4)       =    982.09
                                                  Prob > chi2        =    0.0000
Log pseudolikelihood = -2961.5987                 Pseudo R2          =    0.1527
```

highbp	Observed Odds Ratio	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
height	.9688567	.0065946	-4.65	0.000	.9560174	.9818685
weight	1.052489	.0035124	15.33	0.000	1.045627	1.059395
age	1.050473	.0023775	21.76	0.000	1.045824	1.055143
female	.7250086	.0780813	-2.99	0.003	.5870448	.8953958

The standard errors are within 10% of the linearization-based ones, and the inference conclusions regarding significant variables are unchanged. The `dots` option provides additional output, including the warning about the lack of balance.

◁

▷ Example 2: Balanced bootstrap scheme

In this example, the necessary conditions for the first-order balance (section 4.3) will be taken into account to set up a first-order balanced scheme. The least common multiple of the strata sizes is 240. The necessary condition for the first-order balance is that the product $m_h R$ is divisible by 240 for all strata. We can choose the replication scheme with $R = 80$ and $m_h = 3 \leq n_h - 1$ for all h (the new or changed options are underlined):

```
. bsweights bw, reps(80) n(3) seed(10101) balanced dots replace
Balancing within strata:
.......
Rescaling weights
................................................................................
```

(*Continued on next page*)

```
. bs4rw, rweights(bw*): logistic highbp height weight age female [pw=finalwgt]
(running logistic on estimation sample)
BS4Rweights replications (80)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
.................................................    50
.............................
Logistic regression                               Number of obs      =     10351
                                                  Replications       =        80
                                                  Wald chi2(4)       =    786.21
                                                  Prob > chi2        =    0.0000
Log pseudolikelihood = -2961.5987                 Pseudo R2          =    0.1527
```

highbp	Observed Odds Ratio	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
height	.9688567	.0057047	-5.37	0.000	.9577399	.9801025
weight	1.052489	.003199	16.83	0.000	1.046237	1.058777
age	1.050473	.0021039	24.59	0.000	1.046358	1.054605
female	.7250086	.0716213	-3.26	0.001	.5973868	.8798946

There is no warning about lack of balance in the output of `bsweights`. The standard errors are close to the ones obtained earlier.

◁

The above balanced bootstrap scheme does not use information from larger strata very effectively, because only 3 out of 16 PSUs are resampled from the largest stratum. Better schemes would have fewer units omitted from every strata, keeping m_h close to n_h. Given the number of PSUs per stratum in the current data configuration, if $m_h = n_h - 1$, the bootstrap sample sizes are odd numbers ranging from 3 to 15, and hence $R = 240$ replicates need to be taken. If we want to reduce the computational burden, we can set $m_h = n_h - 2$. Then the bootstrap sample sizes are even numbers ranging from 2 to 14, and the number of replicates can be reduced to 120.

▷ Example 3: Mean bootstrap

Another efficient way to reduce the computational burden is to use the mean bootstrap. It creates R replicate weight variables from RK bootstrap replicates. The number of replicates that is averaged over by the mean bootstrap to create one replicate weight, K, must be carried over to `bs4rw` with the `vfactor(`K`)` option.

```
. bsweights bw, reps(120) average(10) n(-1) seed(10101) balanced dots replace
Balancing within strata:
.......
Rescaling weights
................................................................................
> ......................................
Warning: the first-order balance was not achieved
```

```
. bs4rw, rweights(bw*) vfactor(10): logistic highbp height weight age female
> [pw=finalwgt]
(running logistic on estimation sample)
BS4Rweights replications (120)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
.................................................    50
.................................................   100
....................
Logistic regression                               Number of obs      =     10351
                                                  Replications       =       120
                                                  Wald chi2(4)       =    711.62
                                                  Prob > chi2        =    0.0000
Log pseudolikelihood = -2961.5987                 Pseudo R2          =    0.1527
```

highbp	Observed Odds Ratio	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
height	.9688567	.0060641	-5.05	0.000	.957044	.9808152
weight	1.052489	.0033971	15.85	0.000	1.045851	1.059168
age	1.050473	.0024524	21.09	0.000	1.045677	1.055291
female	.7250086	.0725368	-3.21	0.001	.5959102	.8820749

Because the effective number of bootstrap replications $RK = 120 \times 10 = 1200$ is larger than in earlier examples, the standard errors in this scheme are more stable. Note the use of the `vfactor()` option in the call to `bs4rw`.

◁

▷ Example 4: Bootstrap weights as BRR weights

As mentioned in section 4.3, it is possible to use the existing `svy` commands with the bootstrap weights by specifying them as the BRR weights (Phillips 2004). The weight variables will be provided to `svyset` with the `brrweight()` option, and Fay's scaling correction is $1 - 1/\sqrt{K}$:

```
. local mean2fay = 1-sqrt(1/10)
. svyset [pw=finalwgt], vce(brr) brrweight(bw*) fay(`mean2fay´)
      pweight: finalwgt
          VCE: brr
          MSE: off
    brrweight: bw1 bw2 bw3 bw4 bw5 bw6 bw7 bw8 bw9 bw10 bw11 bw12 bw13 bw14
  (output omitted)
              bw109 bw110 bw111 bw112 bw113 bw114 bw115 bw116 bw117 bw118
              bw119 bw120
         fay: .68377223
Single unit: missing
   Strata 1: <one>
       SU 1: <observations>
      FPC 1: <zero>
```

```
. svy brr: logistic highbp height weight age female
(running logistic on estimation sample)
BRR replications (120)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
..................................................   100
..................
Survey: Logistic regression                     Number of obs      =      10351
                                                Population size    =  117157513
                                                Replications       =        120
                                                Design df          =        119
                                                F(   4,    116)    =     174.87
                                                Prob > F           =     0.0000
```

highbp	Odds Ratio	BRR Std. Err.	t	P>|t|	[95% Conf.	Interval]
height	.9688567	.0060387	-5.08	0.000	.9569729	.9808881
weight	1.052489	.0033829	15.92	0.000	1.045811	1.059208
age	1.050473	.0024421	21.18	0.000	1.045648	1.05532
female	.7250087	.0722339	-3.23	0.002	.5952031	.8831231

The advantage of using the bootstrap replicate weights as BRR weights is that Stata post–`svy` estimation commands, such as the design-effect estimation, can be invoked seamlessly:

```
. estat effect
```

highbp	Coef.	BRR Std. Err.	DEFF	DEFT
height	-.0316386	.0062328	1.2491	1.11763
weight	.0511574	.0032142	1.8184	1.34848
age	.0492406	.0023248	1.03103	1.01539
female	-.3215716	.0996318	.972727	.986269
_cons	-2.858968	1.085424	1.27086	1.12732

However, because `svy brr` makes additional assumptions about the design, some of the inferential statistics will be computed incorrectly. Most importantly, the design degrees of freedom will be assumed to be equal to the number of replicates R, which may be much greater than the degrees of freedom of the actual design. Furthermore, the inferential statistics that use this degrees of freedom will also be questionable. Among these inferential statistics are the overall F test, t tests of individual coefficients, and confidence intervals based on the t distribution. Implementation of the bootstrap estimation with `bs4rw` is free of these problems because it relies on asymptotic normality of the estimates. Thus the analysis in example 3 contained χ^2 instead of the F test, and the confidence intervals were based on the normal distribution and hence slightly shorter.

If the usual RBS rather than the mean bootstrap is used, Fay's correction can be omitted.

◁

Example 5: Calibration

As discussed by Shao (1996), if the original probability weights were modified to account for poststratification and nonresponse, the bootstrap procedure must take the original probability weights, process the bootstrap frequencies $m_{hi}^{(*r)}$ according to (13), and then apply the same adjustments that were used with the original weights. `bsweights` allows the performance of weight adjustments using the `calibrate()` option.

Suppose that we want to calibrate the bootstrap weights so that the sums of weights for each gender are equal to the original sums of weights in the NHANES II data. Before we run `bsweights`, we need to store these sums:

```
. svyset upsu [pw=finalwgt], strata(cstrata)
. generate byte ones = 1
. quietly svy: total ones, over(sex)
. matrix Totals = e(b)
```

Next let us define the weight calibration program. It will compute the current sum of weights and scale it to match the existing gender totals.

```
. capture program drop CalGender
. program CalGender
  1.    args wvar
  2.    total ones [pw=`wvar´], over(sex)
  3.    replace `wvar´ = `wvar´*Totals[1,1]/_b[ones:Male] if sex==1
  4.    replace `wvar´ = `wvar´*Totals[1,2]/_b[ones:Female] if sex==2
  5. end
```

We now can create our replicate weights and run the estimation procedure:

```
. bsweights bw, n(-2) reps(120) dots balanced calibrate(CalGender @) seed(10101)
> replace
Balancing within strata:
.......
Rescaling weights
................................................................................
> ........................................
```

(*Continued on next page*)

```
. bs4rw, rweights(bw*): logistic highbp height weight age female [pw=finalwgt]
(running logistic on estimation sample)
BS4Rweights replications (120)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
..................................................   100
....................
Logistic regression                               Number of obs      =     10351
                                                  Replications       =       120
                                                  Wald chi2(4)       =    863.55
                                                  Prob > chi2        =    0.0000
Log pseudolikelihood = -2961.5987                 Pseudo R2          =    0.1527
```

highbp	Observed Odds Ratio	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
height	.9688567	.0058832	-5.21	0.000	.9573943	.9804564
weight	1.052489	.0032217	16.71	0.000	1.046193	1.058822
age	1.050473	.0022441	23.05	0.000	1.046084	1.054881
female	.7250086	.0718354	-3.25	0.001	.5970412	.880404

Each dot under `Rescaling weights` includes both the internal scaling (13) and a call to the calibration program.

◁

Generally speaking, calibration conflicts with the mean bootstrap. Calibration needs to be performed for every bootstrap sample after the weights are rescaled. The mean bootstrap, however, takes averages across the bootstrap replicates and then applies rescaling. An exception to this incompatibility is domain estimation, described next.

▷ Example 6: Domain estimation

Domain estimation (West, Berglund, and Heeringa 2008) is a special case of calibration where the weights outside the domain are set to zero. Suppose that we want to conduct a separate analysis for females only, reproducing the subpopulation estimation of example 2 of [SVY] **svy estimation**. The variable `female` takes on the value 0 for males and 1 for females, so the analogue to using the `subpop(female)` option of `svy` will be achieved through the following calibration program:

```
. capture program drop CalSubpop
. program CalSubpop
  1.    args wvar
  2.    replace `wvar´ = `wvar´ * female
  3. end
```

Now we can run `bsweights` and call `CalSubpop` for the calibration step:

```
. capture drop bw*
. bsweights bw, reps(120) average(10) n(-2) balanced dots calibrate(CalSubpop @)
> seed(10101) replace
Warning: combination of calibration with mean bootstrap can lead to incorrect
> results
Balancing within strata:
.......
Rescaling weights
................................................................................
> ....................................
Warning: the first-order balance was not achieved
. bs4rw, rweights(bw*) vfactor(10): logistic highbp height weight age
> [pw=finalwgt*female]
(running logistic on estimation sample)
BS4Rweights replications (120)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
.................................................    50
.................................................   100
....................
Logistic regression                             Number of obs      =     10351
                                                Replications       =       120
                                                Wald chi2(3)       =    514.79
                                                Prob > chi2        =    0.0000
Log pseudolikelihood = -1359.1291               Pseudo R2          =    0.1703

------------------------------------------------------------------------------
             |               Observed   Bootstrap                         Normal-based
      highbp |          Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      height |            .9765379   .0096453    -2.40   0.016     .9578152    .9956266
      weight |            1.047845   .0039144    12.51   0.000     1.040201    1.055545
         age |            1.058105   .0034868    17.14   0.000     1.051293    1.064961
------------------------------------------------------------------------------
```

`bsweights` issued a warning that calibration and the mean bootstrap are generally incompatible. Note how the original `pweight` was modified in the call to `bs4rw`. Instead of `[pw=finalwgt]`, which would be appropriate for estimation based on the full sample, the units outside the domain were zeroed out by `[pw=finalwgt*female]`. Without this modification, `bs4rw` would report the point estimates for the complete sample rather than the subpopulation only.

◁

5.2 Nonsurvey data

As a final example, let us consider the use of `bsweights` and `bs4rw` outside of the survey context, where the two commands can be used to provide first-order balanced bootstrap simulations.

▷ Example 7: Balanced bootstrap for i.i.d. data

We start with the all-time favorite `auto.dta` and generate a set of balanced bootstrap weights:

```
. sysuse auto, clear
(1978 Automobile Data)
. bsweights bw, nosvy rep(100) n(37) balanced seed(2083)
```

The bootstrap sample size is set to 37, exactly half of the dataset size, and the number of replications is even, so the total number of resampled units in all replicates is the multiple of the dataset size, $n = 74$.

Let us first address bootstrap estimation for an unbiased statistic:

```
. mean price
Mean estimation                     Number of obs    =      74
```

	Mean	Std. Err.	[95% Conf.	Interval]
price	6165.257	342.8719	5481.914	6848.6

```
. quietly bs4rw, rweights(bw*): mean price
. estat bootstrap
Mean estimation                                 Number of obs      =        74
                                                Replications       =       100
```

	Observed Mean	Bias	Bootstrap Std. Err.	[95% Conf.	Interval]	
price	6165.2568	.0000108	298.10338	5570.475	6700.198	(BC)

```
(BC)   bias-corrected confidence interval
. quietly bstrap, rep(100): mean price
. estat bootstrap
Mean estimation                                 Number of obs      =        74
                                                Replications       =       100
```

	Observed Mean	Bias	Bootstrap Std. Err.	[95% Conf.	Interval]	
price	6165.2568	-64.61972	333.47902	5507.595	6903	(BC)

```
(BC)   bias-corrected confidence interval
```

The regular bootstrap provided an estimate of bias that is nonnegligible, while the estimate of bias coming from the balanced bootstrap is a numeric zero.

What happens when the statistic is indeed biased in small samples? Let us consider the bootstrap estimation procedures for a ratio:

```
. ratio price/mpg
Ratio estimation                    Number of obs    =      74
     _ratio_1: price/mpg
```

	Ratio	Linearized Std. Err.	[95% Conf.	Interval]
_ratio_1	289.4854	21.92466	245.7896	333.1812

```
. quietly bs4rw, rw(bw*): ratio price/mpg
. estat bootstrap
Ratio estimation                                Number of obs      =        74
                                                Replications       =       100
```

	Observed Ratio	Bias	Bootstrap Std. Err.	[95% Conf.	Interval]	
_ratio_1	289.48541	.4471263	19.208605	245.2041	322.1176	(BC)

```
(BC)   bias-corrected confidence interval
. quietly bstrap, rep(100): ratio price/mpg
. estat bootstrap
Ratio estimation                                Number of obs      =        74
                                                Replications       =       100
```

	Observed Ratio	Bias	Bootstrap Std. Err.	[95% Conf.	Interval]	
_ratio_1	289.48541	-1.924227	21.9389	257.3014	342.1601	(BC)

```
(BC)   bias-corrected confidence interval
```

Now we see that a nontrivial amount of bias is reported by both methods. However, the estimate coming from the balanced bootstrap is much more trustworthy, because the simulation noise has been removed from it. An interested reader is encouraged to vary `seed(#)` and observe changes in the reported results for both balanced and unbalanced bootstraps.

◁

6 Further remarks

There are several scaling issues associated with replication methods. The first issue is scaling of the point estimates. If there are too few units resampled from stratum h, the weights of these units need to be increased so that the totals can be estimated without bias. Examples of the explicit expressions are given by the BRR and the jackknife replicate weights (8) and (10). The second issue is the scale of the deviations between $\widehat{\theta}^{(r)}$ and $\widehat{\theta}$. In BRR and BWR, these deviations are on the correct scale, so the scaling factor for the resulting estimator $v_m\left(\widehat{\theta}\right)$ to match the known v_r is $A = 1$. On the other hand, these deviations are too small in the jackknife, Fay's modification of BRR, and the mean bootstrap, so these methods need to apply the scaling factors $A > 1$ (and the jackknife needs the scaling to be performed within strata). The third scaling issue is that of internal scaling for the bootstrap procedures where samples are taken from small populations of size n_h. The differences between $\widehat{\theta}^{(r)}$ and $\widetilde{\theta}$ are on the wrong scale if $m_h \neq n_h - 1$, so modifications like (12) or (13) need to be taken.

While `bsweights` provides functionality to create the bootstrap replicate weights on the spot, a better practice is to create a fixed set of weights and run different analyses

using the same weights. This guarantees reproducibility of results between different runs and different researchers.

Because designs with two PSUs per stratum are widely used in practice, BRR is the most popular replication variance estimation procedure. Examples of datasets supplied with BRR weights include U.S. datasets from NHANES and the National Education Longitudinal Survey. Given the popularity and simplicity of BRR estimation, survey organizations often approximate their actual designs with stratified PSUs/stratum designs and provide quasi-BRR weights. The modifications to an original design that could make it "BRR-able" include collapsing of strata, reallocating PSUs to similar strata, or merging PSUs in a stratum to obtain two groups of PSUs so that grouped BRR can be applied to these groups. In some situations, this can cause problems, as discussed in section 3.2.

While the U.S. agencies tend to favor BRR estimation, Statistics Canada extensively uses the bootstrap procedures. Researchers in Canadian universities have access to Statistics Canada complex survey data through the network of Research Data Centers. The bootstrap procedures based on replicate weights are run on Statistics Canada servers to process researchers' data analysis requests. The particular version of the bootstrap favored by Statistics Canada is the Rao–Wu rescaling bootstrap with $m_h = n_h - 1$.

Difficulties may arise in replication variance estimation for domains. Because some units are removed when replicates are constructed, the number of available observations in the domain decreases. Some strata or even the complete replicate dataset may be left with no observations in the domain, and estimation will result in missing parameter estimates. In such situations, Stata will print a red `e` or `x` instead of a dot in the `bs4rw` output.

A similar issue may occur in logistic regression and some other limited-dependent-variable models where insufficient variability in the replicate data may lead to perfect prediction. In this case, Stata drops the perfect predictor, and the estimation results become invalid for use by `bs4rw`.

A possible remedy for both problems is using replication methods that lead to nonzero weights for all units, such as Fay's modification of BRR and the mean bootstrap. The latter, however, is not compatible with poststratification and nonresponse adjustments.

7 Acknowledgments

I would like to thank Jeff Pitblado for helpful discussions and code suggestions. I would also like to thank a referee for a very informative review of the article with multiple clarifications. Many of the referee's ideas and suggestions have been incorporated. All remaining errors, omissions, and bugs are my responsibility.

8 References

Binder, D. A. 1983. On the variances of asymptotically normal estimators from complex surveys. *International Statistical Review* 51: 279–292.

Binder, D. A., and G. R. Roberts. 2003. Design-based and model-based methods for estimating model parameters. In *Analysis of Survey Data*, ed. R. L. Chambers and C. J. Skinner, 29–48. New York: Wiley.

Canty, A. J., A. C. Davison, D. V. Hinkley, and V. Ventura. 2006. Bootstrap diagnostics and remedies. *Canadian Journal of Statistics* 34: 5–27.

Chambers, R. L., and C. J. Skinner, ed. 2003. *Analysis of Survey Data*. New York: Wiley.

Cochran, W. G. 1977. *Sampling Techniques*. 3rd ed. New York: Wiley.

Davison, A. C., D. V. Hinkley, and E. Schechtman. 1986. Efficient bootstrap simulation. *Biometrika* 73: 555–566.

Efron, B., and R. J. Tibshirani. 1993. *An Introduction to the Bootstrap*. New York: Chapman & Hall/CRC.

Eltinge, J. 1996. Discussion of "Resampling methods in sample surveys" by J. Shao. *Statistics* 27: 241–244.

Gleason, J. R. 1988. Algorithms for balanced bootstrap simulations. *American Statistician* 42: 263–266.

Graham, R. L., D. V. Hinkley, P. W. M. John, and S. Shi. 1990. Balanced design of bootstrap simulations. *Journal of the Royal Statistical Society, Series B* 52: 185–202.

Gupta, V. K., and A. K. Nigam. 1987. Mixed orthogonal arrays for variance estimation with unequal numbers of primary selections per stratum. *Biometrika* 74: 735–742.

Gurney, M., and R. S. Jewett. 1975. Constructing orthogonal replications for variance estimation. *Journal of the American Statistical Association* 70: 819–821.

Hedayat, A. S., N. J. A. Sloane, and J. Stufken. 1999. *Orthogonal Arrays: Theory and Applications*. New York: Springer.

Heeringa, S. G., B. T. West, and P. A. Berglund. 2010. *Applied Survey Data Analysis*. Boca Raton, FL: Chapman & Hall/CRC.

Huber, P. J. 1967. The behavior of maximum likelihood estimates under nonstandard conditions. In Vol. 1 of *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 221–233. Berkeley: University of California Press.

Judkins, D. R. 1990. Fay's method for variance estimation. *Journal of Official Statistics* 6: 223–239.

Kish, L. 1995. *Survey Sampling*. 3rd ed. New York: Wiley.

Korn, E. L., and B. I. Graubard. 1995. Analysis of large health surveys: Accounting for the sampling design. *Journal of the Royal Statistical Society, Series A* 158: 263–295.

Kott, P. S. 2001. The delete-a-group jackknife. *Journal of Official Statistics* 17: 521–526.

Kovar, J. G., J. N. K. Rao, and C. F. J. Wu. 1988. Bootstrap and other methods to measure errors in survey estimates. *Canadian Journal of Statistics* 16 (Suppl.): 25–45.

Krewski, D., and J. N. K. Rao. 1981. Inference from stratified samples: Properties of the linearization, jackknife, and balanced repeated replication methods. *Annals of Statistics* 9: 1010–1019.

Lehtonen, R., and E. Pahkinen. 2004. *Practical Methods for Design and Analysis of Complex Surveys*. 2nd ed. New York: Wiley.

Lohr, S. L. 2010. *Sampling: Design and Analysis*. 2nd ed. Pacific Grove, CA: Duxbury.

Lumley, T. S. 2010. *Complex Surveys: A Guide to Analysis Using R*. Hoboken, NJ: Wiley.

McCarthy, P. J. 1969. Pseudo-replication: Half samples. *Review of the International Statistical Institute* 37: 239–264.

McCarthy, P. J., and C. B. Snowden. 1985. The bootstrap and finite population sampling. In *Vital and Health Statistics*, 1–23. Washington, DC: U.S. Government Printing Office.

Nigam, A. K., and J. N. K. Rao. 1996. On balanced bootstrap for stratified multistage samples. *Statistica Sinica* 6: 199–214.

Phillips, O. 2004. Using bootstrap weights with WesVar and SUDAAN. *Research Data Centres Information and Technical Bulletin* 1: 6–15.

Rao, J. N. K., and C. F. J. Wu. 1985. Inference from stratified samples: Second-order analysis of three methods for nonlinear statistics. *Journal of the American Statistical Association* 80: 620–630.

———. 1988. Resampling inference with complex survey data. *Journal of the American Statistical Association* 83: 231–241.

Rao, J. N. K., C. F. J. Wu, and K. Yue. 1992. Some recent work on resampling methods for complex surveys. *Survey Methodology* 18: 209–217.

Rust, K. F., and J. N. K. Rao. 1996. Variance estimation for complex surveys using replication techniques. *Statistical Methods in Medical Research* 5: 283–310.

Särndal, C.-E., B. Swensson, and J. Wretman. 1992. *Model Assisted Survey Sampling*. New York: Springer.

Shao, J. 1996. Resampling methods in sample surveys (with discussion). *Statistics* 27: 203–254.

———. 2003. Impact of the bootstrap on sample surveys. *Statistical Science* 18: 191–198.

Shao, J., and D. Tu. 1995. *The Jackknife and Bootstrap*. New York: Springer.

Sitter, R. R. 1993. Balanced repeated replications based on orthogonal multi-arrays. *Biometrika* 80: 211–221.

Skinner, C. J. 1989. Domain means, regression and multivariate analysis. In *Analysis of Complex Surveys*, ed. C. J. Skinner, D. Holt, and T. M. F. Smith, 59–88. New York: Wiley.

Skinner, C. J., D. Holt, and T. M. F. Smith. 1989. *Analysis of Complex Surveys*. New York: Wiley.

Sloane, N. J. A. 2004. A Library of Hadamard Matrices. http://www2.research.att.com/~njas/hadamard/.

Thompson, M. E. 1997. *Theory of Sample Surveys*. London: Chapman & Hall.

Valliant, R. 1996. Discussion of "Resampling methods in sample surveys" by J. Shao. *Statistics* 27: 247–251.

Valliant, R., J. M. Brick, and J. A. Dever. 2008. Weight adjustments for the grouped jackknife variance estimator. *Journal of Official Statistics* 24: 469–488.

West, B. T., P. Berglund, and S. G. Heeringa. 2008. A closer examination of subpopulation analysis of complex-sample survey data. *Stata Journal* 8: 520–531.

White, H. 1982. Maximum likelihood estimation of misspecified models. *Econometrica* 50: 1–26.

Wolter, K. M. 2007. *Introduction to Variance Estimation*. 2nd ed. New York: Springer.

Wu, C. F. J. 1991. Balanced repeated replications based on mixed orthogonal arrays. *Biometrika* 78: 181–188.

Yeo, D., H. Mantel, and T.-P. Liu. 1999. Bootstrap variance estimation for the National Population Health Survey. In *Proceedings of the Survey Research Methods Section*, 778–785. American Statistical Association.

Yung, W. 1997. Variance estimation for public use files under confidentiality constraints. In *Proceedings of the Survey Research Methods Section*, 434–439. American Statistical Association.

About the author

Stanislav (Stas) Kolenikov is a statistical consultant and an adjunct assistant professor in the Department of Statistics at the University of Missouri in Columbia, MO. His research interests include statistical methods in social sciences, with a focus on structural equation models, microeconometrics, and analysis of complex survey data. He started using Stata in 1998 with version 5.

Appendix. Commonly used notation

The generic datum x_{hij} denotes the measurement on variable x taken on the jth observation in the ith PSU in the hth stratum.

f	sampling fraction: $f = n/N$
f_h	sampling fraction in stratum h: $f_h = n_h/N_h$
$h = 1, \ldots, L$	stratum index
$i = 1, \ldots, n_h$	PSU index within strata
j	observation index within PSU
L	number of strata
m_h	bootstrap sample size; the number of PSUs taken from stratum h to form a bootstrap replicate
$m_{hi}^{(*r)}$	bootstrap frequency; the number of times unit hi is sampled in the rth replicate
n	total sample size; in complex surveys, the total number of PSUs in the sample: $n = \sum_{h=1}^{L} n_h$
N	population size; in complex surveys, the total number of PSUs in the population: $N = \sum_{h=1}^{L} N_h$
n_h	sample size in stratum h; in complex surveys, the number of PSUs taken from stratum h
N_h	population size in stratum h; in complex surveys, the number of PSUs in stratum h
R	the number of replicates; the number of replicate weights for the mean bootstrap
$T(x)$	population total: $T(x) = \sum_h \sum_i \sum_j x_{hij}$
$t(x)$	estimate of the population total $T(x)$
$v\left(\widehat{\theta}\right)$	estimator of variance $V\left(\widehat{\theta}\right)$
$v_m\left(\widehat{\theta}\right)$	estimator of variance $V\left(\widehat{\theta}\right)$ obtained by method m; the methods include linearization L, the jackknife J, BRR, or the rescaling bootstrap RBS
$V\left(\widehat{\theta}\right)$	(design) variance of the estimate $\widehat{\theta}$ with respect to the sampling distribution
W_h	fraction of stratum h in population: $W_h = N_h/N$
w_{hij}	sampling weight of unit hij
$w_{hij}^{(r)}$	replicate weight of unit hij in the rth replicate
θ	population parameter, such as total, mean, ratio, or regression coefficient
$\widehat{\theta}$	parameter estimate obtained from survey data
$\widehat{\theta}^{(r)}$	parameter estimate obtained in the rth replicate

The Stata Journal (2010)
10, Number 2, pp. 200–214

Optimal power transformation via inverse response plots

Charles Lindsey
Texas A & M University
Department of Statistics
College Station, TX
lindseyc@stat.tamu.edu

Simon Sheather
Texas A & M University
Department of Statistics
College Station, TX
sheather@stat.tamu.edu

Abstract. We present a new Stata command, `irp`, that generates the inverse response plot (Cook and Weisberg, 1994, *Biometrika* 81: 731–737) of a response on its predictors. Using the inverse response plot, an appropriate scaled power transformation for the positive response variable can be found so that the transformed response mean is linear in the predictors. The optimal transformation is displayed in the plot, as are user-specified guesses. By using the graphical display, the user may determine whether an appropriate transformation exists as well as determine its value. We demonstrate the `irp` command using both a generated and a real example.

Keywords: st0188, irp, regress, scaled power transformation

1 Theory/motivation

When investigating the regression of the positive variable y on $x_1, \ldots, x_p$, one may discover or suspect a nonlinear relationship between the response and the predictors. To overcome this difficulty, one may decide to transform y or its predictors so that the transformed variables have a linear relationship via the invertible power transformation t. There are methods for this procedure that are already implemented in Stata (see [R] **boxcox**). These methods are useful and powerful, but they mostly rely on numeric output.

By using an inverse response plot, we provide a concise and intuitive plot that demonstrates the effectiveness of the transformation by the closeness of the transformation's curve to points in a scatterplot. This plot may significantly ease comparison between different transformations and vividly demonstrate the adequacy of the optimal transformation.

Cook and Weisberg (1994) used results in Li and Duan (1989) to develop the inverse response plot. To use an inverse response plot to estimate t, Cook and Weisberg showed that the following results should also hold.

For unknown constant matrices $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_p)$, $\Lambda_{p\times p} = (\lambda_{ij})$, $\Phi_{p\times p} = (\phi_{ij})$, and zero mean random variable ϵ, where ϵ is independent of the $x_0, \ldots, x_p$,

$$t(y) = \alpha_0 + \alpha_1 x_1 + \cdots + \alpha_p x_p + \epsilon \tag{1}$$

$$p \neq 1 \qquad E(x_i \,|\, x_j) = \lambda_{ij} + \phi_{ij} x_j, \qquad i, j = 1, \ldots, p \tag{2}$$

$$p = 1 \qquad x_1 \text{ is elliptically symmetric} \tag{3}$$

Both (2) and (3) are implied by (multivariate) normality of the predictors. Equation (2) need only apply approximately for the inverse response plot to work in practice. Equation (3) is not satisfied if the predictor is skewed. A skewed predictor is easy to check for with a kernel density plot (see [R] **kdensity**) or a Q–Q plot (see [R] **diagnostic plots**). Similarly, checking the linear relation of (2) with a matrix plot (see [G] **graph matrix**) is simple. Equation (1) is difficult to check before using the inverse response plot, because (1) depends directly on t, which we use the inverse response plot to estimate. If we find that (2) and (3) are met, and we obtain a satisfactory estimate of t from the inverse response plot, then (1) is implied.

Supposing that (2) and (3) are met, we estimate t by plotting the fitted values $\widehat{y} = \widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \cdots + \widehat{\beta}_p x_p$ for the regression of y on $x_1, \ldots, x_p$ on the vertical axis versus the actual response values y on the horizontal axis. We fit scaled power curves to this scatterplot. The power of the curve suggests the functional form of t. The curve fitting is done by transforming y with a scaled power transformation, and then scaling and relocating the transformed values so that they are an optimal fit for the scattered points.

Essentially, we regress $\widehat{y}$ on the transformed y. Then we plot a curve through the predictions of $\widehat{y}$ from the transformed y. This methodology of predicting t was developed by Cook and Weisberg (1994) based on results of Li and Duan (1989). In particular, Cook and Weisberg show that this method works even though the fitted values are from an invalid regression model.

It will be instructive to look at a sample inverse response plot (see figure 1). Here y is the cube of its predictor and error term. (We will return to this example in the next section, where all details will be given.) The first entry in the legend corresponds to the optimal choice of transformation. We have $y^{1/3} = x + \epsilon$, so this choice is quite logical. The 0 curve shows how $\log(y)$ approximates $y^{1/3}$. The 1 curve shows how poorly y approximates $y^{1/3}$.

(*Continued on next page*)

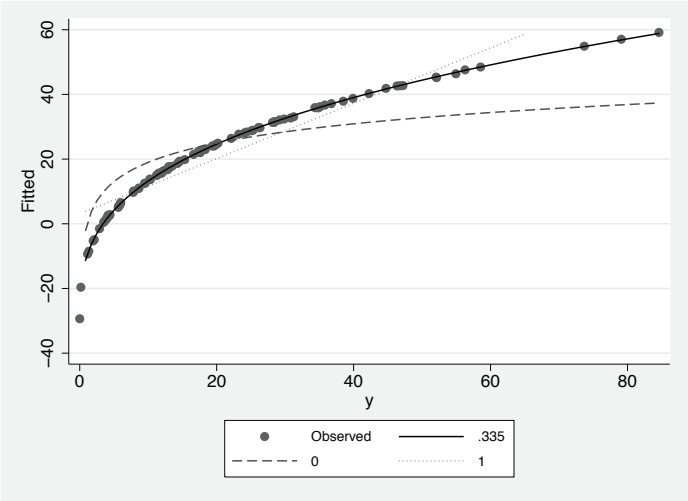

Figure 1. Generated example $y = (x + \epsilon)^3$

In this example, t was an unscaled power transformation. Sometimes using such a transformation will switch the direction of the relationship between the response and predictors. For example, suppose y and x are directly related. When we transform y to $1/y$, we make x and the new y inversely related.

We define a scaled power transformation as

$$\psi_s(y, \lambda) = \begin{pmatrix} \frac{y^\lambda - 1}{\lambda}, & \text{if} \lambda \neq 0 \\ \log y, & \text{if} \lambda = 0 \end{pmatrix}$$

Scaled power transformations preserve the direction of associations that the transformed variable had with other variables. So scaled power transformations will not switch or erase collinear relationships of interest. The power of the optimal scaled power transformation should in principle be identical to the power of the optimal unscaled transformation.

Finding the closest fitting scaled power curve (and thus the best estimate of t) is equivalent to finding the scaled power transformation that, when performed on y, minimizes the residual sum of squares for the regression of $\widehat{y}$ on the transformed y. The optimal transformation is found in `irp` via a numerical optimization on this residual sum of squares.

The power of the closest fitting scaled power curve may be quite esoteric to the user or the subject matter experts that he or she is helping. So it is often necessary to try more than one transformation, picking the curve with the best fit and most plausibility with respect to the subject matter of the data. We visually compared three transformations in our first example. The process was intuitive and simple.

We can augment a scaled power transformation by multiplying the transformed variable by the original's geometric mean, $\text{gm}(Y) = \exp\big(1/n \sum_{i=1}^{n} \log y_i\big)$ raised to the $1-\lambda$ power. This transformation would maintain the scale of the transformed variables. We denote this as a modified scaled power transformation:

$$\psi_M(y, \lambda) = \text{gm}(Y)^{1-\lambda} \psi_s(y, \lambda)$$

Box and Cox (1964) make the additional assumption that the $\psi_M(y, \lambda)$ is normal with a mean that is linearly determined by $x_1, \ldots, x_p$ for the true transformation power λ. Under Box and Cox's assumption, the optimal transformation power λ minimizes the residual sum of squares for the regression of $\psi_M(y, \lambda)$ on $x_1, \ldots, x_p$. Using this method does allow for statistical inference on λ through likelihood methods, but it also imposes distributional assumptions.

2 Use and a generated example

The `irp` command is straightforward to use. We will demonstrate this as we revisit the generated example that produced figure 1.

In this example, we generate predictor `x` distributed as $N(2.6, 0.75^2)$ and error term ϵ distributed as $N(0, 0.01^2)$. Our response variable `y` $= (\texttt{x} + e)^3$. See figure 2.

```
. set obs 100
obs was 0, now 100
. set seed 1234
. generate x = .75*invnormal(runiform()) + 2.6
. generate e = .01*invnormal(runiform())
. generate y = (x + e)^3
```

(*Continued on next page*)

```
. twoway scatter y x
```

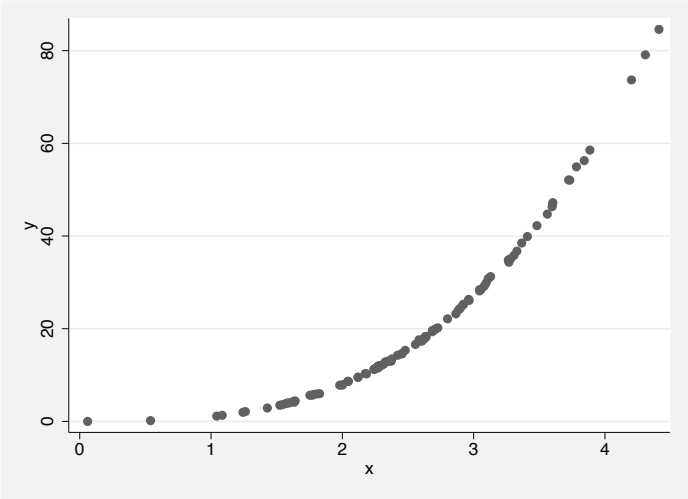

Figure 2. x versus y, $y = (x + \epsilon)^3$

Clearly, y and x lack a linear relationship. So linear regression of y on x is ill-advised. We will see if we can use an inverse response plot to transform y to linearize its relationship with x. See figure 3.

```
. summarize y
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
           y |        100    21.23332     17.5544   .0000891   84.59297

. swilk x
                    Shapiro-Wilk W test for normal data

    Variable |   Obs       W           V         z       Prob>z
-------------+--------------------------------------------------
           x |   100   0.99201       0.660    -0.922    0.82175
```

```
. qnorm x
```

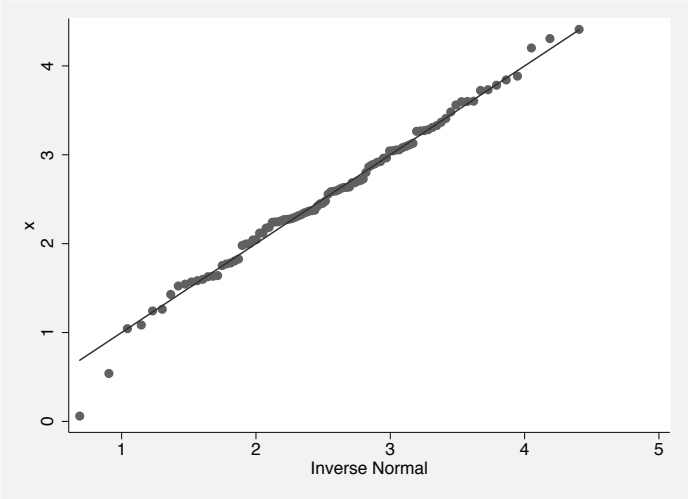

Figure 3. `x` normal quantile plot

The response `y` is positive. The `swilk` and `qnorm` commands show that `x` is consistent with a normal distribution. So we can use an inverse response plot to determine a scaled power transformation for `y` so that the new `y` is linear in `x`.

The `irp` command has the following syntax:

`irp` *depvar indepvars* [*if*] [*in*] [`, optimum try(`*numlist*`) old(`*matname*`) generate` *twoway_options*]

The *depvar* variable is the response variable of the inverse response plot, and *indepvars* are the predictors. A subset of the data may be specified with the optional *if* condition or *in* range. The `optimum` option instructs `irp` to find the best scaled power transformation for *depvar* by a numerical optimization. The `try()` option allows the user to specify a list of transformation powers to be examined in the plot. The `old()` option allows the user to specify a matrix containing calculations from a previous execution of `irp` that will be incorporated into the inverse response plot. The `generate` option outputs the transformed response variables for the given input powers to the data. Finally, additional graphical options are allowed to be specified in *twoway_options*.

Now let's test `irp` on our example. See figure 1 for the graphical output.

(*Continued on next page*)

```
. irp y x, opt try(0 1)
```

Response	y
Fitted	20.34*x + -30.58

Optimal Power	.3353386

Power	RSS(F | R)
.3353386	4.263288
0	8082.853
1	3787.411

```
. return list
scalars:
            r(optimum) =  .335338566748325
matrices:
            r(tranres) :  3 x 4
. matrix list r(tranres)
r(tranres)[3,4]
         Power          RSS   Intercept       Slope
r1   .33533857   4.2632882  -30.373909   6.7538795
r1           0    8082.853  -.80120822   8.5953352
r2           1   3787.4108   3.0839681   .85475809
```

The `irp` command provides the residual sum of squares for the regression of the fitted values on each scaled power transformation attempted on the response. Under the default display of `irp`, the residual sum of squares are shown under the `RSS( F | R )` column. The corresponding power transformation parameters are given in the `Power` column. This quantitative information can help the user understand the magnitude of fit difference between transformations.

In the returned results, the slopes and intercepts for each regression of the fitted values on a transformed response are also provided. If the user wishes to compare additional transformation powers without reperforming any calculations, then the user can pass the name of a matrix with the same format as `r(tranres)` into the `old()` option. This option is only useful when the user wants to redraw an inverse response plot that took considerable computation to draw the first time. We will demonstrate the use of the `old()` option in our next example, in section 3.

In this example, the additional quantitative information provided by `irp` strongly corroborates the notion that 1/3 is an appropriate transformation power.

We will put 1/3 into the `try()` list and reexecute `irp` to compare the fit of 1/3 and the optimal transformation. See figure 4.

```
. irp y x, opt try(.3333333)
```

Response	y
Fitted	20.34*x + -30.58

Optimal Power	.3353386

Power	RSS(F | R)
.3353386	4.263288
.3333333	4.36517

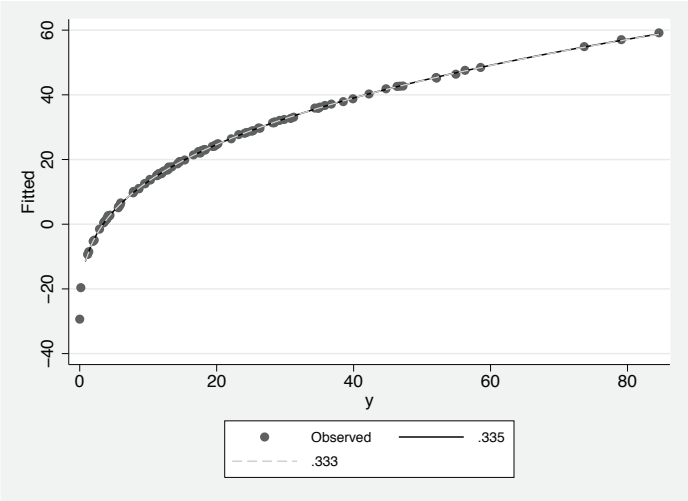

Figure 4. 1/3 versus optimum inverse response plot

The fits are nearly identical. We will transform y to $\psi_s(\mathtt{y}, 1/3)$ and recheck the linearity of y in x. See figure 5.

```
. generate trany = (y^(1/3) - 1)/(1/3)
. summarize trany
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       trany |        100    4.641468    2.390982  -2.865988    10.1694
```

(*Continued on next page*)

```
. twoway scatter trany x
```

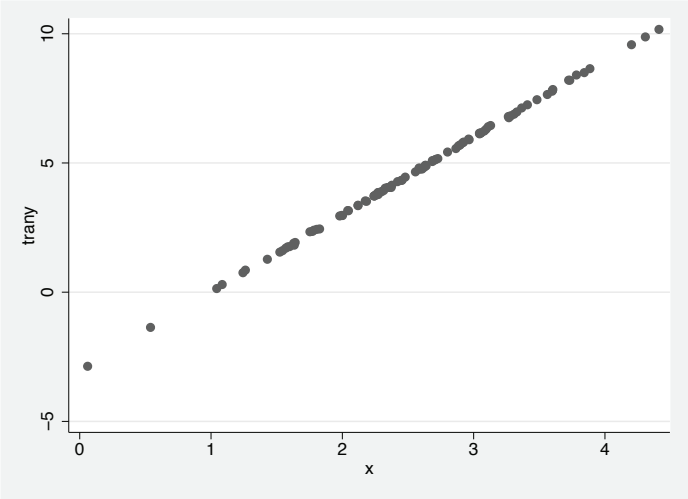

Figure 5. `x` versus $\psi_s(\mathtt{y}, 1/3)$, $\mathtt{y} = (\mathtt{x} + \epsilon)^3$

We could have used an unscaled power transformation in this case without disturbing the direction of association between `y` and `x`, but it was instructive to practice implementation of the scaled power transformation. We will have strong reason to use one in our next example.

3 Real example

Consider the UCI Machine Learning Repository (1993) dataset `auto-mpg.dta`. The data contain information on individual automobiles. Of interest is the regression of miles per gallon, `mpg`, on the car's weight, `wt`, and horsepower, `hp`. A matrix plot of the three variables reveals a problem and the path to a solution. See figure 6.

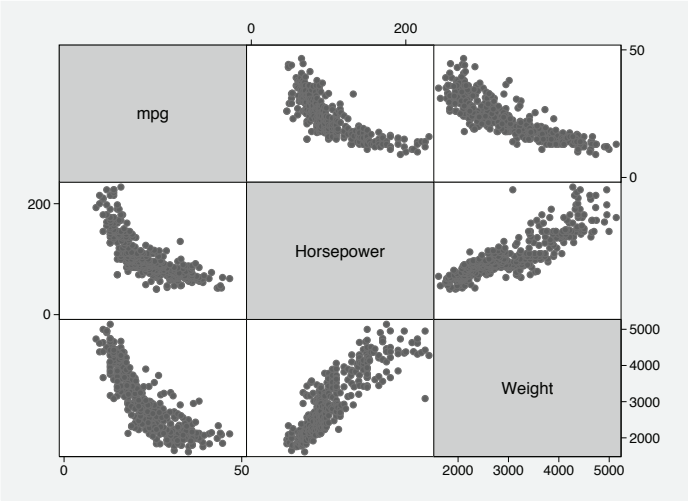

Figure 6. Auto 1993 example, matrix plot

The `mpg` variable appears to have a nonlinear relationship with both `hp` and `wt`, but the two predictors appear to have an approximately linear relationship. Therefore (2) for the use of an inverse response plot is satisfied.

We will now use `irp` to render an inverse response plot. We are going to be conservative and not guess any transformation powers. See figure 7.

```
. irp mpg hp wt, optimum
```

Response	mpg
Fitted	-.0473*hp + -.0058*wt + 45.64

Optimal Power	-.8776173

Power	RSS(F | R)
-.8776173	3149.255

(*Continued on next page*)

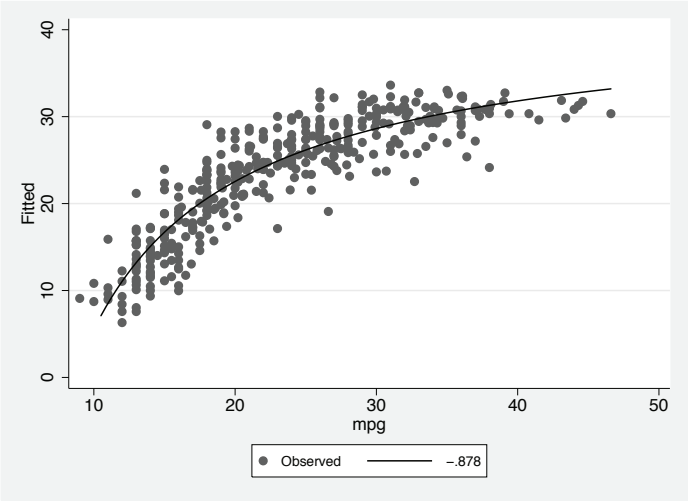

Figure 7. Auto 1993 example, inverse response plot optimum

```
. matrix b = r(tranres)
```

We stored the results of our optimum power calculation in the matrix `b`. Momentarily, we will use them again. Miles per gallon is often translated to gallons per mile via the reciprocal transformation 1/`mpg`. We will compare this transformation with our optimal transformation. See figure 8.

```
. irp mpg hp wt, try(-1) old(b) generate
```

Response	mpg
Fitted	-.0473*hp + -.0058*wt + 45.64

Optimal Power	Not Calculated/Re-Calculated

Power	RSS(F | R)
-1	3157.848
-.8776173	3149.255

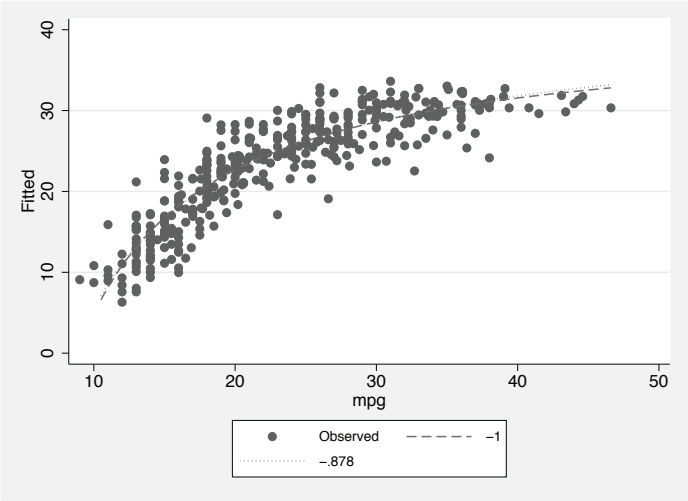

Figure 8. Auto 1993 example, inverse response plot optimum, −1

The reciprocal transformation is nearly as good as the optimal transformation. Because it makes more sense theoretically, we will use the reciprocal transformation instead of the optimal.

Note how the matrix `b` was used in the last `irp` call. We were able to save computation time by using the previously calculated results in `b`. This was not necessary; we could have executed `irp mpg hp wt, optimum try(-1)` and obtained the same results.

It may be tempting to just raise `mpg` to the −1 power and ignore whatever relationships `mpg` had before we transformed it. Under the reciprocal transformation, any direct relationships `mpg` had with other variables are now inverse relationships. Similarly, all inverse relationships `mpg` had with other variables are now direct relationships.

The following correlation matrices demonstrate what happens when we use a reciprocal transformation. The `ecc` variable is the acceleration capability of the automobile.

```
. correlate mpg hp wt acc
(obs=392)
```

	mpg	hp	wt	acc
mpg	1.0000			
hp	-0.7784	1.0000		
wt	-0.8322	0.8645	1.0000	
acc	0.4233	-0.6892	-0.4168	1.0000

```
. generate rmpg = 1/mpg
```

```
. correlate rmpg hp wt acc
(obs=392)
```

	rmpg	hp	wt	acc
rmpg	1.0000			
hp	0.8548	1.0000		
wt	0.8851	0.8645	1.0000	
acc	-0.4563	-0.6892	-0.4168	1.0000

It may be wise to maintain the direction of the variable relationships `mpg` holds so that we do not falsely take them for granted and make a mistake. We can easily maintain the direction by transforming `mpg` using the scaled power transformation $\psi_s(\texttt{mpg}, -1) = -1/\texttt{mpg} + 1$. We will generate a new variable using this formula and compare it with that produced by `irp` in the `irp1` variable.

The following correlation matrix demonstrates how the scaled power transformation maintains the directionality of `mpg` relationships and how the generated variable from `irp` perfectly matches with the scaled power transformation.

```
. describe irp1
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
irp1            float  %9.0g                  Transform Power = -1
. generate srmpg = -1/mpg + 1
. correlate mpg srmpg irp1 hp wt acc
(obs=392)
```

	mpg	srmpg	irp1	hp	wt	acc
mpg	1.0000					
srmpg	0.9359	1.0000				
irp1	0.9359	1.0000	1.0000			
hp	-0.7784	-0.8548	-0.8548	1.0000		
wt	-0.8322	-0.8851	-0.8851	0.8645	1.0000	
acc	0.4233	0.4563	0.4563	-0.6892	-0.4168	1.0000

So we will be careful and use the scaled power transformation to transform `mpg`. Now we should check that this transformation solves our nonlinearity problem. First, we will revisit our matrix plot from figure 6. See figure 9.

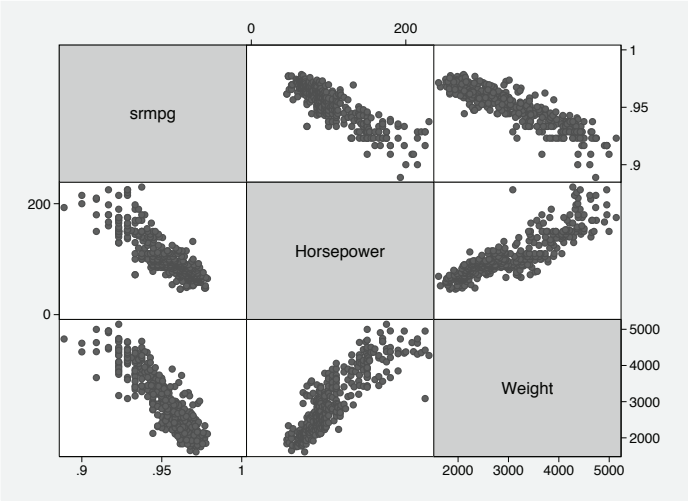

Figure 9. Auto 1993 example, matrix plot revisited

The relationships between `srmpg` (the transformed `mpg`), `hp`, and `wt` are all approximately linear. Our transformation was successful. We will conclude this example with a final graphic that demonstrates how the transformation we found via `irp` affected the fit of the regression of `mpg` on `hp` and `wt`; see figure 10. The first plot shows `mpg` versus its predicted values under initial regression, with no transformation. The second plot shows `mpg` versus its predicted values, after the transformation. Note the change in linearity.

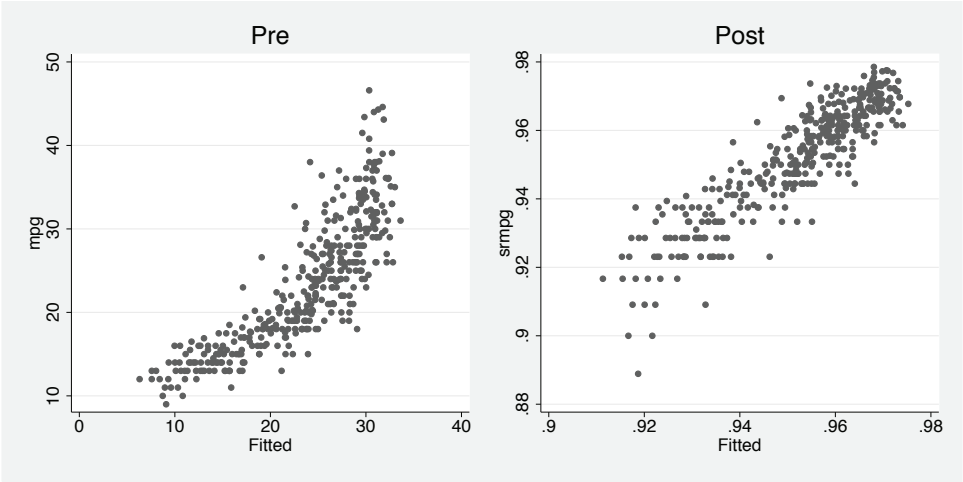

Figure 10. Auto 1993 example, fitted versus response

4 Conclusion

We have demonstrated the use of inverse response plots in response transformations to linearity. We used generated and real datasets. Both the theory and the practice of the method was explored.

The `irp` command was fully defined as a method for using inverse response plots in Stata. Its graphical and numeric output were demonstrated, and the process of fitting multiple inverse response plots to the same data was also shown.

5 References

Box, G. E. P., and D. R. Cox. 1964. An analysis of transformations. *Journal of the Royal Statistical Society, Series B* 26: 211–252.

Cook, R. D., and S. Weisberg. 1994. Transforming a response variable for linearity. *Biometrika* 81: 731–737.

Li, K.-C., and N. Duan. 1989. Regression analysis under link violation. *Annals of Statistics* 17: 1009–1052.

UCI Machine Learning Repository. 1993. http://mlearn.ics.uci.edu/databases/auto-mpg/.

About the authors

Charles Lindsey is a PhD candidate in statistics at Texas A & M University. His research is currently focused on nonparametric methods for regression and classification. He currently works as a graduate research assistant for the Institute of Science Technology and Public Policy within the Bush School of Government and Public Service. In the summer of 2007, he worked as an intern at StataCorp. Much of the groundwork for this article was formulated there.

Simon Sheather is a professor in and head of the Department of Statistics at Texas A & M University. Simon's research interests are in the fields of flexible regression methods, and nonparametric and robust statistics. In 2001, Simon was named an honorary fellow of the American Statistical Association. Simon is currently listed on http://www.ISIHighlyCited.com among the top one-half of one percent of all mathematical scientists, in terms of citations of his published work.

The Stata Journal (2010)
10, Number 2, pp. 215–225

Model fit assessment via marginal model plots

Charles Lindsey
Texas A & M University
Department of Statistics
College Station, TX
lindseyc@stat.tamu.edu

Simon Sheather
Texas A & M University
Department of Statistics
College Station, TX
sheather@stat.tamu.edu

Abstract. We present a new Stata command, `mmp`, that generates marginal model plots (Cook and Weisberg, 1997, *Journal of the American Statistical Association* 92: 490–499) for a regression model. These plots allow for the comparison of the fitted model with a nonparametric or semiparametric model fit. The user may precisely specify how the alternative fit is computed. Demonstrations are given for logistic and linear regressions, using the `lowess` smoother to generate the alternate fit. Guidelines for the use of `mmp` under different models (through `glm` and other commands) and different smoothers (such as `lpoly`) are also presented.

Keywords: st0189, mmp, regress, glm, lpoly, logit, logistic, marginal model plots

1 Theory/motivation

Graphical assessment of a model's fit can be an intuitive (even essential) tool in regression analysis. Ordinary least-squares linear regression allows powerful graphical assessment diagnostics through the model's residuals. In other forms of generalized linear model regression, this may not be the case. For example, when we are performing binary logistic regression, all the residuals normally used (Pearson, deviance, and response) will display a nonrandom and uninterpretable pattern when plotted against the model's predictors, even if the correct model has been fit. Bypassing the use of the residuals, Cook and Weisberg (1997) provide an alternative graphical assessment for regression model fit: marginal model plots. The assessment is performed as follows.

Suppose that we have k predictors. If all of them are continuous, we will produce $k + 1$ plots, one for each predictor and one for the $\beta'x$ linear forms. If a predictor is not reasonably continuous (more than two values), then we omit its plot. Each of the generated plots is called a marginal model plot. In each of the plots, a scatterplot is drawn, with the response on the vertical axis and the predictor or linear form on the horizontal axis.

The regression model's estimate of the response's mean, conditional on the predictor (linear form), is then computed at each value of the predictor (linear form). A line is passed through the estimated points. This is the model line. Now a nonparametric estimate (smooth) of the response's mean, conditional on the predictor (linear form), is computed at each value of the predictor (linear form). A new line (usually of a different color or pattern) is then passed through these points. This is the alternative line.

When we pick a good nonparametric estimator for generating the alternative line, we can assess the fit of the model by judging how closely the lines overlap. If the lines match each other closely in each of the generated plots, we conclude that our regression model is a good fit. If there is significant disparity between the lines in any of the plots, our regression model is not a good fit. Figure 1 shows a set of marginal model plots that demonstrate the good fit of a linear regression model. We will discuss this figure futher in section 2.1. The dashed line is the model line.

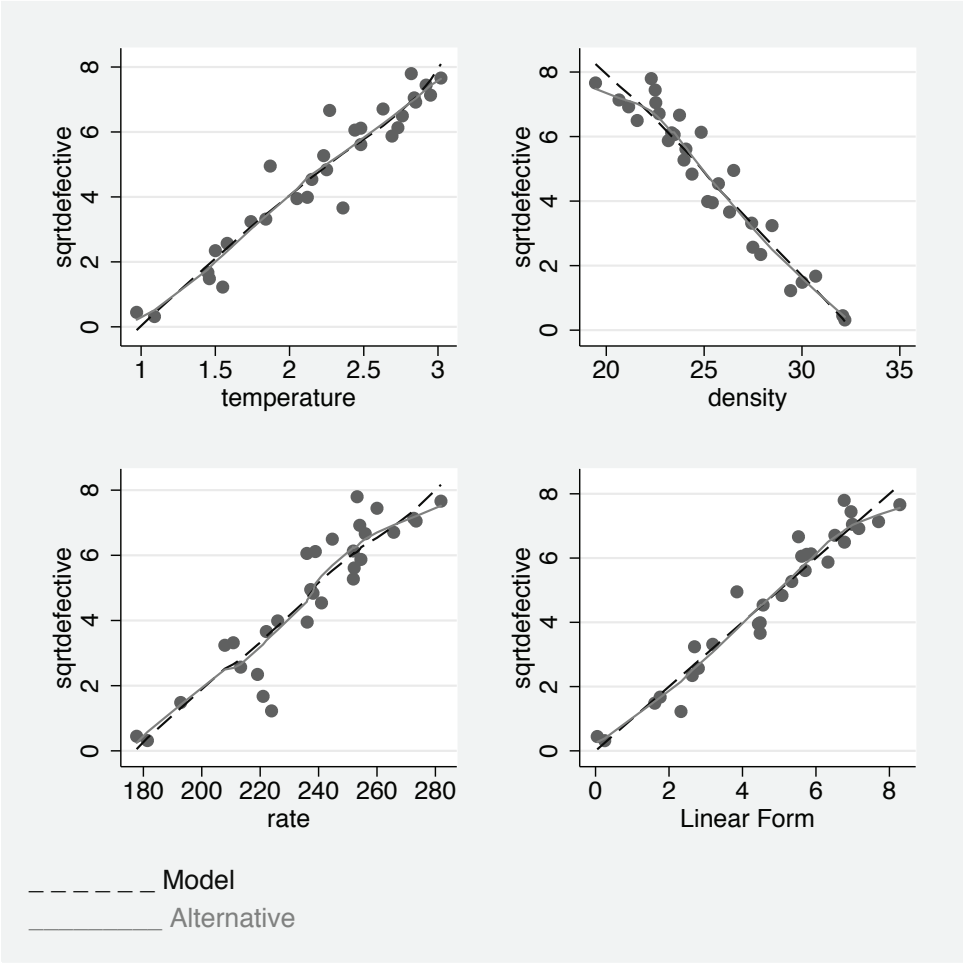

Figure 1. Marginal model plot example

Choosing a good nonparametric estimator is key to correctly use this method. There are many options. Cook and Weisberg (1997) use a lowess smoother to make their estimates. In this article, we will use the lowess smoother provided by Stata (see [R] **lowess**) with a tuning parameter $\alpha = 2/3$.

Estimation of the alternative line is completely dependent on the nonparametric estimator used, but there are interesting general results used in estimation of the model line. We will discuss these briefly.

Suppose that we observe the response y and the predictors $x = (x_1, \ldots, x_k)'$. We have as a regression model the generalized linear model $E_M(y \,|\, x) = g(\beta' x)$. To find the model line, we estimate $E_M(y \,|\, \alpha' x)$, where $\alpha' x$ is the linear form $\widehat{\beta}' x$ or a single predictor $x_1, \ldots, x_k$. Cook and Weisberg (1997) saw that for any α vector of the proper dimension,

$$E_M(y \,|\, \alpha' x) = E\left\{E_M(y \,|\, x) \,|\, \alpha' x\right\}$$

We estimate $E_M(y \,|\, x)$ by $g(\widehat{\beta}' x)$. So the estimator of $E_M(y \,|\, \alpha' x)$ is an estimator of $E\left\{g(\widehat{\beta}' x) \,|\, \alpha' x\right\}$. The closed form solution of this expectation for an arbitrary α is probably unknown or rather complicated. We can estimate it by using the nonparametric estimator that generates the alternative line.

So we will use the nonparametric estimator twice, once for the alternative line $(y \,|\, \alpha' x)$ and again for the model line $\left\{g(\widehat{\beta}' x) \,|\, \alpha' x\right\}$. Again note that $\alpha' x$ is $\widehat{\beta}' x$ or a single predictor $x_1, \ldots, x_k$.

Generally, the closer the estimated line is to the points in the plot, the more accurate the estimation method is. So one may extend the methodology and use a semiparametric estimate or even a parametric estimate to generate the alternative line. The marginal model plots can then be used to assess whether the alternative estimator is as good as the model estimator, or vice versa.

We present a new Stata command, `mmp`, that creates marginal model plots. With very mild restrictions, it allows users to construct a traditional marginal model plot using any nonparametric smoother and generalized linear model that they wish. In addition, a user may try some of the less orthodox strategies already presented if desired.

We will restrict ourselves to linear and logistic regressions, because the appropriateness of the lowess smoother with tuning parameter $\alpha = 2/3$ is well documented.

In this article, we are only concerned with estimating the mean of the response given the predictors. We note that Cook and Weisberg (1997) extended the marginal model plot method for variance estimation as well. We also emphasize that our method is only appropriate after running generalized linear model estimation commands and that the input sample should have independent observations.

(*Continued on next page*)

2 Use and examples

`mmp` is to be executed after an estimation command that performs a regression. The `mmp` command has the following syntax:

`mmp, mean(`*string*`) smoother(`*string*`)` [`smooptions(`*string*`) linear predictors varlist(`*varlist*`) generate indgoptions(`*string*`) goptions(`*string*`)`]

With the `mean()` option, the user informs `mmp` how it should use the `predict` command to generate the estimated response mean. For example, if the user specifies `mean(xb)`, then `mmp` will generate the estimated response mean by calling `predict, xb`.

The `smoother()` option tells `mmp` which nonparametric estimation (smoothing) command to use for generating the alternative and model lines. The only restriction on the smoothing command is that it must have a `generate()` option that takes a single variable (where the smoothed values are stored); for example, the `lowess` command has a `generate()` option and so is appropriate to specify in `smoother()`. Additional options for the smoothing command are passed into the `smooptions()` option.

When `linear` is specified, a marginal model plot will be generated for the $\widehat{\beta}'x$ linear forms. If `predictors` is specified, then a marginal model plot is generated for each predictor. Marginal model plots for single predictors (or even unrelated variables) can be generated through placement in the optional *varlist* in the `varlist()` option.

The `generate` option makes `mmp` save the lowess estimates for the model and alternative lines as variables for each of the produced plots. If a plot is produced for predictor `x`, then variables `x_model` and `x_alt` are added to the data. If the `linear` option is specified to draw a plot for the linear form, then variables `linform_model` and `linform_alt` are added to the data.

The last two options of `mmp` concern the visual presentation of the plots. Graphical options passed into `indgoptions()` affect the display of individual marginal model plots, while options passed into `goptions()` affect the display of the combined array of marginal model plots.

We will now demonstrate the use of these options with some examples.

2.1 Ordinary least-squares example: Defective parts

First, we investigate the regression that yielded figure 1. These data were taken from Sheather (2009). The manufacturing criteria `temperature`, `density`, and `rate` are used to predict the number of defective parts produced, `defective`. See figure 2.

```
. use defects
. regress defective temperature density rate
```

Source	SS	df	MS	Number of obs =	30
				F(3, 26) =	63.36
Model	9609.44261	3	3203.14754	Prob > F =	0.0000
Residual	1314.43141	26	50.5550544	R-squared =	0.8797
				Adj R-squared =	0.8658
Total	10923.874	29	376.685311	Root MSE =	7.1102

defective	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
temperature	16.07792	8.294106	1.94	0.063	-.9708617	33.12669
density	-1.827292	1.497068	-1.22	0.233	-4.90456	1.249976
rate	.1167321	.1306268	0.89	0.380	-.1517751	.3852394
_cons	10.32437	65.92648	0.16	0.877	-125.1894	145.8382

```
. mmp, mean(xb) smoother(lowess) smooptions(bwidth(.6666667)) predictors linear
> goptions(xsize(10) ysize(10)) generate
```

(*Continued on next page*)

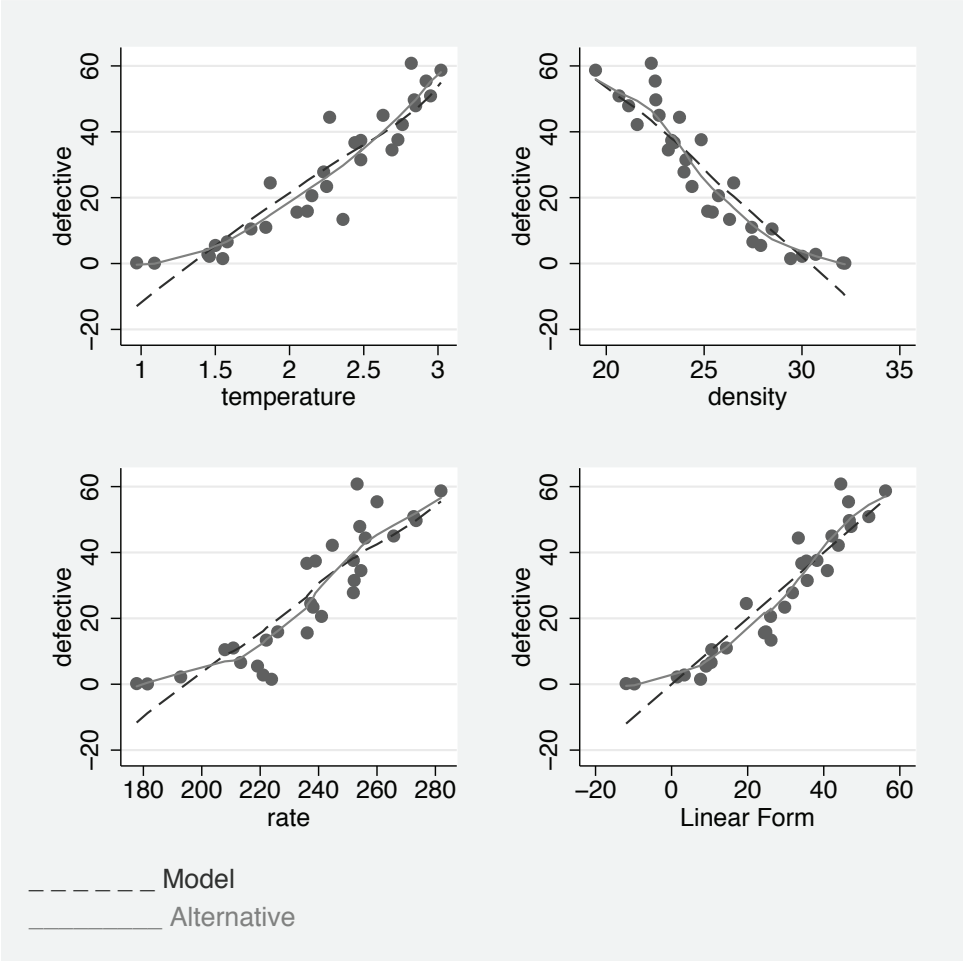

Figure 2. Defective marginal model plots

In the code, we specified that the mean of `defective` be estimated with `predict, xb`. The smoother would be `lowess` with a bandwidth of 2/3. We also specified that the resulting graphic would be square with a width of 10 centimeters.

The marginal model plots show us that the model is not perfect. When we examine the first plot, we see a quadratic trend to the fit. Further investigation in Sheather (2009) shows that it is appropriate to transform `defective`. The regression with $\texttt{defective}^{1/2}$ yielded the well-fitting marginal model plots in figure 1. We used the `generate` option in this example, so we will use the `summarize` command (see [R] **summarize**) to compare the `*_model` and `*_alt` variables. As suggested by the plots, the model estimates and the alternative estimates have notable differences.

```
. summarize temperature_model temperature_alt density_model density_alt
> rate_model rate_alt linform_model linform_alt
```

Variable	Obs	Mean	Std. Dev.	Min	Max
temperatur~l	30	27.18532	18.28548	-12.95191	54.9467
temperatur~t	30	27.62225	18.2362	-.3680072	58.10513
density_mo~l	30	27.0847	17.79125	-9.657054	55.92059
density_alt	30	27.34462	17.56772	-.214293	55.99818
rate_model	30	27.21515	17.05721	-11.64078	55.50286
rate_alt	30	27.44537	17.01681	-.6469474	56.58284
linform_mo~l	30	27.14333	18.2033	-11.95628	56.24563
linform_alt	30	27.49958	18.17563	-.4954669	57.07471

2.2 Logistic example: Michelin

Now we will try a logistic regression example. Another example in Sheather (2009) predicts the log odds of inclusion of a French restaurant in New York City in the Michelin restaurant guide, `inmichelin`, with the cost of a standard dinner at the restaurant, `cost`, and scores from the Zagat restaurant guide of the restaurant criteria, `decor`, `food`, and `service`. See figure 3.

```
. use michelin, clear
. logit inmichelin food decor service cost, nolog
Logistic regression                             Number of obs   =        164
                                                LR chi2(4)      =      77.39
                                                Prob > chi2     =     0.0000
Log likelihood = -74.198473                     Pseudo R2       =     0.3428
```

inmichelin	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
food	.4048471	.131458	3.08	0.002	.1471942	.6624999
decor	.0999727	.0891936	1.12	0.262	-.0748436	.274789
service	-.1924241	.1235695	-1.56	0.119	-.4346159	.0497677
cost	.0917195	.031753	2.89	0.004	.0294848	.1539542
_cons	-11.19745	2.308961	-4.85	0.000	-15.72293	-6.671971

```
. mmp, mean(pr) smoother(lowess) smooptions(bwidth(.6666667)) predictors linear
> goptions(xsize(9) ysize(10))
```

(*Continued on next page*)

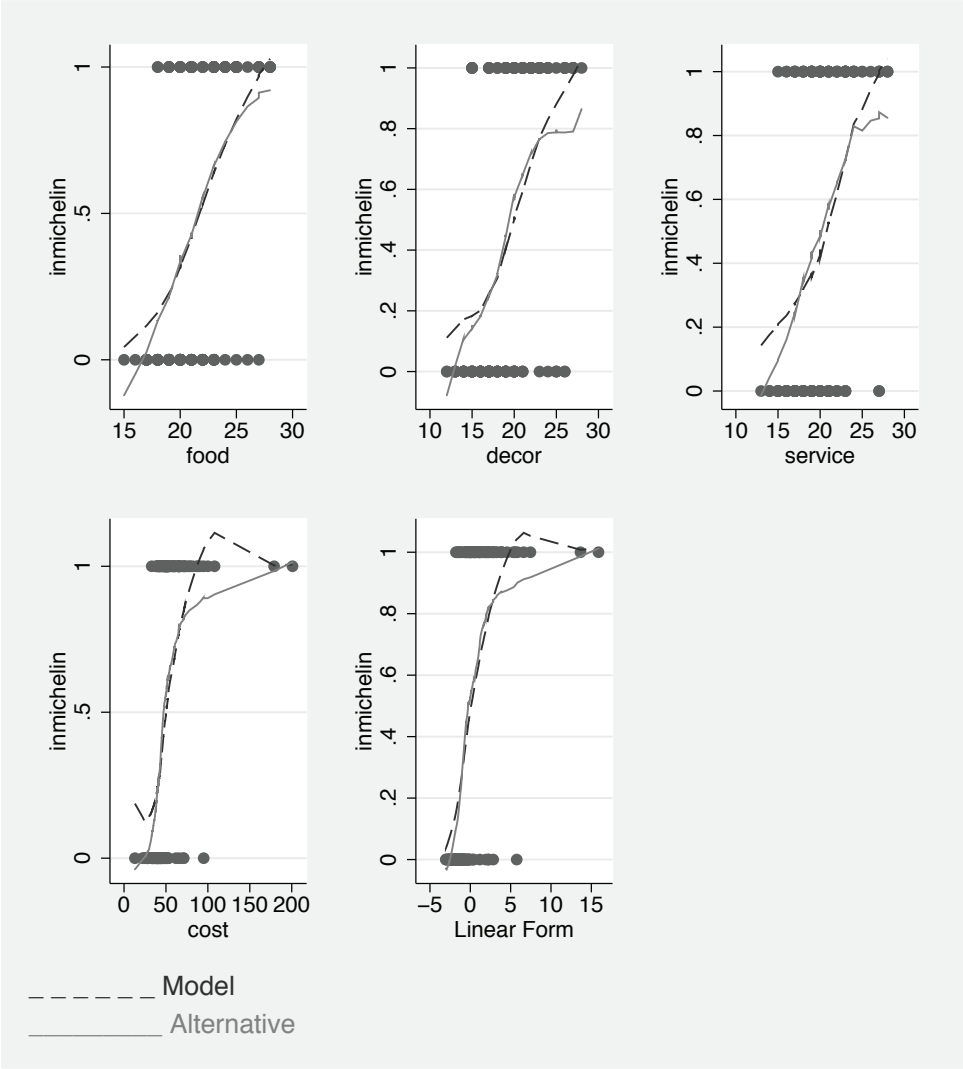

Figure 3. Michelin marginal model plots

Here we specified `mean(pr)`. If we had used `mean(xb)`, we would have estimated the log odds of Michelin inclusion. The marginal model plots for our model indicate that it does not fit particularly well. Note how the model line jumps above the value 1 for the predictor `cost` and the linear form. Our smoother does not truncate its output so that it must fall between 0 and 1. If the lowess estimate of the conditional mean exceeds 1, the excessive number is reported. Here the actual points used to calculate the estimate will of course not exceed 1, but the trend they suggest to the lowess calculation may lead to a surprisingly high value. As detailed in Sheather (2009), after some diagnostic

checks that examine the conditional distributions of the predictors under `inmichelin`, we find a superior model. This superior model is obtained by adding an interaction term for `service` and `decor` and the natural logarithm of `cost` to the model. These marginal model plots indicate a good fit for our model. See figure 4.

```
. generate srvdr = service*decor
. generate lncost = ln(cost)
. logit inmichelin food decor service cost lncost srvdr, nolog
Logistic regression                               Number of obs   =        164
                                                  LR chi2(6)      =      95.97
                                                  Prob > chi2     =     0.0000
Log likelihood = -64.910085                       Pseudo R2       =     0.4250

------------------------------------------------------------------------------
  inmichelin |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        food |   .6699594    .182764     3.67   0.000     .3117486    1.02817
       decor |   1.297883   .4929856     2.63   0.008     .3316491    2.264117
     service |   .9197059   .4882945     1.88   0.060    -.0373338    1.876746
        cost |  -.0745649   .0441641    -1.69   0.091    -.1611249    .0119951
      lncost |   10.96401   3.228443     3.40   0.001      4.63638    17.29164
       srvdr |  -.0655087   .0251228    -2.61   0.009    -.1147485    -.016269
       _cons |  -70.85311   15.45785    -4.58   0.000    -101.1499   -40.55628
------------------------------------------------------------------------------
. mmp, mean(pr) smoother(lowess) varlist(food decor service cost)
> smooptions(bwidth(.6666667)) linear goptions(xsize(9) ysize(10))
```

(*Continued on next page*)

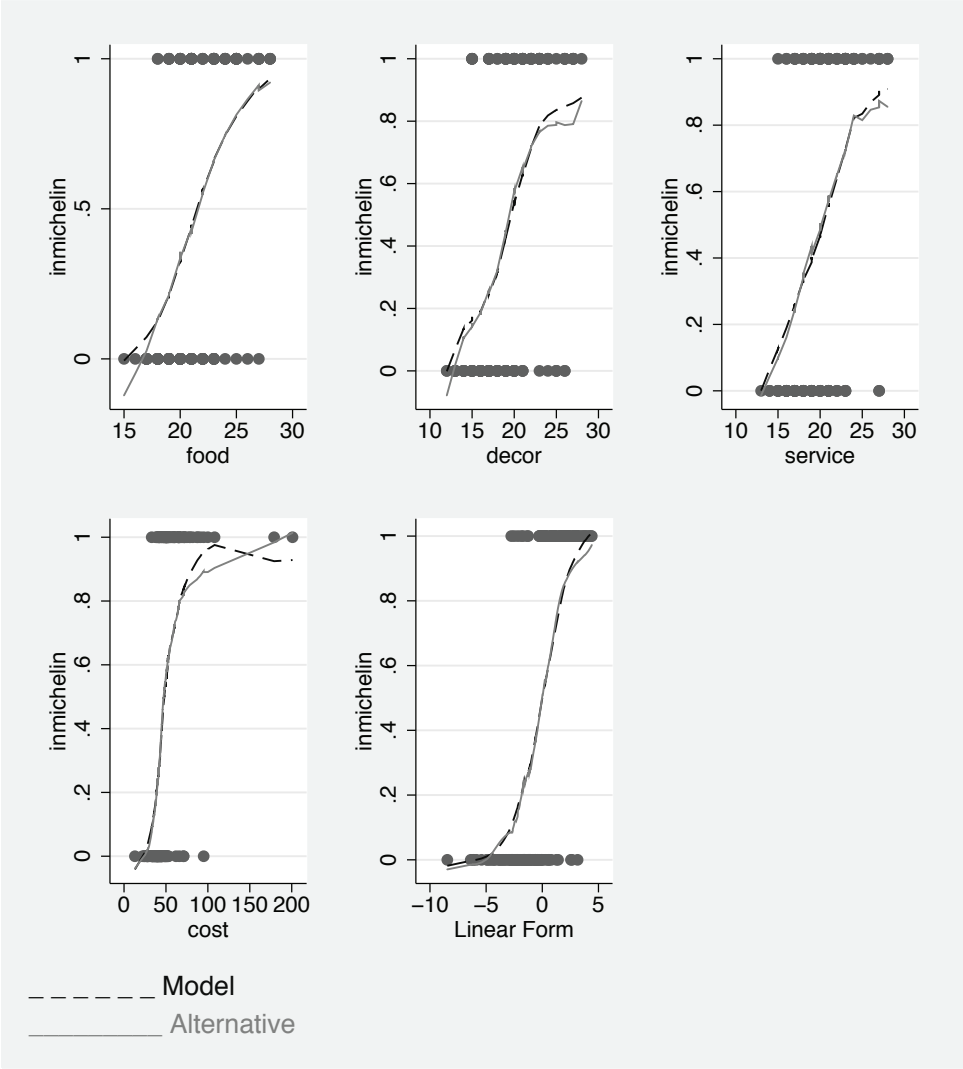

Figure 4. Michelin marginal model plot `srvdr`, log(`cost`)

3 Conclusion

Both the theory and the practice of marginal model plots have been explained in this article. We have demonstrated the use of marginal model plots in both linear and logistic regressions. The `mmp` command was fully defined as a method for using marginal model plots in Stata.

4 References

Cook, R. D., and S. Weisberg. 1997. Graphics for assessing the adequacy of regression models. *Journal of the American Statistical Association* 92: 490–499.

Sheather, S. J. 2009. *A Modern Approach to Regression with R*. New York: Springer.

About the authors

Charles Lindsey is a PhD candidate in statistics at Texas A & M University. His research is currently focused on nonparametric methods for regression and classification. He currently works as a graduate research assistant for the Institute of Science Technology and Public Policy within the Bush School of Government and Public Service. In the summer of 2007, he worked as an intern at StataCorp. Much of the groundwork for this article was formulated there.

Simon Sheather is a professor in and head of the Department of Statistics at Texas A & M University. Simon's research interests are in the fields of flexible regression methods, and nonparametric and robust statistics. In 2001, Simon was named an honorary fellow of the American Statistical Association. Simon is currently listed on http://www.ISIHighlyCited.com among the top one-half of one percent of all mathematical scientists, in terms of citations of his published work.

The Stata Journal (2010)
10, Number 2, pp. 226–251

Analyzing longitudinal data in the presence of informative drop-out: The jmre1 command

Nikos Pantazis
Department of Hygiene, Epidemiology, and Medical Statistics
University of Athens Medical School
Athens, Greece
npantaz@med.uoa.gr

Giota Touloumi
Department of Hygiene, Epidemiology, and Medical Statistics
University of Athens Medical School
Athens, Greece
gtouloum@med.uoa.gr

Abstract. Many studies in various research areas have designs that involve repeated measurements over time of a continuous variable across a group of subjects. A frequent and serious problem in such studies is the occurrence of missing data. In many cases, missing data are caused by an event that leads to a premature termination of the series of repeated measurements on some subjects. When the probability of the occurrence of this event is related to the subject-specific underlying trend of the variable of interest, this missingness process is called informative censoring or informative drop-out. Standard likelihood-based methods (for example, linear mixed models) fail to give consistent estimates. In such cases, one needs to apply methods that simultaneously model the observed data and the missingness process. In this article, we review a method proposed by Touloumi et al. (1999, *Statistics in Medicine* 18: 1215–1233) to adjust for informative drop-out in longitudinal data analysis. We also present the `jmre1` command, which can be used to fit the proposed model. The estimation method combines the restricted iterative generalized least-squares method with a nested expectation-maximization algorithm. The method is implemented mainly using Stata's matrix programming language, Mata. Our example is derived from the epidemiology of the HIV infection.

Keywords: st0190, jmre1, jmre1_p, datajoint1, missing data, informative censoring, informative drop-out, longitudinal data

1 Introduction

In many research areas, some studies collect longitudinal data on a continuous response variable for each participating subject. One major drawback in such studies, where the main objective is to estimate the rate of change of the response variable, is that the series of repeated measurements is prematurely terminated for some subjects because of the occurrence of an event resulting in highly unbalanced datasets.

Several methods to analyze such unbalanced data are available, provided that missing data are missing at random. Following the terminology of Little and Rubin (2002), conditionally on covariates, missing data can be classified as missing completely at random, when the missingness mechanism does not depend upon the response vector $\mathbf{Y}$; missing at random (MAR), when the probability of nonresponse depends on the observed part of the response ($\mathbf{Y}_0$) but not on the unobserved part ($\mathbf{Y}_m$); and missing nonignorable, when the probability of nonresponse depends on the unobserved outcomes.

Wu and Carroll (1988) introduced the term *informative right-censoring* for the special case of nonignorable missing data where the hazard of the terminating event is related to the subject-specific underlying trend of the response variable. However, from now on, we will use the term *informative drop-out* instead of informative censoring to avoid confusion with the common use of the term *censoring* in the survival analysis literature. It should be noted though that in many cases, the event that causes the termination of longitudinal measurements in a nonignorable way is death or disease progression and not drop-out of some individuals from a study.

Laird (1988) discussed the impact of various missingness processes on inference about the response variable with the main conclusion being that likelihood-based approaches can provide valid inferences, ignoring the missingness process only when data are missing completely at random or MAR. However, serious biases can occur when missingness is nonignorable. In such cases, one needs to apply methods that simultaneously model the response variable and the missingness process. Little (1995) summarized various methods to jointly model the response variable and the informative drop-out process, and he outlined possible extensions. One specific class of such models was derived by extending the so-called parametric conditional linear model (Wu and Carroll 1988; Wu and Bailey 1989), assuming a linear random-effects model for the longitudinal measurements of the response variable and a lognormal model for the terminating event, linked by shared or jointly distributed random parameters (Schluchter 1992; Faucett and Thomas 1996; Touloumi et al. 1999).

To our knowledge, there is no official nor user-written Stata command for modeling longitudinal data subject to informative drop-out, with the exception of a model proposed by Diggle and Kenward (1994) and shared random-effects models, which can be fit using `gllamm` (Rabe-Hesketh, Skrondal, and Pickles 2002). In this article, we describe the joint multivariate random-effects (JMRE) model introduced by Touloumi et al. (1999), and we present the `jmre1` command. For illustrative purposes, we apply `jmre1` to a dataset with longitudinal measurements of an immunologic marker (CD4 cell count) taken over a sample of HIV infected individuals, where death or AIDS onset lead to premature termination of CD4 cell-count measurements in a nonignorable way (that is, informative drop-out).

2 Background

Let $\mathbf{Y}_i = (Y_{i1}, Y_{i2}, \ldots, Y_{in_i})^T$ be the sequence of measurements of the continuous response variable (called *marker* from now on) on subject i ($i = 1, 2, \ldots, m$). Let X_i be

the $n_i \times p$ design matrix associated with the fixed effects ($\mathbf{b}$) of the marker, and let Z_i be the corresponding $n_i \times \pi$ design matrix associated with the random effects ($\boldsymbol{\beta}_i$) of the marker. We assume the following linear mixed model for the longitudinal marker's measurements:

$$\mathbf{Y}_i = X_i\mathbf{b} + Z_i\boldsymbol{\beta}_i + \mathbf{e}_i$$

where $\mathbf{e}_i$ are the within-subject residuals, assumed to be independent and normally distributed with mean $\mathbf{0}$ and variance–covariance matrix $\sigma^2 I_{n_i}$, where I_n denotes the $n \times n$ identity matrix. In addition, we assume that the random effects or between-subjects residuals $\boldsymbol{\beta}_i$ follow the joint multivariate normal distribution with mean $\mathbf{0}$ and variance–covariance matrix Σ^m (superscripts m and d will be used appropriately where needed to denote marker or informative drop-out related parts of the model, respectively). In a typical scenario, Z consists of a column of ones and a column taking the values of the times of measurement so that there is a random intercept and a random coefficient of time (random slope). Similarly, X consists of the same two columns corresponding to a fixed intercept and a fixed slope. There may be additional columns in X corresponding to fixed effects of other covariates on intercept and/or slope. This kind of random effects model is often referred to as the linear growth-curve model or the Laird and Ware model (Laird and Ware 1982). Under the MAR assumption, the mixed model can be fit using Stata's `xtmixed` command.

Repeated measurements of the response variable are terminated either because of development of a terminating event or because of loss to follow-up or study termination. In the former case, the terminating event could be, for example, death. If the probability of death is related to the underlying marker's evolution, the event is considered informative (that is, informative drop-out). In the latter case, it is usually assumed that the resulting missing measurements are at random and thus ignorable in a maximum likelihood analysis. In this case, the actual time of the terminating event is not observed either. We treat these censored event times as noninformative according to the usual survival analysis assumption.

For each subject i, let T_i^s denote event time (for example, death time), and let C_i denote the censoring time (for example, study termination before the occurrence of the event). Note that other kinds of termination of the study may be viewed as informative. In general, T refers to the time to the event whose occurrence is assumed to be informative, whereas censoring refers to reasons for termination that are not considered as informative. The observed "survival" data consist of $T_i = \min(T_i^s, C_i)$ and an indicator variable, δ_i, taking the value of 1 if $T_i = T_i^s$ (that is, the terminating event is observed) and 0 otherwise (that is, the terminating event is not yet observed). To model event times, we used an accelerated failure-time lognormal model of the form

$$\log(T_i^s) = \mathbf{x}_i^{dT}\mathbf{b}^d + e_i^d$$

where $\mathbf{x}_i^d$ is the design vector of ν explanatory variables for the event time and e_i^d are the errors assumed to follow the normal distribution with mean 0 and standard deviation σ^d. This model, independently of the marker, can be estimated using Stata's `tobit` or `streg` command.

To allow for informative drop-out in the marker's measurements, we assume that subject-level random coefficients $\boldsymbol{\beta}_i$ (for example, subject-specific baseline value and rate-of-change deviations from the corresponding population average) and the survival model's residuals, e_i^d (deviations from mean log event time), jointly follow the multivariate normal distribution:

$$\boldsymbol{\beta}_i^{\text{joint}} = \begin{pmatrix} \boldsymbol{\beta}_i \\ e_i^d \end{pmatrix} \sim N\left\{ \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}, \Sigma^{\text{joint}} \right\} \text{where } \Sigma^{\text{joint}} = \begin{Bmatrix} \Sigma^m & \\ \boldsymbol{\sigma}^{md^T} & (\sigma^d)^2 \end{Bmatrix}$$

$\boldsymbol{\sigma}^{md}$ is a $\pi \times 1$ vector of the covariances of the marker's random coefficients ($\boldsymbol{\beta}_i$) with the survival model's residuals. If $\boldsymbol{\sigma}^{md} = \mathbf{0}$, then the marker's missing measurements due to the occurrence of the terminating event are at random; otherwise, the marker's missing data are informative.

2.1 Fitting procedure

The model can be written as

$$\mathbf{Y}_i^{\text{joint}} = X_i^{\text{joint}} \mathbf{b}^{\text{joint}} + Z_i^{\text{joint}} \boldsymbol{\beta}_i^{\text{joint}} + \mathbf{e}_i^{\text{joint}}$$

where

$$\mathbf{Y}_i^{\text{joint}} = \left\{ \mathbf{Y}_i^T, \ln(T_i^s) \right\}^T$$

$$\mathbf{b}^{\text{joint}} = \left(\mathbf{b}^T, \mathbf{b}^{d^T} \right)^T, \boldsymbol{\beta}_i^{\text{joint}} = \left(\boldsymbol{\beta}_i{}^T, e_i^d \right)^T, \mathbf{e}_i^{\text{joint}} = \left(\mathbf{e}_i^T, 0 \right)^T$$

$$X_i^{\text{joint}} = \begin{pmatrix} X_i & 0 \\ \mathbf{0}^T & \mathbf{X}_i^d \end{pmatrix} \text{ and } Z_i^{\text{joint}} = \begin{pmatrix} Z_i & 0 \\ \mathbf{0}^T & 1 \end{pmatrix}$$

Note that the last element of $\boldsymbol{\beta}_i^{\text{joint}}$ includes the error term of the survival model (e_i^d), and thus the corresponding one in $\mathbf{e}_i^{\text{joint}}$ is set to zero.

For uncensored event-times data (that is, $T_i = T_i^s$ for all subjects), all model parameters could be estimated using standard software for mixed models, provided that it has the required flexibility in the definition of the distribution of the random effects and level-1 errors. For example, MLn or MLwiN (for more information, see http://www.cmm.bris.ac.uk/MLwiN/index.shtml) implementing the iterative generalized least-squares (IGLS) and restricted iterative generalized least-squares (RIGLS) algorithms could be used. For a detailed description of IGLS and RIGLS, see Goldstein (2003).

With censored survival data, however, the standard RIGLS method has to be modified to obtain unbiased estimates of the model parameters. To deal with censored survival data, a version of the expectation-maximization (EM) algorithm was applied, considering censored survival observations as missing data. At each iteration, the survival part of the response variable and the survival component of the residuals cross-product

are replaced for the censored observations by their conditional expectations given the observed data and the current parameter estimates (E step), and then new parameters are obtained via RIGLS (M step). The estimation of these conditional expectations is based on multivariate normal theory (Johnson and Wichern 2007) and properties of the truncated normal distribution (Johnson et al. 1970). Initial values of the model parameters were obtained by treating censored survival times ($T_i = C_i$) as known survival times and applying the RIGLS method.

The null hypothesis $\boldsymbol{\sigma}^{md} = \mathbf{0}$ (that is, noninformative drop-out) can be tested via the likelihood-ratio test, estimating the likelihood of the joint model and that of the model in which $\boldsymbol{\sigma}^{md}$ (or some of its elements) are constrained to $\mathbf{0}$.

Standard errors of the fixed-effects parameters (as obtained from RIGLS in the M step of the algorithm) can be underestimated because of the replacement of censored survival times by their conditional expectations during the fitting procedure, ignoring the uncertainty of these expectations. A modified multiple-imputation (MI) method can be used to adjust the standard errors of the estimated fixed-effects parameters of the markers' trajectories for missing survival data (Rubin 1978; Touloumi et al. 2003).

The method consists of creating N pseudo-complete datasets where censored survival times have been replaced by random draws from the truncated normal distribution with mean and variance conditional on the marker's data and based on parameters' estimates at convergence. Fitting the model to each pseudo-complete dataset, one can estimate the empirical between-datasets variance for each fixed parameter. This between-datasets variance, weighted by $(N+1)/N$ to adjust for a finite number of samples, is then added to the within variance estimated from the previously fit joint model at its convergence to obtain the adjusted variance–covariance matrix of the fixed-effects parameters. It has been shown (Touloumi et al. 2003) that estimates obtained by this version of the MI method are similar to the ones obtained by the Louis (1982) method, with the MI method being less computationally intensive.

Most steps of the estimation procedure, including the whole RIGLS algorithm, have been implemented using Stata's matrix language, Mata.

3 The jmre1 command

3.1 Syntax

The `jmre1` command for Stata fits the JMRE model, as described in section 2.1. The syntax of `jmre1` is the following:

`jmre1` *depvar indepvars* [*if*] , `remark(`*varlist*`)` `dropout(`*varlist*`)` `id(`*varlist*`)`
`timevar(`*varname*`)` [`redrop(`*varname*`)` `l1(`*varname*`)` `maxi(#)` `tol(#)`
`burnin(#)` `trace(#)` `restr`
`corr0(`*varname1 varname2* [*varname3 varname4* ...]`)` `level(#)` `mi(#)`]

depvar is the joint response variable $\mathbf{Y}_i^{\text{joint}}$, comprising a column of the marker's measurements and the logarithm of the observed or censored event time for each level-2 unit.

indepvars are covariates for the evolution of the marker (X_i). `jmre1` does not automatically introduce a constant term in the regressors (intercept); thus a constant term should be explicitly included among *indepvars*.

We highly recommend that you use the accompanying `datajoint1` command (see section 3.5) to generate the joint response variable and to appropriately manipulate remaining data. In this case, the name of the response variable will be `_JY`, and all covariates' names start with `_M` or `_D` for the marker or drop-out model, respectively, with just `_M` and `_D` being the corresponding intercepts.

Finally, because of extensive manipulation of the data during the fitting procedure, `jmre1` uses a `preserve` and a `restore` command; thus `jmre1` cannot be used if data are already `preserve`d.

3.2 Options

`remark(`*varlist*`)` defines the random effects for the marker's model. At least one variable is required, which in most cases will be the intercept term (`_M` if the data are generated by `datajoint1`; see section 3.5). This is a required option.

`dropout(`*varlist*`)` defines the structure of the lognormal drop-out model. The first variable should be the event indicator variable (1 means observed, 0 means censored in the usual survival analysis sense). Remaining variables define the independent covariates of the survival model. `jmre1` does not add an intercept automatically as most Stata commands do, so the user should (almost always) explicitly enter the drop-out model's constant variable (`_D` if the data are generated by `datajoint1`; see section 3.5). This is a required option.

`id(`*varlist*`)` defines ID variables for identifying subjects (level-2 units). This is a required option.

`timevar(`*varname*`)` indicates the constant term in the drop-out model. If left blank, the default is `timevar(_D)`, which is the corresponding variable created by `datajoint1`. This is a required option.

`redrop(`*varname*`)` indicates the drop-out model's constant variable (`_D` if the data are generated by `datajoint1`; see section 3.5). The default is `redrop(_D)`.

`l1(`*varname*`)` indicates the constant term in the marker model. The default is `l1(_M)`, which is the corresponding variable created by `datajoint1` (see section 3.5).

`maxi(#)` denotes the maximum number of iterations for the EM algorithm. The default is `maxi(40)`.

`tol(#)` denotes the tolerance criterion for declaring convergence. The default is `tol(0.01)`, which means that the model has converged when relative differences in all model parameters between successive iterations are less than 0.01 (that is, 1%).

`burnin(#)` specifies the number of "burn-in" iterations. The default is `burnin(2)`, which means that the command will perform two simple RIGLS or IGLS iterations before invoking the nested EM algorithm.

`trace(#)` specifies to display all model parameters (fixed-effects parameters and random parameters) after each iteration. Random parameters are shown as a column vector created by the elements of the estimated variance–covariance matrix of the random effects (in lower triangular representation) and the estimated level-1 variance in the marker model.

`restr` specifies to use the RIGLS method instead of the default IGLS method. For normally distributed data, RIGLS is equivalent to restricted maximum likelihood and IGLS to maximum likelihood estimation.

`corr0(`*varname1 varname2* [*varname3 varname4* `...`]`)` indicates pairs of variables that denote that the corresponding elements in the variance–covariance matrix are constrained to zero. The variance–covariance matrix includes the random effects of the marker model and the residuals of the lognormal drop-out model.

`level(#)` denotes the level of significance for the confidence intervals.

`mi(#)` requests corrected standard errors of fixed effects by using the modified MI method. The suggested number of MIs is five (that is, `mi(5)`). This correction is required to account for the uncertainty when replacing censored survival times with their expectations. Given that this procedure can be time consuming, it can be invoked only once a final model has been chosen and not during all stages of a model selection procedure. If not specified or if specified as `mi(0)`, standard errors remain uncorrected.

3.3 Saved results

`jmre1` saves the following in `e()`:

Scalars

`e(N)`	number of records	`e(ll)`	(restricted) log likelihood
`e(var_eij)`	level-1 variance in marker model	`e(Nid)`	number of subjects

Macros

`e(cmd)`	`jmre1`	`e(id)`	ID variable(s)
`e(method)`	`RIGLS` or `IGLS`	`e(depvar)`	name of joint dependent variable
`e(re)`	varlist with drop-out model's constant and marker's random effects	`e(properties)`	`b V`
`e(predict)`	program used to implement `predict`		

Matrices

`e(b)`	coefficient vector (fixed effects)	`e(V)`	variance–covariance matrix of fixed effects
`e(cov_re)`	variance–covariance matrix of random effects	`e(corr_re)`	correlation matrix of random effects

Functions

`e(sample)`	marks estimation sample

3.4 Syntax for predict

As with all Stata estimation commands, `jmre1` supports the postestimation command `predict` (see [R] **predict**) to compute fitted values, empirical Bayes estimates of the marker's random effects (best linear unbiased predictions [BLUPs]), and residuals. The syntax for `predict` following `jmre1` is

`predict` [*type*] *newvarname* [*if*] [*in*] [`,` *statistic*
`equation(Drop_Out|Marker)`]

statistic	Description
`xb`	linear predictor for the fixed portion of the model; the default
`stdp`	standard error of the fixed-portion linear prediction xb
`reffects`	predictions of the marker's random effects (BLUPs)
`fitted`	linear predictor of the fixed portion plus contributions based on predicted random effects (fitted values)
`residuals`	residuals, response minus fitted values

(*Continued on next page*)

`equation(Drop_Out|Marker)` specifies the desired part of the model (only for the `xb` or `stdp` options). The default is `equation(Marker)`.

All predicted statistics, except for `xb` and `stdp`, are calculated for the marker part of the model and the corresponding observations. For `xb` and `stdp`, one can use the `equation()` option to specify the desired part of the model (drop-out or marker).

3.5 The datajoint1 utility command

The `datajoint1` command makes the appropriate manipulation of the data, which is required before fitting a `jmre` model.

`datajoint1 using` *marker_file*`, dfile(`*drop-out_file*`) markervalue(`*varname*`)`
`markertime(`*varname*`) dropevent(`*varname*`) droptime(`*varname*`) id(`*varlist*`)`
`[covariates(`*varlist*`) clear saving(`*filename*`) replace]`

The `datajoint1` command takes the name of a Stata data file (*marker_file*), which contains longitudinal measurements of a continuous variable (marker) along with the time the measurement was taken (`markertime()`). The file should also include an identifier variable for each subject. Each subject should typically have more than one marker's measurements, although subjects with only one measurement are allowed.

A second file (*drop-out_file*), provided in `dfile()`, contains the same identifier variable and has one record per subject. Required variables are a binary variable indicating whether the informative censoring event (`dropevent()`) has occurred and the time (`droptime()`) of the occurrence of the event (if the event has occurred) or the censoring time (here the word *censoring* has the usual survival analysis meaning). This file can also contain other baseline, non–time-dependent variables. The command checks the data for some basic requirements, but it is the user's responsibility to correctly prepare *marker_file* and *drop-out_file*.

Options

`dfile(`*drop-out_file*`)` specifies the name of the file that contains data on the informative drop-out event along with other time-constant covariates. This is a required option.

`markervalue(`*varname*`)` specifies the name of the variable that contains the longitudinal values of the marker. This is a required option.

`markertime(`*varname*`)` specifies the name of the variable that contains the time each marker's measurement was taken. This is a required option.

`dropevent(`*varname*`)` specifies the indicator variable for the occurrence of the informative drop-out event. This is a required option.

`droptime(`*varname*`)` specifies the event or censoring time. This is a required option.

`id(`*varlist*`)` specifies the subject's identifier variables. This is a required option.

`covariates(`*varlist*`)` specifies other baseline, non–time-dependent variables that will probably be used to model the marker's trend or the informative drop-out mechanism. These variables should only be included in the *drop-out_file*.

`clear` specifies to clear the data in memory, even though they have not yet been saved to disk.

`saving(`*filename*`)` specifies the name of the file to be created.

`replace` specifies that the file specified in the `saving()` option may be replaced if it already exists.

4 Example

The data for this example are derived from an HIV/AIDS cohort study (CASCADE Collaboration 2003). This study had two main objectives: 1) to estimate the rate of decline of the immunological marker called CD4 cell count after infection with the HIV-1 virus and 2) to evaluate the effect of potential prognostic factors (such as age, mode of infection-risk group, sex, etc.) on the initial levels of this marker and its subsequent rate of decline. The period of interest starts at the date of infection or, more precisely, the date of seroconversion and stops at the date a patient started antiretroviral treatment (ART), developed clinical AIDS, or died. Censoring the series of longitudinal CD4 cell-count measurements at the date of ART initiation should probably be considered a MAR mechanism because physicians' decisions regarding treatment are usually based on observed values of CD4 cell count. On the other hand, premature termination of measurements due to AIDS onset or death could be informative because the probability of AIDS onset or death is likely related to an individual's underlying trend of CD4 cell-count evolution. Hence, in the following analysis, death or AIDS onset will be treated as an informative (nonignorable) event, whereas all other causes of measurements' termination (end of study, loss to follow-up, censoring due to ART initiation) will be considered noninformative (ignorable) events.

The original dataset consisted of data on 5,739 individuals, but here we are using a random sample of 400 individuals. We have prepared two datasets: one containing multiple records per individual with longitudinal CD4 cell-count measurements and the other containing one record per individual with just two covariates (age at seroconversion and risk group) along with information about the drop-out mechanism. In the following output, we describe the variables of these two files and list their contents for the first three individuals.

(*Continued on next page*)

```
. use cd4
(CD4 sample file)
. describe
Contains data from cd4.dta
  obs:         4,677                          CD4 sample file
 vars:             5                          13 Mar 2009 12:31
 size:       140,310 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
study_id        byte   %8.0g                  Study ID
patient_id      str11  %11s                   Patient ID
cd4             int    %9.0g                  CD4 count
sqrt_cd4        float  %9.0g                  CD4 count(sq.root)
time            float  %9.0g                  CD4 time (yrs.)
-------------------------------------------------------------------------------
Sorted by:  study_id  patient_id  time
. list in 1/32, abbreviate(10) noobs sepby(study_id patient_id)
```

study_id	patient_id	cd4	sqrt_cd4	time
1	1531	383	19.57038	4.284737
1	1531	455	21.33073	4.476386
1	1531	423	20.56696	4.744695
1	1531	449	21.18962	4.988364
1	1531	548	23.4094	5.054072
1	1531	548	23.4094	5.108829
1	1531	399	19.97499	5.256673
1	1531	576	24	5.631759
1	1531	429	20.71231	6.105407
1	1531	435	20.85665	6.324435
1	1531	437	20.90454	6.387406
1	1534	1096	33.10589	5.24846
1	1534	614	24.77902	6.146475
1	1534	586	24.20744	7.022587
1	1534	479	21.88607	7.457906
1	1534	563	23.72762	7.939767
1	1534	510	22.58318	8.687201
1	1534	631	25.11971	8.91718
1	1534	449	21.18962	9.138946
1	1534	464	21.54066	9.399042
1	1534	374	19.33908	9.744011
1	1552	445	21.09502	.276523
1	1552	540	23.2379	.506502
1	1552	524	22.89105	.793977
1	1552	726	26.94439	1.062286
1	1552	608	24.65766	1.774127
1	1552	1098	33.13608	2.020534
1	1552	823	28.68798	2.269678
1	1552	969	31.12877	2.557153
1	1552	827	28.75761	2.8282
1	1552	1352	36.76955	3.271732
1	1552	683	26.13427	3.63039

Variables `study_id` and `patient_id` are study identifiers and patient identifiers, respectively (data are derived from a multicenter study), and are both required to uniquely identify a patient. `cd4` is the CD4 cell-count measurement in its original scale (cells/μL), and `sqrt_cd4` is the corresponding square-root–transformed values. Finally, `time` is the time in years of the CD4 cell-count measurement since seroconversion.

```
. use surv
(AIDS/Death + covariates sample file)
. describe
Contains data from surv.dta
  obs:           400                          AIDS/Death + covariates sample file
 vars:             6                          13 Mar 2009 12:31
 size:        10,800 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
study_id        byte   %8.0g                  Study ID
patient_id      str11  %11s                   Patient ID
AD              byte   %9.0g       ynlbl      AIDS or Death
ADtime          float  %9.0g                  AIDS/Death or censoring time
agesero         byte   %9.0g       agesero    Age at SC
expo            byte   %17.0g      expo       Risk group
-------------------------------------------------------------------------------
Sorted by:  study_id  patient_id
. list in 1/3, abbreviate(15) noobs sep(0)
```

study_id	patient_id	AD	ADtime	agesero	expo
1	1531	No	6.387408	20-29	Heteros.(male)
1	1534	No	9.744013	20-29	Homos.
1	1552	No	4.246408	20-29	Homos.

```
. label list agesero
agesero:
           1 15-19
           2 20-29
           3 30-39
           4 40+
. label list expo
expo:
           1 Homos.
           2 Heteros.(male)
           3 Heteros.(fem.)
           4 Drug users (male)
           5 Drug users (fem.)
           6 Haemoph.
```

`AD` is a binary variable having the value of 1 if an individual developed AIDS or died and 0 otherwise. `ADtime` contains the time of AIDS development or death when `AD` equals 1 or contains the time of end of follow-up or censoring due to ART initiation if `AD` equals 0. `agesero` and `expo` are categorical variables containing information about age at seroconversion and risk group, respectively.

4.1 Preparing the "joint" dataset

We will now use the `datajoint1` command to prepare our "joint" dataset for the final analysis

```
. datajoint1 using cd4, dfile(surv) markervalue(sqrt_cd4) markertime(time)
> id(study_id patient_id)  dropevent(AD) droptime(ADtime)
> covariates(agesero expo) saving(jointdata) clear replace
file jointdata.dta saved
```

The command creates two indicator variables named `_M` and `_D` to separate records that correspond to marker models and to drop-out models, respectively. For each covariate specified in the `covariates()` option, two new variables with the same names but prefixed with `_M` and `_D` have been created. The new variables whose names start with `_M` have the same values as the original ones but only for records corresponding to the marker model (that is, those where `_M` equals one). New variables whose names start with `_D` have been created in a similar way for records corresponding to the drop-out model. Variable `_JY` is the joint response variable. Within each patient's block of records, `_JY` has the corresponding value of the dependent variable of the marker model (that is, the square root of CD4 cell count, `sqrt_cd4`) for the first `_N-1` records and has the logarithm of the event or censoring time for the `_N`th patient's record. Finally, a new variable (named `_Mtime` in this case) has been created containing the time of marker's measurements only for records corresponding to the marker model. The following list shows the actual data for the first three patients (some variables have been omitted):

```
. list patient_id _M _D _JY _Mtime _Magesero _Dagesero in 1/35, abbreviate(10)
> noobs sepby(study_id patient_id)
```

patient_id	_M	_D	_JY	_Mtime	_Magesero	_Dagesero
1531	1	0	19.57038	4.284737	20-29	.
1531	1	0	21.33073	4.476386	20-29	.
1531	1	0	20.56696	4.744695	20-29	.
1531	1	0	21.18962	4.988364	20-29	.
1531	1	0	23.4094	5.054072	20-29	.
1531	1	0	23.4094	5.108829	20-29	.
1531	1	0	19.97499	5.256673	20-29	.
1531	1	0	24	5.631759	20-29	.
1531	1	0	20.71231	6.105407	20-29	.
1531	1	0	20.85665	6.324435	20-29	.
1531	1	0	20.90454	6.387406	20-29	.
1531	0	1	1.854329	.	.	20-29
1534	1	0	33.10589	5.24846	20-29	.
1534	1	0	24.77902	6.146475	20-29	.
1534	1	0	24.20744	7.022587	20-29	.
1534	1	0	21.88607	7.457906	20-29	.
1534	1	0	23.72762	7.939767	20-29	.
1534	1	0	22.58318	8.687201	20-29	.
1534	1	0	25.11971	8.91718	20-29	.
1534	1	0	21.18962	9.138946	20-29	.
1534	1	0	21.54066	9.399042	20-29	.
1534	1	0	19.33908	9.744011	20-29	.
1534	0	1	2.276653	.	.	20-29
1552	1	0	21.09502	.276523	20-29	.
1552	1	0	23.2379	.506502	20-29	.
1552	1	0	22.89105	.793977	20-29	.
1552	1	0	26.94439	1.062286	20-29	.
1552	1	0	24.65766	1.774127	20-29	.
1552	1	0	33.13608	2.020534	20-29	.
1552	1	0	28.68798	2.269678	20-29	.
1552	1	0	31.12877	2.557153	20-29	.
1552	1	0	28.75761	2.8282	20-29	.
1552	1	0	36.76955	3.271732	20-29	.
1552	1	0	26.13427	3.63039	20-29	.
1552	0	1	1.446073	.	.	20-29

Note that for patient 1,531, for example, the last value of `_JY` is the logarithm of his event or censoring time [that is, $\log(6.387408) = 1.854329$]. It is highly recommended that the user not alter the file created at all. In case new variables (or recoding of existing variables) are required, these should be created on the original `surv` file followed by a new run of the `datajoint1` command.

4.2 Fitting the JMRE model using the jmre1 command

We will now fit a JMRE model to the data described above. The marker model will include a random intercept and a random slope (that is, time coefficient). Fixed effects will include intercept and slope and their interactions with age at seroconversion and

risk group. Consequently, the model will allow for the estimation of the average initial marker levels and their subsequent rate of change for the reference category (fixed intercept and slope, respectively), and the average deviations of these quantities according to age at seroconversion categories and risk group. The inclusion of a random intercept and a random slope implies that each individual is allowed to have initial marker values and a subsequent rate of change that deviate from the corresponding average values for the covariate pattern the individual belongs to. The random-effects variance–covariance matrix Σ^m is unstructured (there are no constraints defined via the `corr0()` option), allowing the variances of the random intercept and slope, along with their correlation, to be freely estimated.

The drop-out model will have the same prognostic factors as the marker model, allowing the average (log-transformed) time to death or AIDS onset to be different according to age at seroconversion and risk group. In general, we recommend that the user fit various simple lognormal survival models, using the `streg` command, to select a set of significant predictors. Then the user should use these predictors when fitting the JMRE model.

```
. xi: jmre1 _JY _M i._Magesero*_Mtime i._Mexpo*_Mtime, remark(_M _Mtime)
> dropout(AD _D i._Dagesero i._Dexpo) timevar(_Mtime) id(study_id patient_id)
> restr
```

(*output omitted*)

The part following the `jmre1` command follows the usual Stata conventions with two exceptions: 1) the intercept term (`_M`) is explicitly entered and 2) the *depvars* part corresponds only to the marker model. The command uses the old `xi:` syntax and does not yet support factor notation. The drop-out model is practically declared within the `dropout()` option, where the first variable is the binary variable for the drop-out event (`AD`) and the remaining variables denote the corresponding covariates. As in the marker model, the intercept term (`_D`) is also entered explicitly. Options `redrop()` and `l1()` have been omitted; thus they have their default values (`_D` and `_M`, respectively). The required `timevar()` option declares the variable containing the time of the marker's measurements. Finally, we added the `restr` option to use the RIGLS estimation algorithm.

The results of the previous command are shown in the following output:

```
Response variable: _JY, 400 subjects, 4677 marker measurements
Restricted Log-likelihood = -13074.39
Fixed effects
```

_JY	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
Drop_Out						
_D	2.439713	.107579	22.68	0.000	2.228863	2.650564
_I_Dageser~2	-.0308937	.1114061	-0.28	0.782	-.2492456	.1874583
_I_Dageser~3	-.2716915	.1206316	-2.25	0.024	-.508125	-.035258
_I_Dageser~4	-.4275789	.1248543	-3.42	0.001	-.6722889	-.182869
_I_Dexpo_2	.4460437	.1672487	2.67	0.008	.1182423	.773845
_I_Dexpo_3	.1914361	.1316591	1.45	0.146	-.066611	.4494832
_I_Dexpo_4	-.0339984	.1170231	-0.29	0.771	-.2633595	.1953627
_I_Dexpo_5	.0656182	.1397301	0.47	0.639	-.2082478	.3394843
_I_Dexpo_6	.1245609	.1817095	0.69	0.493	-.2315832	.4807049
Marker						
_M	24.84069	.9066342	27.40	0.000	23.06372	26.61766
_I_Mageser~2	.296751	.9524523	0.31	0.755	-1.570021	2.163523
_I_Mageser~3	-.4586846	1.018263	-0.45	0.652	-2.454444	1.537075
_I_Mageser~4	-.9201732	1.035369	-0.89	0.374	-2.949459	1.109112
_Mtime	-1.117109	.2200785	-5.08	0.000	-1.548455	-.685763
_I_MaX_Mti~2	-.3308813	.2240275	-1.48	0.140	-.7699671	.1082046
_I_MaX_Mti~3	-.4605507	.2498886	-1.84	0.065	-.9503234	.0292219
_I_MaX_Mti~4	-.5433615	.2559171	-2.12	0.034	-1.04495	-.0417731
_I_Mexpo_2	-2.135086	1.326851	-1.61	0.108	-4.735666	.4654932
_I_Mexpo_3	.2751593	1.054363	0.26	0.794	-1.791355	2.341674
_I_Mexpo_4	.5212439	1.005444	0.52	0.604	-1.449389	2.491877
_I_Mexpo_5	3.521577	1.213382	2.90	0.004	1.143392	5.899763
_I_Mexpo_6	5.28702	1.568754	3.37	0.001	2.212318	8.361721
_I_MeX_Mti~2	.8563533	.3350725	2.56	0.011	.1996232	1.513083
_I_MeX_Mti~3	.0470733	.2670043	0.18	0.860	-.4762455	.5703921
_I_MeX_Mti~4	-.3032204	.2449095	-1.24	0.216	-.7832342	.1767934
_I_MeX_Mti~5	-.411662	.290144	-1.42	0.156	-.9803337	.1570098
_I_MeX_Mti~6	-.6364512	.348253	-1.83	0.068	-1.319015	.0461122

```
Random effects (Covariance matrix)

symmetric e(cov_re)[3,3]
              _D         _M     _Mtime
    _D  .52415435
    _M  .06505558  31.417146
_Mtime   .6146643  -3.164154  1.6422581

Level-1 variance (Marker):   9.31419
```

The output first lists the name of the joint response variable, the number of individuals, the number of marker measurements in the dataset, and the restricted (because we used the `restr` option) log likelihood. Then the usual Stata estimation-command table follows, which includes results for the drop-out model and the marker model fixed effects. Finally, the estimates for the random parameters are given: first, the variance–

covariance matrix for the random effects of the marker model's and the drop-out model's residuals, and then the estimate of the variance of the level-1 marker's residuals. One could list the correlation matrix of the random effects instead of the variance–covariance matrix, as shown below:

```
. matrix list e(corr_re)
symmetric e(corr_re)[3,3]
                 _D          _M      _Mtime
    _D            1
    _M    .01603141           1
_Mtime    .66250274  -.44050773           1
```

The drop-out model's residuals have a relatively high positive correlation with the marker model's random slopes [Corr(_D, _Mtime) = 0.663]. This means that individuals with higher (less steep) slopes tend to develop AIDS or die later. The correlation with the initial values of the marker is still positive but much lower. To evaluate the significance of these findings, we could fit a JMRE model with these two correlations constrained to zero and compare with the previous unconstrained model via the likelihood-ratio test. The test will be valid because the two models share the same fixed-effects structure (thus the fitting through the restricted algorithm does not cause any problems), and because the parameters being tested are correlations, thus the null hypothesis does not lie on the boundary of the parameter space. Before proceeding with the fit of the constrained model, we are storing the estimation results of the current model:

```
. estimates store FullModel
```

Now we can fit the constrained model. The only change compared with the previous `jmre1` command is the addition of the `corr0(_D _M _D _Mtime)` option. This option specifies that the correlation of the drop-out residuals with the marker's random intercept (_D and _M) as well as with the marker's random slope (_D and _Mtime) should be constrained to zero. This is equivalent to fitting the marker and drop-out models separately (that is, ignoring the informative drop-outs). The age effect on the marker's slope (coefficients _I_MaX_Mti~2, _I_MaX_Mti~3, _I_MaX_Mti~4) is now attenuated and not significant (compare with the output of the unconstrained model)

```
. xi: jmre1 _JY _M i._Magesero*_Mtime i._Mexpo*_Mtime, remark(_M _Mtime)
> dropout(AD _D i._Dagesero i._Dexpo) timevar(_Mtime) id(study_id patient_id)
> restr corr0(_D _M _D _Mtime)
  (output omitted)
Response variable: _JY, 400 subjects, 4677 marker measurements
Restricted Log-likelihood = -13199.23
Fixed effects
```

_JY	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
Drop_Out						
_D	2.461083	.1168558	21.06	0.000	2.23205	2.690116
_I_Dageser~2	-.0261404	.1211109	-0.22	0.829	-.2635135	.2112327
_I_Dageser~3	-.2622229	.1309363	-2.00	0.045	-.5188534	-.0055925
_I_Dageser~4	-.4619534	.1351883	-3.42	0.001	-.7269177	-.1969892
_I_Dexpo_2	.4568295	.1851469	2.47	0.014	.0939482	.8197108
_I_Dexpo_3	.1958888	.1443903	1.36	0.175	-.0871109	.4788886
_I_Dexpo_4	-.0221913	.1263402	-0.18	0.861	-.2698135	.2254309
_I_Dexpo_5	.0939474	.1517552	0.62	0.536	-.2034874	.3913823
_I_Dexpo_6	.133236	.1942086	0.69	0.493	-.2474059	.5138779
Marker						
_M	24.71954	.9150865	27.01	0.000	22.92601	26.51308
_I_Mageser~2	.1971617	.9625667	0.20	0.838	-1.689434	2.083758
_I_Mageser~3	-.5378366	1.028217	-0.52	0.601	-2.553104	1.477431
_I_Mageser~4	-1.021058	1.044502	-0.98	0.328	-3.068243	1.026128
_Mtime	-1.081002	.2237515	-4.83	0.000	-1.519547	-.6424574
_I_MaX_Mti~2	-.2355486	.2265238	-1.04	0.298	-.679527	.2084299
_I_MaX_Mti~3	-.318824	.2551211	-1.25	0.211	-.8188521	.1812041
_I_MaX_Mti~4	-.3664625	.2604899	-1.41	0.159	-.8770134	.1440883
_I_Mexpo_2	-2.087981	1.336893	-1.56	0.118	-4.708242	.5322803
_I_Mexpo_3	.3305508	1.063717	0.31	0.756	-1.754297	2.415398
_I_Mexpo_4	.4232636	1.015625	0.42	0.677	-1.567326	2.413853
_I_Mexpo_5	3.274163	1.229877	2.66	0.008	.8636475	5.684678
_I_Mexpo_6	5.309659	1.58036	3.36	0.001	2.212211	8.407107
_I_MeX_Mti~2	.845122	.3432575	2.46	0.014	.1723496	1.517894
_I_MeX_Mti~3	-.0199391	.272227	-0.07	0.942	-.5534942	.5136161
_I_MeX_Mti~4	-.1980574	.2522235	-0.79	0.432	-.6924065	.2962916
_I_MeX_Mti~5	-.3552085	.2958411	-1.20	0.230	-.9350465	.2246295
_I_MeX_Mti~6	-.6829444	.3431703	-1.99	0.047	-1.355546	-.0103431

```
Random effects (Covariance matrix)

symmetric e(cov_re)[3,3]
                _D          _M      _Mtime
    _D   .58751149
    _M           0   31.641699
_Mtime           0  -3.3338155   1.5358076

Level-1 variance (Marker):   9.28585
```

We can now save the new estimates and compare the last two models using the likelihood-ratio test, as follows:

```
. estimates store ConstrModel
. estimates restore FullModel
(results FullModel are active now)
. local FMll = e(ll)
. estimates restore ConstrModel
(results ConstrModel are active now)
. local CMll = e(ll)
. display -2*(`CMll´-`FMll´)
249.68
. display chi2tail(2, -2*(`CMll´-`FMll´))
6.063e-55
```

The unconstrained model has a significantly higher likelihood (likelihood-ratio chi-squared = 249.68, degrees of freedom = 2, p-value < 0.001), indicating that the correlations being tested do significantly differ from zero. The significance implies that premature termination of the marker's measurements due to AIDS or death is informative (informative drop-out mechanism).

Note that the `lrtest` command is not fully supported; it does not work when the models under comparison differ in random-effects parameters. When it is used for testing fixed-effects parameters, it is the user's responsibility to make sure both models have been fit without the `restr` option. (Likelihood-ratio tests between models that differ in their fixed-effects structure are not valid when the models have been fit with the RIGLS algorithm.)

We can also use the `mi()` option to adjust the standard errors of the fixed-effects estimates to account for the uncertainty when replacing censored survival times (due to study termination, loss to follow-up, or ART initiation) with their expectations. We will use five imputed datasets because this is the minimum recommended value.

```
. xi: jmre1 _JY _M i._Magesero*_Mtime i._Mexpo*_Mtime, remark(_M _Mtime)
> dropout(AD _D i._Dagesero i._Dexpo) timevar(_Mtime) id(study_id patient_id)
> restr mi(5)
```

(*output omitted*)

```
Fixed effects
```

_JY	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
Drop_Out						
_D	2.439713	.1248479	19.54	0.000	2.195016	2.684411
_I_Dageser~2	-.0308937	.1290179	-0.24	0.811	-.2837641	.2219767
_I_Dageser~3	-.2716915	.1437134	-1.89	0.059	-.5533647	.0099817
_I_Dageser~4	-.4275789	.1398424	-3.06	0.002	-.7016651	-.1534928
_I_Dexpo_2	.4460437	.2210933	2.02	0.044	.0127088	.8793785
_I_Dexpo_3	.1914361	.1438282	1.33	0.183	-.090462	.4733342
_I_Dexpo_4	-.0339984	.1405277	-0.24	0.809	-.3094277	.2414308
_I_Dexpo_5	.0656182	.1651509	0.40	0.691	-.2580715	.389308
_I_Dexpo_6	.1245609	.1947826	0.64	0.523	-.2572061	.5063278
Marker						
_M	24.84069	.9128988	27.21	0.000	23.05144	26.62994
_I_Mageser~2	.296751	.9595332	0.31	0.757	-1.5839	2.177401
_I_Mageser~3	-.4586846	1.025669	-0.45	0.655	-2.468958	1.551589
_I_Mageser~4	-.9201732	1.04611	-0.88	0.379	-2.970511	1.130164
_Mtime	-1.117109	.2291318	-4.88	0.000	-1.566199	-.6680188
_I_MaX_Mti~2	-.3308813	.2386036	-1.39	0.166	-.7985358	.1367733
_I_MaX_Mti~3	-.4605507	.2613587	-1.76	0.078	-.9728045	.051703
_I_MaX_Mti~4	-.5433615	.2703591	-2.01	0.044	-1.073256	-.0134674
_I_Mexpo_2	-2.135086	1.330645	-1.60	0.109	-4.743103	.47293
_I_Mexpo_3	.2751593	1.059339	0.26	0.795	-1.801108	2.351426
_I_Mexpo_4	.5212439	1.00704	0.52	0.605	-1.452517	2.495005
_I_Mexpo_5	3.521577	1.216155	2.90	0.004	1.137957	5.905197
_I_Mexpo_6	5.28702	1.570501	3.37	0.001	2.208894	8.365146
_I_MeX_Mti~2	.8563533	.3386069	2.53	0.011	.1926961	1.520011
_I_MeX_Mti~3	.0470733	.2795631	0.17	0.866	-.5008604	.595007
_I_MeX_Mti~4	-.3032204	.247707	-1.22	0.221	-.7887173	.1822765
_I_MeX_Mti~5	-.411662	.2959009	-1.39	0.164	-.9916171	.1682931
_I_MeX_Mti~6	-.6364512	.3512918	-1.81	0.070	-1.324971	.0520682

(*output omitted*)

The inflation of the standard errors is almost negligible in the marker model and relatively small in the drop-out model.

4.3 Postestimation

We will now use the `predict` postestimation command to informally (that is, graphically) examine the distributions of the marker's random effects (the marker's level-1 residuals) and graphically compare the previously fit constrained and unconstrained models. We start by obtaining empirical Bayes estimates of the marker's random intercept and random slope (also known as BLUPs):

```
. estimates restore FullModel
(results FullModel are active now)
. predict eb, ref
(5077 real changes made)
. describe eb*
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
eb_M            float  %9.0g                  BLUP of Constant(M)
eb_Mtime        float  %9.0g                  BLUP of CD4 time (yrs.)(M)
```

Note the names and the variable labels of the newly created variables. To examine their distributions, we will `preserve` the dataset and keep only the first record of each individual (see figure 1). (The BLUPs are constant within each patient's block of records and missing at the last record, which corresponds to the drop-out model.)

```
. preserve
. by study_id patient_id: keep if _n==1
(4677 observations deleted)
. histogram eb_M, scheme(s2mono)
(bin=20, start=-17.303648, width=1.6328218)
. restore
```

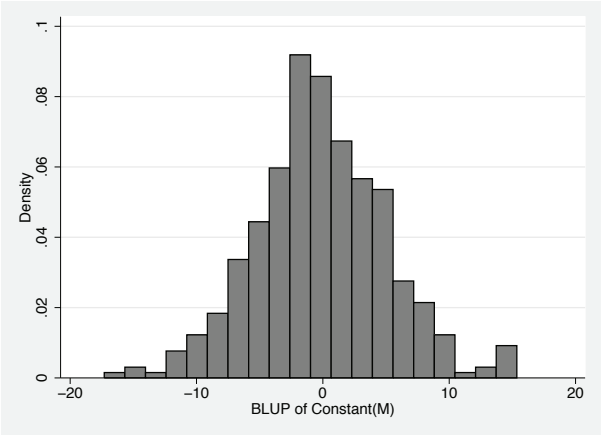

Figure 1. Histogram of BLUPs for the marker's random intercept

Similarly, we can have a histogram for the BLUPs of the marker's random slope:

```
. preserve
. by study_id patient_id: keep if _n==1
(4677 observations deleted)
. histogram eb_Mtime, scheme(s2mono)
(bin=20, start=-3.5219173, width=.33129596)
. restore
```

Or for the BLUPs of the marker's level-1 residuals:

```
. predict resid, residuals
(5077 real changes made)
. histogram resid, scheme(s2mono)
(bin=36, start=-16.156229, width=1.0067631)
```

Now we will use predicted values based on fixed-effects estimates from both the constrained and the unconstrained (full) models to graphically display (see figure 2) the predicted average evolution of CD4 cell count by age at seroconversion groups. Risk group will be kept constant to keep the graph simple. We are also using the `fillin` command to add some missing covariate patterns. Finally, note that we are back-transforming the predictions to the original scale:

```
. preserve
. drop _I*
. replace _Mtime = round(_Mtime,0.5)
(4658 real changes made)
. fillin _M _Mexpo _Magesero _Mtime
. xi  i._Magesero*_Mtime i._Mexpo*_Mtime
i._Magesero       _I_Magesero_1-4     (naturally coded; _I_Magesero_1 omitted)
i._Mag~o*_Mtime   _I_MaX_Mtim_#       (coded as above)
i._Mexpo          _I_Mexpo_1-6        (naturally coded; _I_Mexpo_1 omitted)
i._Mexpo*_Mtime   _I_MeX_Mtim_#       (coded as above)
. estimates restore FullModel
(results FullModel are active now)
. predict fe_fitted_full, xb
(7240 real changes made)
. replace fe_fitted_full = fe_fitted_full^2
(5956 real changes made)
. label var fe_fitted_full "Full model"
. estimates restore ConstrModel
(results ConstrModel are active now)
. predict fe_fitted_constr, xb
(7240 real changes made)
. replace fe_fitted_constr = fe_fitted_constr^2
(5956 real changes made)
. label var fe_fitted_constr "Constrained model"
. sort _Mexpo _Magesero _Mtime
. scatter fe_fitted_full _Mtime if _Mexpo==2 & _M, msymbol(i) connect(l)
> lpattern(solid) lcolor(gs0)|| scatter fe_fitted_constr _Mtime if _Mexpo==2
> & _M, msymbol(i) connect(l) lpattern(solid) lcolor(gs10) by(_Magesero)
> ytitle("Predicted CD4 cell count" "(cells/microL)")
> xtitle("Years since Seroconversion") ylabel(0(100)600,angle(hori))
> scheme(s2mono)
. restore
```

(*Continued on next page*)

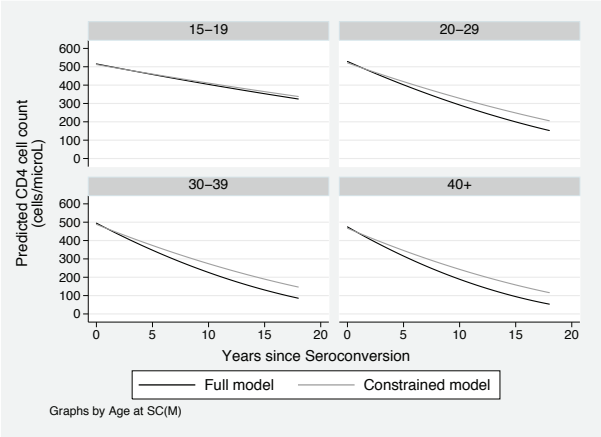

Figure 2. Average predicted CD4 cell-count evolution by age at seroconversion (risk group = heterosexuals/males); predictions based on models with ("full") or without ("constrained") adjustment for informative drop-outs

You can see that the constrained model, which does not make any adjustment for the informative drop-out, estimates less steep CD4 cell-count declines than does the unconstrained model. The differences between the two models are more pronounced at older ages, where informative drop-out is more severe (that is, more persons die or develop AIDS).

5 Conclusion

We presented the new Stata `jmre1` command, which implements the joint multivariate random-effects model (Touloumi et al. 1999). The model adjusts the estimates for the longitudinal evolution of a continuous marker when the series of its measurements are subject to informative drop-out. That is, the model adjusts the estimates when the series of the marker's measurements is prematurely terminated by an event whose probability is related to subject-specific parameters of the underlying marker's trend.

The estimation procedure is based on a combination of the EM algorithm with the RIGLS or IGLS method and is implemented mainly using Stata's matrix language, Mata. It has been shown that the model itself, if correctly specified, yields estimates of minimal bias with close to nominal coverage probabilities (Touloumi et al. 1999; Touloumi et al. 2003). It has also been shown that an expanded bivariate version of the same model (that is, two continuous markers modeled simultaneously in the presence of informative drop-out) is fairly robust to at least moderate deviations of its distributional assumptions (Pantazis and Touloumi 2007); therefore, we believe that the current model will be equally robust.

The Stata implementation we presented in this article is much more user friendly compared with the initial implementation (see the appendix in Touloumi et al. [2002]) in the MLn statistical software package. We have also provided a utility command (`datajoint1`) that facilitates the required data preparation.

During the development of these commands, we compared `jmre1` with the prior implementation of the same model in MLn; differences in the estimates were in the order of the convergence tolerance. We also performed simulations where we observed that if the drop-out is not informative, the results for the marker model from `jmre1` are coinciding with those obtained by `xtmixed`. Similarly, when we constrained the correlations between the random effects of the marker model and the residuals of the drop-out model to zero in the `jmre1` command (that is, forcing the model to ignore the informative drop-out effects), the obtained results for the marker model were the same as those produced by a corresponding random-effects model fit through the `xtmixed` command (results not shown).

However, our implementation of the RIGLS algorithm that was required in the estimation procedure is not fully optimized; thus convergence with big datasets can be slow. For example, the fit of the full model in the previously given example dataset (400 individuals with 4,677 marker measurements in total) required approximately 45 seconds using an Intel Core 2 Quad Q9450 PC clocked at 2.66 GHz. We plan to fully optimize the code in an updated version using all the computational details given in Goldstein and Rasbash (1992). We also plan to expand our implementation to the bivariate version of the JMRE model (Pantazis and Touloumi 2005) to allow simultaneous modeling of two continuous markers in the presence of informative drop-out.

6 Acknowledgments

We would like to thank the steering committee and all participants of the CASCADE collaboration for allowing us to use part of the CASCADE data for the illustration of `jmre1`'s use.

7 References

CASCADE Collaboration. 2003. Differences in CD4 cell counts at seroconversion and decline among 5739 HIV-1-infected individuals with well-estimated dates of seroconversion. *Journal of Acquired Immune Deficiency Syndromes* 34: 76–83.

Diggle, P., and M. G. Kenward. 1994. Informative drop-out in longitudinal data analysis. *Journal of the Royal Statistical Society, Series C* 43: 49–93.

Faucett, C. L., and D. C. Thomas. 1996. Simultaneously modelling censored survival data and repeatedly measured covariates: A Gibbs sampling approach. *Statistics in Medicine* 15: 1663–1685.

Goldstein, H. 2003. *Multilevel Statistical Models*. 3rd ed. London: Arnold.

Goldstein, H., and J. Rasbash. 1992. Efficient computational procedures for the estimation of parameters in multilevel models based on iterative generalised least squares. *Computational Statistics & Data Analysis* 13: 63–71.

Johnson, N. L., S. Kotz, and N. Balakrishnan. 1970. *Continuous Univariate Distributions*, vol. 1. New York: Wiley.

Johnson, R. A., and D. W. Wichern. 2007. *Applied Multivariate Statistical Analysis*. 6th ed. Upper Saddle River, NJ: Prentice Hall.

Laird, N. M. 1988. Missing data in longitudinal studies. *Statistics in Medicine* 7: 305–315.

Laird, N. M., and J. H. Ware. 1982. Random-effects models for longitudinal data. *Biometrics* 38: 963–974.

Little, R. J. A. 1995. Modeling the drop-out mechanism in repeated-measures studies. *Journal of the American Statistical Association* 90: 1112–1121.

Little, R. J. A., and D. B. Rubin. 2002. *Statistical Analysis with Missing Data*. 2nd ed. Hoboken, NJ: Wiley.

Louis, T. A. 1982. Finding the observed information matrix when using the EM algorithm. *Journal of the Royal Statistical Society, Series B* 44: 226–233.

Pantazis, N., and G. Touloumi. 2005. Bivariate modelling of longitudinal measurements of two human immunodeficiency type 1 disease progression markers in the presence of informative drop-outs. *Journal of the Royal Statistical Society, Series C* 54: 405–423.

———. 2007. Robustness of a parametric model for informatively censored bivariate longitudinal data under misspecification of its distributional assumptions: A simulation study. *Statistics in Medicine* 26: 5473–5485.

Rabe-Hesketh, S., A. Skrondal, and A. Pickles. 2002. Multilevel selection models using gllamm. Combined Dutch and German Stata Users Group meeting proceedings. http://www.stata.com/meeting/2dutch/select.pdf.

Rubin, D. B. 1978. Multiple imputations in sample surveys: A phenomenological Bayesian approach to nonresponse. In *Proceedings of the Survey Research Methods Section of the American Statistical Association*, 20–34.

Schluchter, M. D. 1992. Methods for the analysis of informatively censored longitudinal data. *Statistics in Medicine* 11: 1861–1870.

Touloumi, G., A. G. Babiker, M. G. Kenward, S. J. Pocock, and J. H. Darbyshire. 2003. A comparison of two methods for the estimation of precision with incomplete longitudinal data, jointly modelled with a time-to-event outcome. *Statistics in Medicine* 22: 3161–3175.

Touloumi, G., S. J. Pocock, A. G. Babiker, and J. H. Darbyshire. 1999. Estimation and comparison of rates of change in longitudinal studies with informative drop-outs. *Statistics in Medicine* 18: 1215–1233.

———. 2002. Impact of missing data due to selective dropouts in cohort studies and clinical trials. *Epidemiology* 13: 347–355.

Wu, M. C., and K. R. Bailey. 1989. Estimation and comparison of changes in the presence of informative right censoring: Conditional linear model. *Biometrics* 45: 939–955.

Wu, M. C., and R. J. Carroll. 1988. Estimation and comparison of changes in the presence of informative right censoring by modeling the censoring process. *Biometrics* 44: 175–188.

About the authors

Nikos Pantazis has a PhD in biostatistics and works as a research collaborator in the Department of Hygiene, Epidemiology, and Medical Statistics at the University of Athens Medical School, Greece.

Giota Touloumi is an associate professor of biostatistics in the same department. Both share an interest on longitudinal data modeling, missing-data problems, and HIV epidemiology.

The Stata Journal (2010)
10, Number 2, pp. 252–258

Computing Murphy–Topel-corrected variances in a heckprobit model with endogeneity

Juan Muro
Department of Statistics, Economic Structure,
and International Economic Organization
Universidad de Alcalá
Madrid, Spain

Cristina Suárez
Department of Statistics, Economic Structure,
and International Economic Organization
Universidad de Alcalá
Madrid, Spain

María del Mar Zamora
Department of Statistics, Economic Structure,
and International Economic Organization
Universidad de Alcalá
Madrid, Spain
mariam.zamora@uah.es

Abstract. We outline a fairly simple method to obtain in Stata Murphy–Topel-corrected variances for a two-step estimation of a heckprobit model with endogeneity in the main equation. The procedure uses `predict`'s `score` option and the powerful matrix tool `accum` in Stata and builds on previous works by Hardin (2002, *Stata Journal* 2: 253–266) and Hole (2006, *Stata Journal* 6: 521–529).

Keywords: st0191, binary choice model, heckprobit, selectivity, endogenous variables, two-step estimation, qualitative models, Murphy–Topel-corrected variances

1 Introduction

Probit models with selectivity, referred to as heckprobit models, are an important tool in empirical analysis. Estimating a heckprobit model in the presence of endogenous variables is usually achieved using a two-step method, though a full maximum likelihood (ML) method is also possible. In this article, we stress the relevance of obtaining a variance estimator (Murphy and Topel 2002; Hardin 2002) when a two-step estimation method is chosen, and we show a fairly simple procedure to compute Murphy–Topel-corrected variances in Stata. Our procedure builds on previous work by Hardin (2002) and Hole (2006) and illustrates the application of their approaches to a model with two index functions.

We organize the article as follows. In section 2, we describe our model. Section 3 contains the Stata procedure for computing Murphy–Topel-corrected variances and an illustration. Section 4 provides a brief summary.

2 A Murphy–Topel estimator for a heckprobit model with endogeneity in the equation of interest

The model considered is described by an extension of a well known result in the econometric literature first outlined by Lahiri and Schmidt (1978) and also discussed by Greene (1998, 2008).

As is well known, inefficient but consistent estimators of the parameters in the component models are given by the two-step procedure:

1. Estimate the reduced-form model for the endogenous variable by ML probit and obtain its predictions.
2. Substitute the predictions obtained in step 1 for the appropriate covariate column and estimate the heckprobit by ML.
3. Calculate appropriate corrected variance–covariance estimations; see Murphy and Topel (2002) and Greene (2008, 302–303).

We have to correct the estimated covariance matrix for the selectivity probit model in the second model. The unadjusted variance matrix is sometimes called the naïve covariance matrix because it assumes that the predicted values used as a covariate are measured without error.

A straightforward way of calculating the components of the Murphy–Topel variance expression for models with a simple index using Stata is described in detail in Hole (2006). We extend this approach in the next section for heckprobit models with endogeneity.

(*Continued on next page*)

3 Murphy–Topel-corrected variances

A sequence of commands to calculate the Murphy–Topel variance using Stata is described as follows:

```
use data
local x1 "country1-country4 aacc2-aacc6"

/* First stage: probit, save score as s0 */
probit y1 `x1´          /* `x1´ contains k1 variables
                           (included in k1 the constant) */
matrix V1 = e(V)        /* Variance estimate, matrix dimension (k1, k1) */
predict double y1hat    /* Generate prediction of endogenous variable for
                           second stage */
predict s0, score
```

As a result of the above Stata commands, we save the estimated covariance matrix of the probit equation, `V1`, and the predicted values of the endogenous variable to be included in the matrix of covariates of the second-stage model.

```
/*Second stage: heckprobit, save scores as s1, s2, and s3 */
local x2 "age24 age25_44 age45_64 country1-country4 aacc2-aacc6 y1hat"
local x3 "age24 age25_44 age45_64 border borderaacc"
heckprob y2 `x2´, select(y3 = `x3´)

/* `x2´ and `x3´ contain (k2-1) and k3 variables, respectively (included in
k2-1 and k3 the constant) */
matrix V2 = e(V)        /* Matrix dimension (k2+k3+1, k2+k3+1) */
scalar TP = _b[y1hat]   /* Coef. of endogenous variable in main equation */
matrix coef = e(b)      /* Vector dimension: k2+k3+1 */
predict s1 s2 s3, score
```

In the second stage, we obtain heckprobit ML estimates and the naïve covariance matrix. Table 1 shows two-step heckprobit ML estimation results, where standard errors, z statistics, probabilities, and confidence intervals derive from the naïve covariance matrix (the data and model come from Muro, Suárez, and Zamora [2006, 2009]).

Table 1. Two-step heckprobit estimation results (uncorrected covariance matrix)

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
y2						
age24	-.0458653	.033738	-1.36	0.174	-.1119905	.0202599
age25_44	-.1537932	.0282394	-5.45	0.000	-.2091414	-.098445
age45_64	-.0720658	.0262045	-2.75	0.006	-.1234256	-.020706
country1	-.6337745	.0913744	-6.94	0.000	-.812865	-.4546841
country2	.1011763	.0213756	4.73	0.000	.0592809	.1430717
country3	.3173457	.0176745	17.96	0.000	.2827044	.3519871
country4	-.2298831	.0296122	-7.76	0.000	-.287922	-.1718443
aacc2	.6911638	.0330778	20.90	0.000	.6263325	.755995
aacc3	.9601613	.2879347	3.33	0.001	.3958196	1.524503
aacc4	.8319112	.4004438	2.08	0.038	.0470557	1.616767
aacc5	.5350787	.0405529	13.19	0.000	.4555965	.6145609
aacc6	.6227361	.0574185	10.85	0.000	.5101979	.7352743
y1hat	-1.523668	.5041929	-3.02	0.003	-2.511868	-.5354684
_cons	-.5805467	.0394076	-14.73	0.000	-.6577843	-.5033092
y3						
age24	.9653056	.0344481	28.02	0.000	.8977886	1.032823
age25_44	.912114	.0241071	37.84	0.000	.8648649	.9593631
age45_64	.4015542	.0243098	16.52	0.000	.3539079	.4492006
border	-1.654292	.0162945	-101.52	0.000	-1.686229	-1.622356
borderaacc	-.8970486	.0150943	-59.43	0.000	-.9266328	-.8674644
_cons	1.139033	.0220057	51.76	0.000	1.095902	1.182163
/athrho	-.6547092	.0639639	-10.24	0.000	-.7800761	-.5293423
rho	-.5748316	.0428282			-.6527504	-.4848782

LR test of indep. eqns. (rho = 0): chi2(1) = 97.60 Prob > chi2 = 0.0000

Given the initial estimates, we calculate the $\widehat{\mathbf{C}}$ and $\widehat{\mathbf{R}}$ matrices described in Hardin (2002) and Hole (2006). For the sake of clarity, we remind readers that in a heckprobit model, we have censored and uncensored observations. Only uncensored observations enter into the main equation. Thus we can split summations into two parts: uncensored and censored. `s1` and `s3` scores computed in Stata are vectors with null values for censored observations, while `s2` has no null values in the whole sample.

Partial derivatives of the log likelihood of the second stage with respect to the parameter vector in the second stage have two components: the first one is the derivative with respect to the index, and the second one is the derivative of the index with respect to the parameter. The first component is the score vector calculated in Stata's `heckprob` command: `s1` for the parameters of the main equation, `s2` for the parameters of the selection equation, and `s3` for the correlation term ρ. The second component is a matrix with `'x2'`, `'x3'`, and a vector of 1s.

Partial derivatives of the log likelihood of the second stage with respect to the parameter vector in the first stage also have two components. The first component is the `s1` score vector, which has null values for censored observations. The second component is matrix `'x1'` times the estimated parameter of `y1hat` in the heckprobit

model times the derivative of `y1hat` with respect to the index function of the probit model. The formula is

$$\widehat{\mathbf{C}} = \breve{X}'\text{diag}\left(\texttt{s2} \times \texttt{s1}\frac{\partial \texttt{y1hat}}{\partial \text{`x1'}\theta_1}\widehat{\gamma}\right) \text{`x1'}$$

where $\breve{X}$ has as components `'x2'` times `s1/s2`, `'x3'`, and `s3/s2`; the derivatives in our probit model are $N(0,1)$ probability density function; and $\widehat{\gamma}$ is the estimated parameter of `y1hat` in the heckprobit model.

For matrix $\widehat{\mathbf{R}}$, a similar reasoning leads us to the formula

$$\widehat{\mathbf{R}} = \breve{X}'\text{diag}\left(\texttt{s2} \times \texttt{s0}\right) \text{`x1'}$$

with the equivalences noted above.

The Stata program continues as follows:

```
generate const = 1      /* Needed for the program */
local x2 "age24 age25_44 age45_64 country1 country2 country3 country4 aacc2 /*
        */      aacc3 aacc4 aacc5 aacc6 y1hat"
foreach a1 of local x2 {
        generate `a1´_s = `a1´ * s1/s2
}

/* s2 is the true score */
generate a_s = s1/s2
generate s3_s = s3/s2   /* Auxiliary parameter */
local x2_s "age24_s age25_44_s age45_64_s country1_s country2_s country3_s  /*
        */      country4_s aacc2_s aacc3_s aacc4_s aacc5_s aacc6_s y1hat_s"

/* For main and selection equations */
matrix accum C = `x1´ const `x2_s´  a_s  `x3´ const s3_s        /*
        */      [iw=s1*s2*(s0*((1-y1)+(2*y1-1)*y1hat)*(2*y1-1))*TP], noconstant

/*For main and selection equations*/
matrix accum R = `x1´ const `x2_s´  a_s  `x3´ const s3_s        /*
        */      [iw=s2*s0], noconstant

/* Get only the desired partition; see Hole (2006) */
matrix C = C[11..31,1..10]
matrix R = R[11..31,1..10]

/* For Murphy-Topel matrix */
matrix M = V2 + (V2 * (C*V1*C´ -R*V1*C´ -C*V1*R´) * V2)

capture program drop doit
matrix b = e(b)
program doit, eclass
        ereturn post b M
        ereturn local vcetype "Mtopel"
        ereturn display
end
doit
```

In the first `matrix accum` command, the term in brackets is equivalent to `normalden(xb)`.

Table 2 shows two-step heckprobit ML estimation results, where standard errors, z statistics, probabilities, and confidence intervals derive from the Murphy–Topel-corrected covariance matrix.

Table 2. Two-step heckprobit estimation results (Murphy–Topel-corrected covariance matrix)

	Mtopel Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
y2						
age24	-.0458653	.0337474	-1.36	0.174	-.1120089	.0202784
age25_44	-.1537932	.0282431	-5.45	0.000	-.2091486	-.0984378
age45_64	-.0720658	.026204	-2.75	0.006	-.1234248	-.0207068
country1	-.6337745	.0916261	-6.92	0.000	-.8133584	-.4541906
country2	.1011763	.0223539	4.53	0.000	.0573636	.1449891
country3	.3173457	.0179434	17.69	0.000	.2821774	.3525141
country4	-.2298831	.0299708	-7.67	0.000	-.2886249	-.1711414
aacc2	.6911638	.0338813	20.40	0.000	.6247577	.7575698
aacc3	.9601613	.2992127	3.21	0.001	.3737152	1.546607
aacc4	.8319112	.415564	2.00	0.045	.0174207	1.646402
aacc5	.5350787	.0414957	12.89	0.000	.4537486	.6164088
aacc6	.6227361	.0590565	10.54	0.000	.5069875	.7384847
y1hat	-1.523668	.522931	-2.91	0.004	-2.548594	-.4987424
_cons	-.5805467	.0395836	-14.67	0.000	-.6581292	-.5029643
y3						
age24	.9653056	.0344495	28.02	0.000	.8977859	1.032825
age25_44	.912114	.0241066	37.84	0.000	.8648659	.959362
age45_64	.4015542	.0243096	16.52	0.000	.3539084	.4492001
border	-1.654292	.0163054	-101.46	0.000	-1.68625	-1.622334
borderaacc	-.8970486	.015094	-59.43	0.000	-.9266324	-.8674649
_cons	1.139033	.022006	51.76	0.000	1.095902	1.182164
athrho						
_cons	-.6547092	.064002	-10.23	0.000	-.7801508	-.5292676

4 Summary

In this article, we demonstrate how the Murphy–Topel variance estimator for a heckprobit second-stage model can be estimated following the sequence of commands given by Hardin (2002) and Hole (2006).

5 Acknowledgments

The authors would like to thank A. R. Hole and the reviewers for helpful comments and suggestions that led to improvements in this article.

6 References

Greene, W. H. 1998. Gender economic courses in liberal arts colleges: Further results. *Journal of Economic Education* 29: 291–300.

———. 2008. *Econometric Analysis*. 6th ed. Upper Saddle River, NJ: Prentice Hall.

Hardin, J. W. 2002. The robust variance estimator for two-stage models. *Stata Journal* 2: 253–266.

Hole, A. R. 2006. Calculating Murphy–Topel variance estimates in Stata: A simplified procedure. *Stata Journal* 6: 521–529.

Lahiri, K., and P. Schmidt. 1978. On the estimation of triangular structural systems. *Econometrica* 46: 1217–1221.

Muro, J., C. Suárez, and M. M. Zamora. 2006. The demand for low-cost carriers: An empirical micro analysis. Unpublished manuscript.

———. 2009. Access and use of e-commerce in the Spanish tourism market. In *Advances in Tourism Destination Marketing*, ed. M. Kozak, J. Gnoth, and L. L. A. Andreu, 170–182. London: Routledge.

Murphy, K. M., and R. H. Topel. 2002. Estimation and inference in two-step econometric models. *Journal of Business and Economic Statistics* 20: 88–97.

About the authors

Juan Muro is a professor of economics at the Universidad de Alcalá, Spain. His research interests include microeconometrics, labor economics, duration models, treatment effects, and efficiency measures and production frontiers.

Cristina Suárez is a lecturer of econometrics in the Department of Statistics, Economic Structure, and International Economic Organization at the Universidad de Alcalá, Spain. Her main research interests are microeconometrics, applied industrial economics, and service markets.

María del Mar Zamora is a lecturer of econometrics in the Department of Statistics, Economic Structure, and International Economic Organization at the Universidad de Alcalá, Spain. Her current research projects focus on applied microeconometric models and tourism behavior.

The Stata Journal (2010)
10, Number 2, pp. 259–266

Multivariate outlier detection in Stata

Vincenzo Verardi
University of Namur
(Centre for Research in the Economics of Development)
Namur, Belgium
and Université Libre de Bruxelles
(European Center for Advanced Research in Economics and Statistics
and Center for Knowledge Economics)
Brussels, Belgium
vverardi@ulb.ac.be

Catherine Dehon
Université libre de Bruxelles
(European Center for Advanced Research in Economics and Statistics
and Center for Knowledge Economics)
Brussels, Belgium
cdehon@ulb.ac.be

Abstract. Before implementing any multivariate statistical analysis based on empirical covariance matrices, it is important to check whether outliers are present because their existence could induce significant biases. In this article, we present the minimum covariance determinant estimator, which is commonly used in robust statistics to estimate location parameters and multivariate scales. These estimators can be used to robustify Mahalanobis distances and to identify outliers. Verardi and Croux (1999, *Stata Journal* 9: 439–453; 2010, *Stata Journal* 10: 313) programmed this estimator in Stata and made it available with the `mcd` command. The implemented algorithm is relatively fast and, as we show in the simulation example section, outperforms the methods already available in Stata, such as the Hadi method.

Keywords: st0192, mcd, detection, multivariate outliers, robustness, minimum covariance determinant

1 Multivariate detection

One of the principal objectives of statistics is to provide tools to describe multidimensional relations. However, if these tools are based on classical estimation methods and outliers are present in the dataset, the results can be biased. To deal with this drawback, it could be envisaged to identify the outliers and either downweight these observations or remove them from the dataset before calling on the statistical tool. Unfortunately, the identification of outliers is quite challenging because a visual inspection is troublesome when considering more than two dimensions.

Traditionally, the (multivariate) characterization of an outlier is measured by Mahalanobis distances, defined as

$$d_i = \sqrt{(X_i - \mu)\Sigma^{-1}(X_i - \mu)'} \tag{1}$$

where X is the $n \times p$ matrix containing the realizations of p random variables for n individuals, X_i is the ith row vector of matrix X, μ is the $1 \times p$ multivariate location vector, and Σ is the $p \times p$ covariance matrix. These distances are well known to be distributed as the square root of a chi-squared distribution with p degrees of freedom ($\sqrt{\chi^2_p}$) when X comes from a multivariate normal distribution. It is then rational to set a critical value (for example, the 97.5th percentile of a $\sqrt{\chi^2_p}$) above which an individual observation can be considered an outlier. Unfortunately, if outliers are present, the classical estimation of μ (by the sample mean) is affected, and the estimation of Σ (by the sample covariance matrix) is inflated. Mahalanobis distances will hence be distorted. However, the identification of multivariate outliers using Mahalanobis distances is still possible if μ and Σ are robustly estimated (that is, estimated using a method that is not excessively affected by outliers).

In Stata, an estimator aimed at robustly estimating the multivariate outlyingness (see Hadi [1992,1994]) is available with the `hadimvo` command. Unfortunately, as will be shown in the simulation example section, the results of Hadi's estimator may be adversely affected by outliers under some contamination scenarios. The weakness of Hadi's method is the first step of the proposed iterative algorithm, which is based on ranking individuals by a measure that is not guaranteed to be robust to outliers.

In this article, we present the minimum covariance determinant (MCD) estimator of location and scatter, introduced by Rousseeuw (1985, 877),[1] that has been proven to be particularly well-suited in this context and has become standard in robust statistics. Furthermore, we also present the intuition behind the algorithm Verardi and Croux (2009, 2010) used to implement this method (the `mcd` command) in Stata.

The structure of the article is as follows. In section 2, we briefly describe the MCD estimator, and in section 3, we present the computational algorithm implemented in Stata. In section 4, we use simulations to illustrate how the MCD estimator outperforms other available estimators. In section 5, we present the proposed Stata command and, in section 6, we provide a short summary discussion.

2 MCD

Before presenting the MCD estimator, it is helpful to recall the notion of generalized variance. This measure, originally introduced by Wilks (1932), is a one-dimensional assessment of multidimensional spread. For the sake of clarity, we explain this concept calling on a 2×2 covariance matrix. The generalization to higher dimensions is straightforward.

1. These robust estimators are already implemented in S-plus, R, and SAS.

Let's define the covariance matrix

$$\Sigma = \begin{pmatrix} \sigma^2_{x_1} & \sigma_{x_1x_2} \\ \sigma_{x_1x_2} & \sigma^2_{x_2} \end{pmatrix}$$

where $\sigma^2_{x_1}$, $\sigma^2_{x_2}$, and $\sigma_{x_1x_2}$ are respectively the variance of variable x_1, the variance of variable x_2, and the covariance between variable x_1 and x_2. The generalized variance is defined as the determinant of Σ (i.e., $\sigma^2_{x_1}\sigma^2_{x_2} - \sigma^2_{x_1x_2}$). To understand why this measure can be seen as a generalization of the variance, it is helpful to look more closely at the expression of the determinant. This expression is composed of two elements: the product of $\sigma^2_{x_1}$ and $\sigma^2_{x_2}$, and (minus) the squared covariance $\sigma^2_{x_1x_2}$. The first term ($\sigma^2_{x_1}\sigma^2_{x_2}$) represents the raw bi-dimensional spread of the observations. However, if x_1 and x_2 are not independent, each of the variables conveys some information on the other one. This redundant information should be accounted for. What remains is the bi-dimensional spread once the covariation has been accounted for.

Having defined the generalized variance, it is now easy to present the underlying principle of MCD. Consider that we want an estimator of the covariance matrix that withstands a contamination of up to 50% of sample points[2] (it is then said to have a breakdown point of 50%). The basic idea of MCD is to identify the subsample containing 50% of the observations that is associated with the smallest generalized variance. This is equivalent to finding the subsample with the smallest multivariate spread. The MCD robust covariance matrix, labeled Σ_{MCD}, and the MCD location vector, labeled μ_{MCD}, are then defined as the sample covariance matrix[3] and location vector computed over this subsample.

To identify such a subsample, the idea is to consider all possible subsets containing 50% of the observations and flag the one with the smallest covariance matrix determinant. This is, of course, a cumbersome task. Imagine, for example, that we have a dataset of 100 observations. If we want to consider all possible subsamples containing 50 observations, the total number to check is $\begin{pmatrix} 100 \\ 50 \end{pmatrix} \simeq 1.01 \times 10^{29}$. Fortunately, as shown by Rousseeuw and van Driessen (1999) and briefly explained in the following section, only a limited number of subsets needs to be considered in practice.

3 Fast-MCD algorithm

A feasible and stable algorithm for computing the MCD location and covariance matrix has been the pitfall of the method for a long time. This problem was solved by Rousseeuw and van Driessen (1999), who proposed the fast-MCD algorithm, which is particularly efficient. Verardi and Croux (2009, 2010) programmed this algorithm in Stata and created the `mcd` command. For the sake of brevity, we present the stylized algorithm here and refer the interested reader to the original article for further details.

2. A similar reasoning can be adopted for any h% contamination where $h \leq 50$%.
3. Multiplied by a constant related to the assumed underlying distribution.

The algorithm is based on three steps. In the first step, N subsamples of size $p+1$ (called the p-subsets) are randomly drawn from the dataset. For each j of the N p-subsets, the covariance matrix $\widehat{\Sigma}_p^j$ and the vector of location parameters $\widehat{\mu}_p^j$ are computed. Then for each p-subset, the determinant of $\widehat{\Sigma}_p^j$ is calculated. If the determinant is null, the number of sample points of the subset is increased until the determinant becomes nonnull. In the second step, for each p-subset, Mahalanobis distances are calculated for all points as in (1), using $\widehat{\mu}_p^j$ and $\widehat{\Sigma}_p^j$. The observations are then sorted in ascending order of the estimated Mahalanobis distances. After that, $\widehat{\mu}_p^j$ and $\widehat{\Sigma}_p^j$ are updated to the sample average and covariance matrix estimated on the 50% first-ranked points. The procedure is repeated until convergence. In the third and final step, all the determinants of the final N covariance matrices are compared. The subset that is associated with the smallest determinant is kept to estimate the final MCD covariance matrix and location vector.

To increase efficiency, a one-step robust reweighted covariance matrix (called RMCD) is estimated, calculating the classic covariance matrix exclusively on the nonoutlying individuals (and multiplying it by a consistency factor).

To increase the speed of the algorithm, the second and third steps are performed only for the 10 p-subsets with the smallest preliminary determinants. The minimal number of p-subsets (N) to be considered can be determined according to the formula

$$N = \left\lceil \frac{\log(1 - P_{\text{clean}})}{\log\{1 - (1-\varepsilon)^p\}} \right\rceil \tag{2}$$

where P_{clean} is the (chosen) probability of having at least one noncorrupt p-subset (it is generally set to 0.99), ε is the maximal proportion of outliers that we expect to contaminate the dataset (it is generally set to 20%), and p is the number of variables. The rationale behind this formula is presented in the original article of Rousseeuw and van Driessen (1999).

4 Simulated example

To illustrate why MCD is particularly well suited for multivariate outlier identification, we create a dataset of size $n = 1,000$ by randomly generating 10 independent continuous variables (labeled $X_1, \ldots, X_{10}$) from a Gaussian distribution with mean 0 and unit variance. This dataset is called the clean dataset. Subsequently, we contaminate the dataset by randomly replacing 10% of the observations of the first variable with random values drawn from a Gaussian distribution with mean 5 and standard deviation 0.1. This is called the contaminated dataset.

We then run the `mcd` command (on both the clean and the corrupt dataset) to estimate the covariance matrix and compute the robust distances. We obviously would like the estimated covariance matrix to be close to the identity matrix in both the clean and the contaminated dataset (because this is the covariance matrix that was used to generate the vast majority of the observations). In this example, we compare three

estimated covariance matrices: the classic one, the classic one with outliers identified by the Hadi method removed (defined as Hadi), and the RMCD covariance matrix. The test used to check whether the underlying population covariance matrix is the identity matrix, is a standard likelihood-ratio test. The calculated test statistic is $W = -2\left[np/2 + n/2\log\left\{\det(\widehat{\Sigma})\right\} - n/2\text{trace}(\widehat{\Sigma})\right]$. As can be seen in table 1, before the contamination, all three estimation methods lead to an estimated covariance matrix that is not statistically different from the identity (p-value > 0.05). After the contamination, only the MCD estimated covariance matrix passes the test. As far as the MCD location vector is concerned, all elements are very close (and not statistically different) to 0.

Table 1. Likelihood-ratio test on Σ

	Clean		Contaminated	
	W	p-value	W	p-value
MCD	53.49	0.49	57.24	0.36
Hadi	52.55	0.53	872.15	0.00
Classic	49.73	0.64	1055.74	0.00

To focus on the identification of outliers, we present in figure 1 two distance–distance plots comparing the Mahalanobis distances based on MCD estimations of location and scatter (on the y axis) with (on the x axis) the Hadi based distances (on the left) and the classic Mahalanobis distances (on the right). For the sake of clarity, the artificially generated outliers are identified with hollow circles. A line is drawn on both axes in correspondence to the critical value $\left(\sqrt{\chi^2_{10,0.975}} = 4.52\right)$ to identify the value above which an individual can be considered an outlier.

(*Continued on next page*)

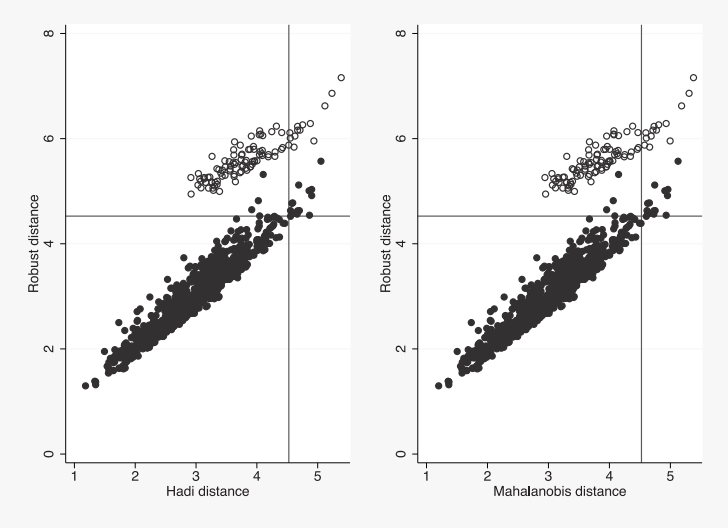

Figure 1. Distance–distance plots

The result is striking. MCD robust distances allow the identification of all generated outliers, because all hollow circles are above the horizontal line. On the other hand, the methods based on Mahalanobis and Hadi distances behave (equivalently) poorly, because almost all hollow circles are on the left of the vertical line. Obviously, and as expected, about 2.5% of sample points generated by the (uncontaminated) normal distribution are above the benchmark set for robust distances.

5 The MCD command

5.1 Syntax

The `mcd` command (Verardi and Croux 2009, 2010) estimates the robust (one-step, reweighted MCD) covariance matrix (and names it `covRMCD`), using the fast-MCD algorithm described above. `mcd` also calculates robust distances according to (1), where μ and Σ are estimated with MCD estimators of location and scatter. Type `search mcd, sj` and follow the online instructions to download `mcd` and its affiliated commands. The syntax of the `mcd` command is

`mcd` *varlist* [*if*] [*in*] [`, e(#) proba(#) trim(#) generate(`*newvar1 newvar2*`)`
`bestsample(`*newvar*`) raw setseed(#)`]

5.2 Options

The `mcd` command has seven options. The first two options allow the change of the number of p-subsets analyzed in the algorithm. More specifically, the `e(#)` option requests that Stata change the maximum expected percentage of outliers in the dataset [ε in (2)]. The default is `e(0.2)`, but it can take any value belonging to the interval $(0, 0.5)$. The `proba(#)` option requests that Stata fix the probability of having at least one noncorrupt p-subset among the N considered by the algorithm [P_{clean} in (2)]. The default is `proba(0.99)`, but it can take any value belonging to the interval $(0, 1)$.

The third option, `trim(#)`, allows the user to specify the percentage of outliers that the estimator can withstand before breaking up. The default is `trim(0.5)`, but it can take any value belonging to the interval $(0, 0.5)$. The fourth option, `generate()`, asks the user to provide two variable names that return identified outliers[4] and robust distances, respectively.

The fifth option, `bestsample()`, asks the user to provide a name for a variable that flags the points that are in the subset with the smallest covariance matrix determinant. The sixth option, `raw`, specifies that Stata return the genuine MCD location and scatter matrices rather than the one-step, reweighted MCD (the default).

The last option, `setseed(#)`, allows the user to specify the seed (to a value chosen by the user). By default, the seed is not set, which means that results may change over replications; see [R] **set seed** for information on setting the seed.

6 Conclusion

Several multivariate outlier detection tools are available in statistical software. Unfortunately, most of them use estimates of the covariance that are sensitive to the presence of the outliers to be detected, thus leading to masking and swamping effects. Several robust alternatives have been proposed, but the complexity behind their practical implementation is such that they did not manage to emerge as standard tools used by applied researchers. In this article, we used the `mcd` command (which allows a robust identification of multivariate outliers) that Verardi and Croux (2009, 2010) programmed in Stata to tackle the problem. To illustrate the usefulness of the method, we ran some simulations to show how it dramatically outperforms the previously available methods.

`mcd` also provides robust estimations of multivariate location and scatter. Relying on these, it is now easy to implement robust multivariate statistical techniques. Such an approach was taken for robust principal component analysis (Croux and Haesbroeck 2000), discriminant analysis (Hubert and Van Driessen 2004), or canonical correlations analysis (Croux and Dehon 2002) by simply replacing the classic estimators with the MCD ones. Moreover, as shown by Verardi and Croux (2009) in a companion article, in the context of regression analysis, the identification of multivariate outliers among explanatory variables allows for the detection of leverage points.

4. That is, points with a robust distance larger than $\sqrt{\chi^2_{p,0.975}}$.

7 Acknowledgments

We would like to thank Christophe Croux for his precious comments. Vincenzo Verardi is an associate researcher of the National Science Foundation of Belgium and gratefully acknowledges their financial support.

8 References

Croux, C., and C. Dehon. 2002. Analyse canonique basée sur des estimateurs robustes de la matrice de covariance. *Revue de Statistique Appliquée* 2: 5–26.

Croux, C., and G. Haesbroeck. 2000. Principal component analysis based on robust estimators of the covariance or correlation matrix: Influence functions and efficiencies. *Biometrika* 87: 603–618.

Hadi, A. S. 1992. Identifying multiple outliers in multivariate data. *Journal of the Royal Statistical Society, Series B* 54: 761–771.

———. 1994. A modification of a method for the detection of outliers in multivariate samples. *Journal of the Royal Statistical Society, Series B* 56: 393–396.

Hubert, M., and K. Van Driessen. 2004. Fast and robust discriminant analysis. *Computational Statistics and Data Analysis* 45: 301–320.

Rousseeuw, P. J. 1985. Multivariate estimation with high breakdown point. In *Mathematical Statistics and Applications*, ed. W. Grossmann, G. Pflug, I. Vincze, and W. Wertz, vol. 8, 283–297. Dordrecht: Reidel.

Rousseeuw, P. J., and K. van Driessen. 1999. A fast algorithm for the minimum covariance determinant estimator. *Technometrics* 41: 212–223.

Verardi, V., and C. Croux. 2009. Robust regression in Stata. *Stata Journal* 9: 439–453.

———. 2010. Software Update: st0173_1: Robust regression in Stata. *Stata Journal* 10: 313.

Wilks, S. S. 1932. Certain generalizations in the analysis of variance. *Biometrika* 24: 471–494.

About the authors

Vincenzo Verardi has a PhD in economics from the Université Libre de Bruxelles. He is an associate researcher of the National Science Foundation of Belgium and is a professor of economics and econometrics at the University of Namur and the Université Libre de Bruxelles. His research fields are applied econometrics, robust methods, political economy, and public economics.

Catherine Dehon has a PhD in statistics from the Université Libre de Bruxelles. She is an associate professor of statistics and econometrics at Université Libre de Bruxelles. Her current fields of research include robust regression, robust multivariate analysis, and recently, the robustification of econometric methods.

The Stata Journal (2010)
10, Number 2, pp. 267–280

Data envelopment analysis

Yong-bae Ji
Korea National Defense University
Seoul, Republic of Korea
jyb77@hanmail.net

Choonjoo Lee
Korea National Defense University
Seoul, Republic of Korea
sarang90@kndu.ac.kr

Abstract. In this article, we introduce a user-written data envelopment analysis command for Stata. Data envelopment analysis is a linear programming method for assessing the efficiency and productivity of units called decision-making units. Over the last decades, data envelopment analysis has gained considerable attention as a managerial tool for measuring performance of organizations, and it has been used widely for assessing the efficiency of public and private sectors such as banks, airlines, hospitals, universities, defense firms, and manufacturers. The `dea` command in Stata will allow users to conduct the standard optimization procedure and extended managerial analysis. The `dea` command developed in this article selects the chosen variables from a Stata data file and constructs a linear programming model based on the selected `dea` options. Examples are given to illustrate how one could use the code to measure the efficiency of decision-making units.

Keywords: st0193, dea, data envelopment analysis, linear programming, nonparametric, efficiency, decision-making units

1 Introduction

In this article, we introduce a new application in Stata for performance measurement of decision-making units (DMUs) using data envelopment analysis (DEA) techniques. DEA is a nonparametric linear programming method for assessing the efficiency and productivity of DMUs. DEA application areas have grown since it was first introduced as a managerial and performance measurement tool in the late 1970s. Since then, new applications with more variables and complicated models have been and are being introduced.

Stata equipped with the `dea` command will provide the user with a new nonparametric tool to analyze productivity data. From within Stata, users will be able to produce DEA scores and analyze them. Because second-stage DEA analysis and DEA efficiency estimates involve statistical inference, DEA users need a software package that can analyze the whole process in one system. The alternative is juggling between a DEA command and a statistical software that uses the DEA scores as dependent variables to find the influential variables in second-stage DEA analysis.

The main purpose of this article is to implement the `dea` command in Stata. The article unfolds as follows. The next section describes the DEA models and calculations in DEA. The remainder of this article illustrates the features and options of the `dea` command.

2 The basics of DEA

DEA is a method for measuring efficiency of DMUs using linear programming techniques to envelop observed input–output vectors as tightly as possible (Boussofiane, Dyson, and Thanassoulis 1991). DEA allows multiple inputs–outputs to be considered at the same time without any assumption on data distribution. In each case, efficiency is measured in terms of a proportional change in inputs or outputs. A DEA model can be subdivided into an input-oriented model, which minimizes inputs while satisfying at least the given output levels, and an output-oriented model, which maximizes outputs without requiring more of any observed input values.

DEA models can also be subdivided in terms of returns to scale by adding weight constraints. Charnes, Cooper, and Rhodes (1978) originally proposed the efficiency measurement of the DMUs for constant returns to scale (CRS), where all DMUs are operating at their optimal scale. Later Banker, Charnes, and Cooper (1984) introduced the variable returns to scale (VRS) efficiency measurement model, allowing the breakdown of efficiency into technical and scale efficiencies in DEA.

In figure 1, frontiers determined by economies of scale are presented considering one input and one output for 5 DMUs labeled A through E. The CRS, VRS, and nonincreasing returns to scale frontiers are displayed in the figure. If CRS is assumed, then only DMU C would be efficient; DMUs A, C, and E are efficient if VRS is assumed. Where the nonincreasing returns to scale and VRS frontiers are equal, decreasing returns to scale exist for those DMUs on the efficient frontier (such as E). Where the two frontiers are unequal, then increasing returns to scale exist for those DMUs (such as DMU B). The remaining DMUs, which are inefficient, can be classified as increasing returns to scale if the sum of the reference weights is less than unity for the CRS frontier or as decreasing returns to scale otherwise.

The efficiency of observation B is defined as $\theta_{B,\text{input,CRS}} = \overline{B_0B_1}/\overline{B_0B}$ for the input-oriented CRS DEA model and represents that one can obtain the same output by reducing the input by the ratio of $1 - \theta_{B,\text{input,CRS}}$. The efficiency for the output-oriented CRS DEA model is defined as $\theta_{B,\text{output,CRS}} = \overline{B_3B}/\overline{B_3C}$ and represents that one can obtain the same input by increasing the output by the ratio of $1 - \theta_{B,\text{output,CRS}}$. Accordingly, the input-oriented efficiency relative to the VRS frontier is defined as $\theta_{B,\text{input,VRS}} = \overline{B_0B_2}/\overline{B_0B}$. All efficiency measures of DMU C are the same regardless of orientation because the frontiers meet at point C.

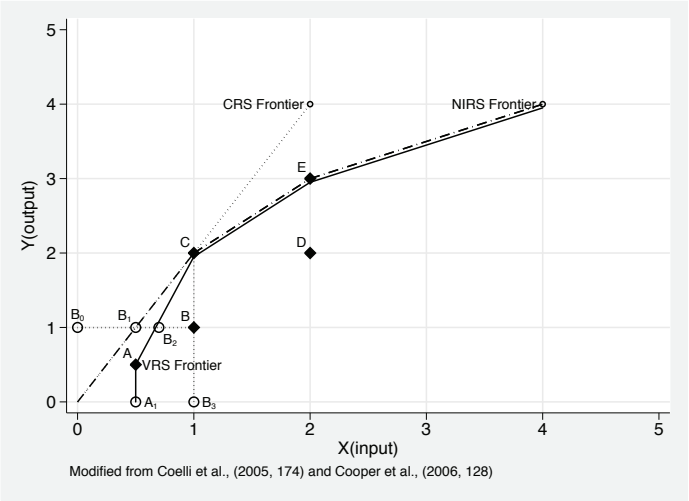

Figure 1. Concepts of efficiency and returns to scale

It is possible to decompose the CRS technical inefficiency into scale efficiency and "pure" technical efficiency. In figure 1, $\overline{B_2B}$ contributes to the technical efficiency of point B regarding the VRS model, and $\overline{B_1B}$ contributes to the technical efficiency of point B regarding the CRS model. Then $\overline{B_1B_2}$ contributes to scale efficiency.

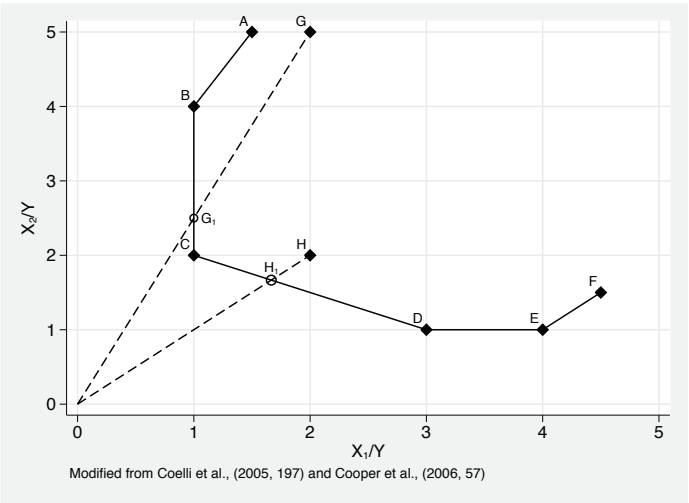

Figure 2. CRS input-oriented DEA example

Figure 2 illustrates the concepts of efficiency, slacks, and references or peers in an intuitive manner using two inputs and one output. The concept of frontier is especially important for the analysis of efficiency, because we measure efficiency as the relative

distance to the frontier. For example, firms that are technically inefficient operate at points in the interior of the frontier, while those that are technically efficient operate somewhere along the technology defined by the frontier. The DMU is called efficient when the DEA score equals 1 and all slacks are 0 (Cooper, Seiford, and Tone 2006). If only the first condition is satisfied, the DMU is called efficient in terms of "radial", "technical", and "weak" efficiency. If both conditions are satisfied, the DMU is called efficient in terms of "Pareto–Koopmans" or "strong" efficiency. The technical efficiencies of DMUs G and H are defined as $\overline{OG_1}/\overline{OG}$ and $\overline{OH_1}/\overline{OH}$, respectively.

Inefficiency can be seen as how much the inputs must contract along a ray from the origin until the ray crosses the frontier. For example, for firm G, the measure of technical efficiency is $\overline{OG_1}/\overline{OG}$. Point G_1 is the Farrell efficient point; however, input X_2 could be further reduced and still produce the same output. For this case, firm G has input slack $\overline{CG_1}$. If we disregard the slack and calculate it residually, the DEA model becomes the single-stage DEA model. The way to reduce the slack and find the Pareto optimal reference set can be further discussed; there are two-stage and multistage DEA models available in the literature (Cooper, Seiford, and Tone 2006; Coelli et al. 2005).

Cherchye and Puyenbroeck (2001) showed that "most representative efficient points" can be found using a direct approach and may differ from those obtained by multistage DEA. The DEA model in this article provides stage options for single-stage and two-stage which are still the most prevalent approaches in DEA literature. Reference or peer is a point that an inefficient DMU, such as point G, targets to move from the Farrell efficient point, such as point G_1, to the Pareto–Koopman efficient point, such as point C, in figure 2 (based on the two-stage DEA solution or Pareto optimal solution). However, the slack issues in DEA models disappear as the number of DMUs increases because the DEA piecewise linear frontier becomes smoother and has fewer chances to run the Farrell point to the input or output axes.

Free disposability means that one can produce the same output by wasting resources or increase the output without increasing resources. Strong disposability assumes that it is costless for firms to dispose of inputs or outputs or the isoquant does not bend backwards. In figure 2, the line that links A and F represents the frontier imposed by weak disposability. The line that links B and E represents the frontier imposed by strong disposability.

Assuming the economic production activities, convexity, strong disposability, and CRS, we can develop the linear program as a type of piecewise linear frontier. Input-oriented CRS efficiency is defined as (1) by applying the piecewise linear frontier to the input requirement set (Cooper, Seiford, and Tone 2006). This enables us to evaluate the efficiency relative to the frontier.

$$\max_{\nu,u} z = uy_j \tag{1}$$

subject to $\nu x_j = 1$, $-\nu X + uY \leq 0$, $\nu \geq 0$, $u \geq 0$, and u_j being free in sign, where a set of observed DMUs is $\text{DMU}_j, j = 1, \ldots, n$; x_j and y_j are input and output vectors; u is the row vector; ν are the output and input multipliers; and X and Y are the input and output matrices. The goal of the input-oriented DEA model is to minimize

the virtual input, relative to a given virtual output, subject to the constraint that no DMU can operate beyond the production possibility set and the constraint relating to nonnegative weights. In practice, most of the available DEA programs use the dual forms as expressed in (2), which lower the calculation burden and are virtually the same as (1).

$$\min_{\theta,\lambda} \theta \tag{2}$$

subject to $\theta x_j - X\lambda \geq 0$, $Y\lambda \geq y_j$, and $\lambda \geq 0$, where λ is a semipositive vector in R^k and θ is a real variable. The computational procedure for (2) can be expressed as

$$\min_{\theta} \theta \tag{3}$$

$$\min_{\lambda,s^+,s^-} \sum -s^+ - s^- \tag{4}$$

subject to $\theta x_j - X\lambda - s^- = 0$, $Y\lambda + s^+ = y_j$, and $\lambda \geq 0$, where s^+, s^-, and λ are semipositive vectors in R^k and θ is a real variable. The single-stage DEA model solves (3), and the two-stage DEA model solves (3) followed by (4), consecutively. The input-oriented CRS model was introduced in this section; however, other variations are easily extendable and available in most DEA literature, including Coelli et al. (2005) and Cooper, Seiford, and Tone (2000, 2006).

3 The dea command

3.1 Syntax

The syntax of the `dea` command is

```
dea ivars = ovars [if] [in] [, rts(crs|vrs|drs|nirs) ort(in|out)
   stage(1|2) trace saving(filename) ]
```

where *ivars* and *ovars* are input and output variable lists, respectively.

3.2 Description

`dea` requires the user to select the input and output variables from the user-designated data file or in the dataset currently in memory and solves DEA models with the specifications set in the options specified. There are several options to enhance the models. The user can select the desired options according to the particular model that is required.

The `dea` command requires an initial dataset that contains the input and output variables for observed DMU. Variable names must be identified by *ivars* for input variables and by *ovars* for output variables so that the `dea` command can identify and handle the multiple input–output dataset. In the output of the `dea` command, the prefix `dmu:` precedes DMU names.

The command has the ability to accommodate an unlimited number of inputs and outputs with an unlimited number of DMUs. The only limitation is the available computer memory. The resulting file reports information including reference points and slacks in the DEA model. This information can be used to analyze the inefficient DMU, for example, the source of the inefficiency and how an inefficient unit could be improved to the desired level.

`saving(`*filename*`)` creates *filename*`.dta`, which contains the results of `dea`, including information about the DMUs, inputs and outputs the data used, ranks of DMUs, efficiency scores, reference sets, and slacks. The log file `dea.log` will be created in the working directory.

Based on the data and the options specified, the `dea` command conducts matrix operations and linear programming to produce a results dataset that is available to print or can be used for further analysis.

3.3 Options

`rts(crs | vrs | drs | nirs)` specifies the returns to scale. The default, `rts(crs)`, specifies constant returns to scale. `rts(vrs)`, `rts(drs)`, and `rts(nirs)` specify variable returns to scale, decreasing returns to scale, and nonincreasing returns to scale, respectively.

`ort(in | out)` specifies the orientation. The default is `ort(in)`, meaning input-oriented DEA. `ort(out)` is output-oriented DEA.

`stage(1 | 2)` specifies the way to identify all efficiency slacks. The default is `stage(2)`, meaning two-stage DEA. `stage(1)` is single-stage DEA.

`trace` specifies to save all the sequences displayed in the Results window in the `dea.log` file. The default is to save the final results in the `dea.log` file.

`saving(`*filename*`)` specifies that the results be saved in *filename*`.dta`. If *filename*`.dta` already exists, the existing data will be moved to the file *filename*`_bak_`*DMYhms*`.dta` before the new data are saved in *filename*`.dta`.

3.4 Saved results

`dea` saves the following in `r()`:

Matrices

`r(dearslt)`	$n \times m$ matrix of the results of `dea`, where n is the number of DMUs and m is dependent on the model specified. Rows correspond to DMUs and columns correspond to variables, including inputs, outputs, rank (of DMU scores), theta (efficiency scores), ref. (reference DMUs), input and output slacks, and more, depending on the model specified.

4 Applications of dea

4.1 Data

This section provides examples using data from Cooper, Seiford, and Tone (2006, 75, table 3.7) and Coelli et al. (2005, 175, table 6.4) for illustration of the `dea` command. The data of Cooper, Seiford, and Tone (2006) consist of five stores that use two inputs—`i_employees` (number of employees as an input variable) and `i_area` (the area of floor as an input variable)—to produce two outputs: `o_sales` (the volume of sales as an output variable) and `o_profits` (the volume of profits as an output variable). The data of Coelli et al. (2005) consist of five firms that use one input, `i_1`, to produce one output, `o_1`.

4.2 CRS input-oriented two-stage DEA model

The `dea` default specifies a CRS input-oriented two-stage DEA model. If you want to use this specification for your analysis, just use the `dea` command as we have below using `cooper_table3.7.dta`. Then you will have the following results. Store E is the only efficient DMU and is the referent for all other stores, which is an equivalent result to Cooper, Seiford, and Tone (2006, 75–76). The two-stage DEA model provides the optimal solution, as shown in the results table.

For example, the optimal solution of efficiency score (theta), reference weights (λ_A, λ_B, λ_C, λ_D, λ_E), and slack (`i_area`, `i_employee`, `o_sales`, `o_profits`) for Store A are 0.933333, $(0, 0, 0, 0, 0.777778)$, and $(11.6667, 0, 0, 0, 0.222222)$, respectively. Thus the performance of Store A can be improved by subtracting 11.6667 units from input (area) and 0.777778 unit from output (profits) even after they have reduced all inputs by 6.67% without worsening any other input and output. Because Store A has an efficient score of 93.33%, all inputs (employee and area) could be reduced by 6.67%. In addition, because Store A has an input slack for area of 11.67, 11.67 units of area could be reduced even after Store A has reduced all inputs by 6.67%.

(*Continued on next page*)

```
. use cooper_table3.7.dta
. dea i_employee i_area = o_sales o_profits
```

```
options: RTS(CRS) ORT(IN) STAGE(2)
                                                  ref:
                    rank       theta     store_A
dmu:store_A            2     .933333           .
dmu:store_B            3     .888889           .
dmu:store_C            5     .533333           .
dmu:store_D            4     .666667           .
dmu:store_E            1           1           .

                    ref:        ref:        ref:
                 store_B     store_C     store_D
dmu:store_A            .           .           .
dmu:store_B            .           .           .
dmu:store_C            .           .           .
dmu:store_D            .           .           .
dmu:store_E            .           .           .

                    ref:     islack:     islack:
                 store_E  i_employee      i_area
dmu:store_A     .777778           .     11.6667
dmu:store_B     1.11111           .     3.33333
dmu:store_C     .888889           .           8
dmu:store_D     1.11111     3.33333           .
dmu:store_E           1           .           0

                 oslack:     oslack:
                o_sales   o_profits
dmu:store_A    2.98e-07     .222222
dmu:store_B    7.45e-07     5.88889
dmu:store_C    4.17e-06     2.11111
dmu:store_D    2.98e-06     6.88889
dmu:store_E           .           .
```

4.3 CRS output-oriented single-stage DEA model

If you want to use the CRS output-oriented single-stage DEA model, you can use the `dea` command as we have below using `coelli_table6.4.dta`.

```
. use coelli_table6.4.dta
. dea i_x = o_q, rts(crs) ort(o) stage(1)
```

```
options: RTS(CRS) ORT(OUT) STAGE(1)
CRS-OUTPUT Oriented DEA Efficiency Results:
                              ref:     ref:     ref:     ref:     ref:
          rank    theta         A        B        C        D        E
dmu:A        4       .5         .        .  .333333        .        .
dmu:B        4       .5         .        .  .666667        .        .
dmu:C        1        1         .        .        1        .        .
dmu:D        3       .8         .        .  1.33333        .        .
dmu:E        2  .833333         .        .  1.66667        .        .
         islack:  oslack:
            i_x      o_q
dmu:A        .        .
dmu:B        .        .
dmu:C        .        .
dmu:D        .        .
dmu:E        .        .
```

The rank of DMUs and efficiency score (`theta`), as well as the residually given reference set (`ref:`) and the slacks (`islack:` or `oslack:`), are listed in the above results. Store C is the only efficient DMU and seems to be the referent for all other stores. The Results window will display the above result, and a `dea.log` file that contains the above result will be created in your working directory. The "`.`" in the results table represent small numbers less than 10 to the minus 12 power, which mostly can be ignored. However, sometimes when you want to analyze financial data, the distinction between zero and "`.`" values may be required to maintain accuracy.

4.4 VRS input-oriented single-stage DEA model

Now we illustrate the VRS input-oriented single-stage DEA analysis using the data of `coelli_table6.4.dta`:

(*Continued on next page*)

```
. dea i_x = o_q, rts(vrs) stage(1)
```

```
options: RTS(VRS) ORT(IN) STAGE(1)
VRS-INPUT Oriented DEA Efficiency Results:
                              ref:     ref:     ref:     ref:     ref:
          rank    theta         A        B        C        D        E
dmu:A        1        1         1        .        .        .        .
dmu:B        5     .625        .5        .       .5        .        .
dmu:C        1        1         0        .        1        .        .
dmu:D        4       .9         .        .       .5        .       .5
dmu:E        1        1         .        .        0        .        1

         islack:  oslack:
            i_x      o_q
dmu:A         .        0
dmu:B         .        .
dmu:C         .        .
dmu:D         .        .
dmu:E         .        .

VRS Frontier(-1:drs, 0:crs, 1:irs)
           CRS_TE     VRS_TE    NIRS_TE      SCALE        RTS
dmu:A    0.500000   1.000000   0.500000   0.500000   1.000000
dmu:B    0.500000   0.625000   0.500000   0.800000   1.000000
dmu:C    1.000000   1.000000   1.000000   1.000000   0.000000
dmu:D    0.800000   0.900000   0.900000   0.888889  -1.000000
dmu:E    0.833333   1.000000   1.000000   0.833333  -1.000000

VRS Frontier:
```

	dmu	o_q	i_x	CRS_TE	VRS_TE	SCALE	RTS
1.	A	1	2	0.500000	1.000000	0.500000	irs
2.	B	2	4	0.500000	0.625000	0.800000	irs
3.	C	3	3	1.000000	1.000000	1.000000	-
4.	D	4	5	0.800000	0.900000	0.888889	drs
5.	E	5	6	0.833333	1.000000	0.833333	drs

If `rts(vrs)` is specified, the results show some additional information, as shown above. Stores D and E are on the decreasing returns to scale (`drs`) portion of the VRS frontier. On the other hand, Stores A and B are on the increasing returns to scale (`irs`) portion of the VRS frontier.

4.5 VRS input-oriented two-stage DEA model

Here we illustrate the VRS input-oriented two-stage DEA model, again using data from `coelli_table6.4.dta`:

```
. dea i_x = o_q, rts(vrs) ort(i)
```

```
options: RTS(VRS) ORT(IN) STAGE(2)
VRS-INPUT Oriented DEA Efficiency Results:
                              ref:     ref:     ref:     ref:     ref:
          rank    theta         A        B        C        D        E
dmu:A        1        1         1        .        0        .        .
dmu:B        5     .625        .5        .       .5        .        .
dmu:C        1        1         .        .        1        .        .
dmu:D        4       .9         .        .       .5        .       .5
dmu:E        1        1         .        .        0        .        1

         islack:  oslack:
            i_x      o_q
dmu:A         .        .
dmu:B         .        .
dmu:C         .        .
dmu:D         .        .
dmu:E         0        .

VRS Frontier(-1:drs, 0:crs, 1:irs)
           CRS_TE     VRS_TE    NIRS_TE      SCALE        RTS
dmu:A    0.500000   1.000000   0.500000   0.500000   1.000000
dmu:B    0.500000   0.625000   0.500000   0.800000   1.000000
dmu:C    1.000000   1.000000   1.000000   1.000000   0.000000
dmu:D    0.800000   0.900000   0.900000   0.888889  -1.000000
dmu:E    0.833333   1.000000   1.000000   0.833333  -1.000000

VRS Frontier:
```

	dmu	o_q	i_x	CRS_TE	VRS_TE	SCALE	RTS
1.	A	1	2	0.500000	1.000000	0.500000	irs
2.	B	2	4	0.500000	0.625000	0.800000	irs
3.	C	3	3	1.000000	1.000000	1.000000	-
4.	D	4	5	0.800000	0.900000	0.888889	drs
5.	E	5	6	0.833333	1.000000	0.833333	drs

Note that the efficiency score (theta) of DMU B is 0.625, and DMUs A and C are the reference DMUs for DMU B. The sum of the reference weights should equal 1 because `rts(vrs)` specifies that $\sum_{j=1}^{n} \lambda_j = 1$. The sum of the reference weights for DMU B equals 1 from $(\lambda_A, \lambda_B, \lambda_C, \lambda_D, \lambda_E) = (0.5, 0, 0.5, 0, 0)$. And note that the efficiency scores of all the DMUs are not changed from the single-stage model shown in the previous section to the current two-stage analysis because there is no slack in this case. This means that the slack level of profits has no effect on the efficiency evaluation. Stores A, C, and E are the efficient points that inefficient DMUs (B and D) can target to move in input-oriented DEA calculation.

Here we analyze data from Coelli et al. (2005, 175) using the `saving()` option:

```
. dea i_x = o_q, rts(vrs) ort(i) saving(coelli_6.4_results)
```

(*output omitted*)

Specifying `saving(coelli_6.4_results)` will save the VRS frontier results, shown above, as `coelli_6.4_results.dta`. The results match the Pareto–Koopman solution of Coelli et al. (2005, 176, table 6.5) because slack has no role in this case.

4.6 Second-stage regression analysis using the efficiency scores

The prevalent method in the literature to find the determinants of efficiency gaps among DMUs is by using tobit regression analysis because the efficiency scores are censored at the maximum value of the efficiency scores. The tobit regression analysis uses the efficiency scores as the dependent variables for the possible candidates of influential variables (see, for example, Lee, Lee, and Kim [2009]).

Here we illustrate the second-stage regression analysis using `seco_stage_regression.dta`.

```
. dea i_x = o_q, saving(seco_stage_regre_results)
  (output omitted)
. use seco_stage_regre_results
. tobit theta rnd, ul(1)
Tobit regression                                  Number of obs   =         20
                                                  LR chi2(1)      =      46.44
                                                  Prob > chi2     =     0.0000
Log likelihood =  19.060498                       Pseudo R2       =     5.5845

------------------------------------------------------------------------------
       theta |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         rnd |   .1464713   .0118676    12.34   0.000     .1216322    .1713104
       _cons |   .3003908   .0360827     8.33   0.000     .2248688    .3759127
-------------+----------------------------------------------------------------
      /sigma |   .0649661    .011565                      .0407603     .089172
------------------------------------------------------------------------------
  Obs. summary:          0  left-censored observations
                        16     uncensored observations
                         4 right-censored observations at theta>=1
```

The results show that the `rnd` (R&D) level is positively related with the CRS efficiency scores of DMUs at the 1% level of significance.

5 Conclusion

Today, many academic researchers recognize Stata as one of the leading packages for statistical analysis; however, there are still uncovered areas that managerial organizations are interested in. In particular, optimization procedures in Stata can be further developed to fill in the gaps between parametric and nonparametric analysis. The `dea` command introduced in this article is a new application in Stata and is a powerful managerial tool for measuring the efficiency and productivity of DMUs.

The `dea` command application has several advantages, including the following:

- It can be used by Stata users with no extra cost for DEA software.
- It is flexible to add other DEA models.
- It provides Stata with managerial tools for reports and statistical analysis, as well as optimization procedures.
- The `dea` command report files can directly feed to other Stata routines for further analysis.

6 Acknowledgments

We thank H. Joseph Newton and an anonymous reviewer for comments.

7 References

Banker, R. D., A. Charnes, and W. W. Cooper. 1984. Some models for estimating technical and scale inefficiencies in data envelopment analysis. *Management Science* 30: 1078–1092.

Boussofiane, A., R. G. Dyson, and E. Thanassoulis. 1991. Applied data envelopment analysis. *European Journal of Operational Research* 52: 1–15.

Charnes, A., W. W. Cooper, and E. Rhodes. 1978. Measuring the efficiency of decision making units. *European Journal of Operational Research* 2: 429–444.

Cherchye, L., and T. V. Puyenbroeck. 2001. A comment on multi-stage DEA methodology. *Operations Research Letters* 28: 93–98.

Coelli, T. J., D. S. P. Rao, C. J. O'Donnell, and G. E. Battese. 2005. *An Introduction to Efficiency and Productivity Analysis*. 2nd ed. New York: Springer.

Cooper, W. W., L. M. Seiford, and K. Tone. 2000. *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software*. 2nd ed. New York: Springer.

———. 2006. *Introduction to Data Envelopment Analysis and Its Uses*. New York: Springer.

Lee, C., J. Lee, and T. Kim. 2009. Innovation policy for defense acquisition and dynamics of productive efficiency: A DEA application to the Korean defense industry. *Asian Journal of Technology Innovation* 17: 151–171.

About the authors

Yong-bae Ji is a graduate student in the Defense Science and Technology Department at Korea National Defense University in Seoul, Republic of Korea.

Choonjoo Lee (corresponding author) is an assistant professor in the Defense Science and Technology Department at Korea National Defense University in Seoul, Republic of Korea.

The Stata Journal (2010)
10, Number 2, pp. 281–296

Speaking Stata: Finding variables

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

Abstract. A standard task in data management is producing a list of variable names showing which variables have specific properties, such as being of string type, or having value labels attached, or having a date format. A first-principles strategy is to loop over variables, checking one by one which have the property or properties concerned. I discuss this strategy in detail with a variety of examples. A canned alternative is offered by the official command `ds` or by a new command, `findname`, published formally with this column.

Keywords: dm0048, findname, ds, variable names, variable labels, value labels, types, formats, characteristics, data management

1 Introduction

In early versions of Stata, users were typically dealing with at most tens of variables. In recent versions, users may be dealing with a hundred or even a thousand times more. Such a difference can turn some data-management tasks that are trivial in principle into labors that are substantial in practice, unless you have the right tools. This column focuses on a common need: finding which variables possess some specified property, such as being a string variable, having value labels attached, or having a date format. The common feature is needing a list of variable names that can be used in some later commands.

The problem is approached on two levels, a first-principles, or low-level, approach and a higher-level approach centered on a new command, `findname`, which is published formally with this article. The deeper principles are more important than any particular implementation: no command can cover all imaginable needs, and there will always be tasks that require a more fundamental approach.

The main difficulty in this problem is not that Stata will refuse to provide the information you need to answer the questions you have. On the contrary, information of all kinds on your observations and on the basic structure and extra properties of your dataset is typically only a command away. What can create difficulties, however, is that you can be swamped with information you do not need or presented with information in a form you cannot easily reuse, at least without retyping a lot of variable names. Such retyping is clearly unattractive, tedious, and error-prone.

A simple example illustrates the main dilemma. Say you notice that the string variables in your dataset make your Stata life more difficult by messing up calculations

or making them impossible. Some systematic treatment of your string variables is called for, perhaps applying `encode` or `destring` (see [D] **encode** or [D] **destring**) or perhaps moving them to one end of the dataset with `order` (see [D] **order**). The first need is to get the names of the variables concerned, which are likely to be scattered throughout the dataset. Perhaps the data came with one or more string identifiers, which may be among the first few variables in the dataset. But, equally likely in any substantial project, all sorts of extra string variables may have been created in approaching some problem and so are scattered through the dataset.

A few sessions with Stata will show users at least one way of approaching this problem. `describe` (see [D] **describe**) displays among other things the storage type of each variable, which for a string variable will be one of the `str` types. But no one wants to scroll through what may be screenfuls of `describe` output. The same applies to `codebook` (see [D] **codebook**) output, to mention another way to find out about the dataset. There should be better ways, and this column explains what they are.

2 The main idea

2.1 varlists: Lists of variable names

One of the first pieces of Stata jargon that users encounter is the term *varlist*. A varlist is a list of variable names. If you read in the canonical Stata `auto.dta` by typing `sysuse auto`, then `price mpg rep78` is a varlist applying to your data, because each of the variables named is part of that dataset. So also is `mpg make`, and so also is any other list containing one or more variable names. Getting varlists that name precisely the variables we care about—no more, no fewer—is thus the problem here.

We can easily imagine typing all the variable names for `auto.dta`, but the idea is not appealing, and early Stata teaching should cover abbreviated ways of catching some of or all the variable names. The simplest and most general abbreviation (often called a wildcard) is `*`, which catches all the variable names in the current dataset.

2.2 Looping over variables

A simple way to see which variables satisfy some property is just to loop over variables, examining each variable in turn. Spelling this out in a kind of pseudocode,

```
1. initialize list of desired variables to empty

2. for each variable that exists {
        2a. check whether it has the desired property
        2b. if it has {
                add name of variable to the list
            }
}
```

The basic idea, labeled 2, is that of a loop over variables with exactly the same structure as you would use for any number of mundane tasks—throwing out magazines you no longer need, deciding whether to attend concerts or movies in a program, or whatever it is.

The prerequisite step, labeled 1, is important too. If the place you store the list already holds stuff from some other task, then you may get slightly confused or even make incorrect decisions, just as you might if you added this week's shopping list to last week's.

Another common twist on the problem is that the complementary list may be useful too. In this example, a list of numeric variables is just as useful as a list of string variables. We will discuss complements later in the column.

The first part of the problem is thus how to loop. `foreach` (see [P] **foreach**) is the most general looping command in Stata and as such the key to finding variables. A basic tutorial was given in a previous column (Cox 2002) and so an introduction is not repeated here. That tutorial explains the basics of local macros, which appear in many later examples.

Focusing on `foreach` does not exclude the possibility of approaching problems with `forvalues` (see [P] **forvalues**), which in special cases could be simple and attractive. For example, if your variables had the names `var1`–`var42`, then `forvalues` would be an alternative to `foreach`. But this kind of situation is exceptional.

```
foreach v of var * {
        commands go here
}
```

is a loop over all variables. `var` is short for `varlist` and `*`, as mentioned earlier, includes all variables.

2.3 How to check properties: Extended macro functions

The other part of the problem is checking variable properties. Let us read in `auto.dta` and return to the example of finding string variables. In this case, it is easy to see (say, by using `describe`) that there is just one string variable, `make`. But how would we find that out automatically?

Frequently, a specific tool for solving your problem is given by a so-called extended macro function, documented in [P] **macro**. Several tools are documented under this heading. Thus the storage type of a variable in `auto.dta` is returned by the construct `: type`. This can be used to put the type in a local macro:

```
. local t : type make
```

or we could cut out the macro and just `display` the information in question:

```
. display "`: type make´"
str18
```

This kind of shortcut is also documented in [P] **macro**. Had the variable type been `str7` or `str42`, the storage type would have been reported accordingly. We have thus a criterion for a string variable: that the first three characters of the type would be `str`. A check of possible storage types at (say) `help data types` shows that the numeric types are `byte`, `int`, `float`, `long`, and `double`, so there is no possibility of confusion if we use this criterion. Given the `substr()` function for extracting substrings, some code for this problem is

```
foreach v of var * {
        local t : type `v´
        if substr("`t´", 1, 3) == "str" {
             local strvarlist `strvarlist´ `v´
        }
}
```

So within the loop, we look up the type for each variable. If the first three characters are `str`, we add the name of the variable in question to our list. The key line,

```
local strvarlist `strvarlist´ `v´
```

defines the local macro `strvarlist` to be whatever it was before the line was entered, together with the new name, from the local macro named `v`.

This code can be refined in two ways. First, the presumption here is that the local macro `strvarlist` does not exist before the loop; otherwise put, just before the loop, `strvarlist` is empty. In careful code, we would ensure that is true by blanking out the macro in advance. Second, using another local macro (here `t`) with the loop is not essential. As in the `display` example earlier in this section, we could code without it.

```
local strvarlist
foreach v of var * {
        if substr("`: type `v´´", 1, 3) == "str" {
                local strvarlist `strvarlist´ `v´
        }
}
```

If we run this code on `auto.dta`, we get the one string variable, `make`.

```
. display "`strvarlist´"
make
```

2.4 How to check properties: confirm and capture

Another commonly used command for this kind of problem is `confirm` (see [P] **confirm**). `confirm` appears to offer a more straightforward answer to this particular question. Typing

```
. confirm str var make
```

is answered by silence. Stata's way of confirming what is asserted to be true is to assent tacitly: no news is good news, in plainer terms. If you follow that command with a check on a numeric variable, you get not silence, but an error message.

```
. confirm str var price
´price´ found where string variable expected
r(7);
```

So `confirm` is like a fire or burglar alarm: it goes off if something is wrong. That alarm would stop any loop over variables as soon as an error was first encountered. But often we want to keep on going, which is where `capture` (see [P] **capture**) enters the story.

```
. capture confirm str var price
. display _rc
7
```

These commands eat the error message, while the nonzero return code that is diagnostic of an error remains defined, so we can have it both ways. We just need to check for a zero error code.

```
local strvarlist
foreach v of var * {
        capture confirm str var `v´
        if _rc == 0 {
             local strvarlist `strvarlist´ `v´
        }
}
```

Note that we could go through the loop building two lists (or in other problems, even more lists). For example,

```
local strvarlist
local numvarlist
foreach v of var * {
        capture confirm str var `v´
        if _rc == 0 {
             local strvarlist `strvarlist´ `v´
        }
        else local numvarlist `numvarlist´ `v´
}
```

The decision-making is simple. If the error code is zero—because the variable is a string variable—we add its name to the list of string variables. If the error code is not zero—because the variable is not a string variable—that can only mean that it is a numeric variable, and we add its name to the list of numeric variables. This idea does not presuppose that there are variables of both kinds; if there are not, we will find that out from an empty list.

However, although it is possible to build two or more lists in this manner, that is not often necessary. Later in the column, we will see other ways to get lists through basic set operations such as complementation.

2.5 How to check properties: count and summarize

Two further commonly used commands for checking properties are `count` and `summarize` (see [D] **count** and [R] **summarize**). `summarize` is perhaps more likely to spring to users' minds, but many tasks that might be assigned to `summarize` are more simply tackled by `count` (Cox 2007a,b,d).

Consider the problem of finding variables with any missing values. Various commands, including `nmissing` (Cox 1999, 2001a, 2003a, 2005), offer ways to address this problem. Here we use our loop-and-check technique. The checking is just counting how many values are missing in each variable. If there is at least one, we add the variable to the list.

```
local missvarlist
quietly foreach v of var * {
        count if missing(`v´)
        if r(N) > 0 {
             local missvarlist `missvarlist´ `v´
        }
}
```

The `missing()` function is the best way to check for missings. Among other advantages, it covers both numeric and string variables (Rising 2010), so we do not need to check for numeric or string and then `count if 'v' >= .` or `count if 'v' == ""` accordingly. `count` is noisy by default; that is, the resulting count is displayed. Here `quietly` suppresses the display but the result stored in `r(N)` is inspected. (If `r(N)` is new to you, start by reading `help return`.) `quietly` could appear just on `count` or on a larger segment of code. The choice is not usually important and is typically a matter of convenience. I like my lines of code to remain short, and that preference sometimes drives small decisions on this point.

This little problem could be varied. Suppose the problem were to find variables not with any values missing, but with all values missing. If all values are missing, the count result `r(N)` will be equal to the number of observations `_N`, so all we need change is one line:

```
local missvarlist
quietly foreach v of var * {
        count if missing(`v´)
        if r(N) == _N {
             local missvarlist `missvarlist´ `v´
        }
}
```

There are yet other ways to do it. For example, you might type
`count if !missing('v')` and then check whether the resulting count was zero.

An approach with `count` will suffice for many basic queries. For example, zero or negative values may make matters difficult or impossible if a variable is necessarily defined as positive, or ideally occurs as positive only, as when, say, a logarithmic transform is being contemplated. More generally, we can count values lying inside or outside de-

fined intervals; the choice between counting "inside" and counting "outside" is usually one of convenience.

Sometimes you do need to reach for `summarize`. Suppose we wanted to find all variables that held proportions, fractions between 0 and 1. Storage type will not help here, because, although we must hold such variables as either `float` or `double`, that is not a criterion. But if we `summarize`, we can inspect minimum and maximum. One tip here is to use the `meanonly` option of `summarize` when possible. The name `meanonly` is misleading, because even under this option other summary statistics are produced (Cox 2007c). With this problem, we have a choice over handling string variables. It is not an error to feed a string variable to `summarize`, so we could just plow straight ahead:

```
local propvarlist
foreach v of var * {
        summarize `v´, meanonly
        if r(min) >= 0 & r(max) <= 1 {
             local propvarlist `propvarlist´ `v´
        }
}
```

After `summarize` with a string variable, a call to the results `r(min)` and `r(max)` yields missings, so no string variable will qualify. Because `summarize, meanonly` displays no results, we can remove the `quietly`, but it would do no harm if we left it.

Hang on, however: The minimum being at least 0 and the maximum being at most 1 includes all indicator or dummy variables coded 0 or 1. We might mind about that inclusion or we might not mind. If we do mind, we could think about adding a check in the code for variables that were only ever 0 or 1 (or missing), but now the code starts getting complicated. Let us pause for a moment and think about a different technique.

2.6 Set operations

Many problems of finding variable names can be phrased in set theory terminology as, for example, the intersection or union of two or more sets, or a subset, complement set, or set difference. So let us define proportions, in a strict sense, as the subset of numeric variables that do have nonmissing values in $(0, 1)$ but are not indicators (nonmissing values only 0 or 1). Let us now approach the problem in steps. First, we get the numeric variables using an approach similar to a previous example:

```
local numvarlist
foreach v of var * {
        capture confirm num var `v´
        if _rc == 0 {
             local numvarlist `numvarlist´ `v´
        }
}
```

Now we find the indicators. Let us say that an indicator is numeric, but all values are 0 or 1 or missing. Naturally, we start with `numvarlist`, not all variables (`*`).

```
local indvarlist
quietly foreach v of var `numvarlist´ {
        count if (`v´ == 0 | `v´ == 1 | missing(`v´))
        if r(N) == _N {
             local indvarlist `indvarlist´ `v´
        }
}
```

Now all we need to do is remove the members of `indvarlist` from `propvarlist` obtained earlier. [P] **macro lists** documents how to do such set manipulations. What we want is

```
local propvarlist : list propvarlist - indvarlist
```

We could choose to put all the code together into one loop over variables, and I sometimes do that. But you can end up with a tangle that is hard to maintain or debug or that is legal code but hides logical errors.

3 The findname command

3.1 Purpose of findname

At the time of writing, the official command that comes closest in spirit to the approach discussed so far is `ds`. For some purposes, `lookfor` (see [D] **lookfor**) also offers a quick and easy solution. However, neither `ds` nor `lookfor` nor any other command can offer canned one-line solutions to all the little problems above, which is precisely why knowing the approach from first principles is valuable.

`ds` and its relatives under differing names have had a long and complicated history in various official and user-written versions (for example, Anonymous [1992]; Cox [2000, 2001b]; Weiss [2008]). Although `ds` was for a while undocumented—meaning precisely, documented through a help file but not a manual entry—`ds` was restored to full status in the 9 March 2010 update to Stata 11. Be that as it may, this column offers yet another variation, `findname`, with extended functionality and a revised syntax.

`findname` lists variable names of the dataset currently in memory in a compact or detailed format and lets you specify subsets of variables to be listed, either by name or by properties (for example, the variables are numeric). In addition, `findname` leaves behind in `r(varlist)` the names of variables selected so that you can use them in a subsequent command.

If two or more options specifying properties of variables are specified, `findname` identifies only those variables that satisfy all the option specifications, that is, the intersection of all the subsets identified. Its `not` option provides a direct way to identify the complementary set. Two or more calls to `findname` with results saved in local macros using its `local()` option may be used together with macro operations to produce the union, set difference, etc., of different subsets.

3.2 Examples of findname

That is the executive summary, but now let us take it more slowly through examples.

`findname` can find all string or numeric variables through a call to its `type()` option:

```
. findname, type(string)
. edit `r(varlist)´
. findname, type(numeric)
. summarize `r(varlist)´
```

Note that the variable names not only are displayed but also are saved in `r(varlist)` so that we can use that, or more precisely its local macro persona, in a subsequent command. But watch out: `r()` results are ephemeral and often quickly overwritten. For example, the `summarize` call above destroys `r(varlist)` and leaves its own `r()` results behind instead. To avoid the annoying problem of losing the results you only just obtained, an easy alternative is to copy results immediately to a local macro with a name of your own choice:

```
. findname, type(string)
. local strvarlist `r(varlist)´
```

An even easier alternative is to call `findname`'s `local()` option:

```
. findname, type(string) local(strvarlist)
. edit `strvarlist´
```

The difference is that the local macro `strvarlist` will remain visible within the same session, program, or do-file, unless you yourself overwrite it. If it is really important, save the contents using `notes` (see [D] **notes**). A `local()` option is not included in `ds`.

To find proportion variables, we could type

```
. findname, all((@ >= 0 & @ <= 1) | missing(@)) local(propvarlist)
. findname, all(@ == 0 | @ == 1 | missing(@)), local(indvarlist)
. local propvarlist : list propvarlist - indvarlist
```

Options `all()` and `any()` are new to `findname` over `ds`. A call to `all()` or `any()` features `@` as a placeholder for variables, so each variable name is substituted in turn for `@`.

There is a twist in the first command just given. When we looped for ourselves, we could rely on the fact that `summarize` just ignores any missings. But `findname` does not do that. If it did, that would seem reasonable behavior up to the point when you wanted to find missings, when it would start to seem highly unreasonable. So in careful code, we need to be explicit about the possibility of missings existing in a variable.

(*Continued on next page*)

A `placeholder()` option is provided for some unlikely but not impossible situations. Your `@` key might be broken. Or you might need a string comparison based on the literal character `@`, such as when you want to look for variables containing that character, perhaps variables containing email addresses. `placeholder()` lets you specify an alternative, although in turn you need to choose something that you do not also need to use literally.

What is likely to be more important is that `findname` quietly traps all tests in `all()` or `any()` calls that result in a type mismatch. In this example, the implied

```
. count if varname >= 0 & varname <= 1
```

would fail as illegal if *varname* was a string variable, because double quotes would be needed around 0 and 1. However, `findname` is smart enough to catch such mismatches. (The smartness here is more a matter of brute force, but it works anyway.)

The `local()` option does come in especially handy in this example. Not only do the macrolist directives require macro names, but also the second call to `findname` overwrites the `r(varlist)` left in memory by the first.

Here is another way to approach this example. Again the results of one `findname` are fed to another, but this time the `not` option of `findname` is used to find the complement of the set of indicators within the larger set:

```
. findname, all((@ >= 0 & @ <= 1) | missing(@)) local(propvarlist)
. findname `propvarlist´, all(@ == 0 | @ == 1 | missing(@)) not
```

If logarithms were being contemplated, then

```
. findname, any(@ <= 0)
```

finds any numeric variables with zero or negative values.

A new example with `findname` is finding variables with only integer values. Storage type is again no criterion here, because nothing stops you holding integers in variables of `float` or `double` type. A suitable criterion is that no values are changed by rounding to the nearest integer, say,

```
. findname, all(@ == int(@))
```

Again any string variables are just ignored in a call like this. The `int()` function is used here, which always rounds toward zero such that `int(-1.2)` is -1, but the `floor()` or `ceil()` function would do just as well (Cox 2003b).

As a final example, which variables are constants, holding precisely the same value in all observations? Unless such variables had been constructed deliberately, say, for some graphical purpose, they would normally appear as candidates for `drop`ping. Here is a `findname` solution:

```
. findname, all(@ == @[1])
```

If all values are the same, then each value is the same as the first value, a criterion that applies to numeric and string variables alike. The first observation is not special in this problem (unless your dataset has only one observation, and if that were so, you would not be asking this question). If there were at least 7 observations, or at least 42, then the criteria `@ == @[7]` or `@ == @[42]` would work as well. And even without those restrictions, `@ == @[_N]` would always work. But the first solution given above is the simplest solution guaranteed to work.

For comparison, here is a loop-and-check solution:

```
local constvarlist
foreach v of var * {
        sort `v´
        if `v´[1] == `v´[_N] {
             local constvarlist `constvarlist´ `v´
        }
}
```

We must `sort` each variable to be sure of picking up any departures from constancy, because otherwise the first and last values might just be equal as a coincidence. To put it more positively, even if there are only two distinct values and one of those occurs just once, `sort`ing guarantees that we will spot that case, because the unique oddity will be sorted to one or the other end of the data.

More statistically minded solutions to this question are not so straightforward. You might think in terms of finding the minimum and maximum of each variable and checking whether they are the same. However, `summarize` ignores missing values; that is no solution for strings; and even if there were no missing values and no string variables, you still have to scan the output of `summarize` somehow.

3.3 Further features of findname

`findname` has a bundle of options for searching not only according to storage types, but also according to formats, label and characteristic names, and included text. Its syntax now distinguishes more clearly than `ds` between (for example) finding out whether value labels are attached, what they are named, and what is included in their text. The ability to search value-label text and characteristic text as well as names is new over `ds`.

Combination of criteria for variables is also extended over `ds`. You can combine options, searching for those variables with all of two or more criteria satisfied. The logic of `findname` is that of finding the intersection of sets. As we have seen, unions and set differences need two or more calls to `findname`, although the `not` option offers scope to get the complement of any specification.

(*Continued on next page*)

4 Syntax for findname

`findname` [*varlist*] [*if*] [*in*] [`,` `insensitive` `local(`*macname*`)` `not`
`placeholder(`*symbol*`)` `alpha` `detail` `indent(`#`)` `skip(`#`)` `varwidth(`#`)`
`type(`*typelist*`)` `all(`*condition*`)` `any(`*condition*`)` `format(`*patternlist*`)` `varlabel`
`varlabeltext(`*patternlist*`)` `vallabel` `vallabelname(`*patternlist*`)`
`vallabeltext(`*patternlist*`)` `char` `charname(`*patternlist*`)` `chartext(`*patternlist*`)`]

4.1 Definitions for options

typelist used in `type(`*typelist*`)` is a list of one or more types, each of which may be `numeric`, `string`, `byte`, `int`, `long`, `float`, or `double`, or may be a numlist, such as `1/8` to mean `str1` `str2` ... `str8`. Examples include

`type(int)`	is of type `int`
`type(byte int long)`	is of integer `type`
`type(numeric)`	is a numeric variable
`type(1/40)`	is `str1`, `str2`, ..., `str40`
`type(numeric 1/2)`	is numeric or `str1` or `str2`

patternlist used in, for example, `format(`*patternlist*`)`, is a list of one or more patterns. A pattern is the expected name or text with the likely addition of the characters `*` and `?`. `*` indicates 0 or more characters go here and `?` indicates exactly 1 character goes here. Examples include

`format(*f)`	format is `%`#`.`#`f`
`format(%t*)`	has time or date format
`format(%-*s)`	is a left-justified string
`varl(*weight*)`	variable label includes word `weight`
`varl(*weight* *Weight*)`	variable label includes word `weight` or `Weight`

To match a phrase, it is important to enclose the entire phrase in quotes.

`varl("*some phrase*")`	variable label has `some phrase`

If instead you used `varl(*some phrase*)`, then only variables having labels ending in `some` or starting with `phrase` would be listed.

condition used in `all()` or `any()` is a true-or-false condition defined by an expression in which variable names are represented by `@`. For example, `any(@ < 0)` selects numeric variables in which any values are negative.

4.2 Options

Control options

`insensitive` specifies that the matching of any pattern in *patternlist* be case-insensitive. For example, `varl(*weight*) inse` is an alternative to, and more inclusive than, `varl(*weight* *Weight*)`.

`local(`*macname*`)` puts the resulting list of variable names into local macro *macname*.

`not` specifies that *varlist* or the specifications given define the set of variables not to be listed. For instance, `findname pop*, not` specifies that all variables not starting with the letters `pop` be listed. The default is to list all the variables in the dataset or, if *varlist* or particular properties are specified, to list the variable names so defined.

`placeholder(`*symbol*`)` specifies an alternative to `@` to use in the `any()` or `all()` option. This should only be necessary for making string comparisons involving `@` as a literal character (or if your `@` key is somehow unavailable).

Display options

`alpha` specifies that the variable names be listed in alphabetical order.

`detail` specifies that detailed output identical to that of [D] **describe** be produced. If `detail` is specified, `indent()`, `skip()`, and `varwidth()` are ignored.

`indent(#)` specifies the amount the lines are indented.

`skip(#)` specifies the number of spaces between variable names; the default is `skip(2)`.

`varwidth(#)` specifies the display width of the variable names; the default is `varwidth(12)`.

Selection by data types, values, and formats

`type(`*typelist*`)` selects variables of the specified *typelist*. Typing `findname, type(string)` would list all the names of string variables in the dataset, and typing `findname pop*, type(string)` would list all the names of string variables beginning with the letters `pop`.

`all(`*condition*`)` selects variables that have all values satisfying *condition*. If either `if` or `in` is specified, attention is restricted to the observations specified.

`any(`*condition*`)` selects variables that have any values satisfying *condition*. If either `if` or `in` is specified, attention is restricted to the observations specified.

With either `all()` or `any()`, *condition*s that mismatch type are ignored.

`format(`*patternlist*`)` selects variables whose format matches any of the patterns in *patternlist*. `format(*f)` would select all variables with formats ending in `f`, which presumably would be all `%`#`.`#`f`, `%0`#`.`#`f`, and `%-`#`.`#`f` formats. `format(*f *fc)` would select all formats ending in `f` or `fc`.

Selection by variable and value labels

`varlabel` selects variables with defined variable labels.

`varlabeltext(`*patternlist*`)` selects variables with variable-label text matching any of the words or phrases in *patternlist*.

`vallabel` selects variables with defined value labels.

`vallabelname(`*patternlist*`)` selects variables with value-label names matching any of the words in *patternlist*.

`vallabeltext(`*patternlist*`)` selects variables with value-label text matching any of the words or phrases in *patternlist*. If either `if` or `in` is specified, attention is restricted to the observations specified. Either way, no attention is paid to value labels that do not correspond to values present in the data.

Selection by characteristics

`char` selects variables with defined characteristics. Notes in the sense of [D] **notes** are characteristics.

`charname(`*patternlist*`)` selects variables with characteristic names matching any of the words in *patternlist*.

`chartext(`*patternlist*`)` selects variables with characteristic text matching any of the words or phrases in *patternlist*.

5 Conclusions

Finding variable names within Stata datasets is a basic task that has increased in importance as the size of datasets allowed has increased. As always in data management, users need versatile basic commands for common versions of the problem and ways of going beyond those commands when they do not offer a solution. `findname` is offered here as an alternative to `ds`. The strategies and tricks in this column for looping and checking are offered as an alternative to both.

6 Acknowledgments

My earlier work on versions of `ds` was aided by suggestions from Richard Goldstein, William Gould, Jay Kaufman, and Fred Wolfe. More recently, Maarten Buis, Martin Weiss, Vince Wiggins, and several members of Statalist provided helpful comments both directly and indirectly that led to the development of `findname`.

7 References

Anonymous. 1992. dm67.1: Short describes, finding variables, and codebooks. *Stata Technical Bulletin* 8: 3–5. Reprinted in *Stata Technical Bulletin Reprints*, vol. 2, pp. 11–14. College Station, TX: Stata Press.

Cox, N. J. 1999. dm67: Numbers of missing and present values. *Stata Technical Bulletin* 49: 7–8. Reprinted in *Stata Technical Bulletin Reprints*, vol. 9, pp. 26–27. College Station, TX: Stata Press.

———. 2000. dm78: Describing variables in memory. *Stata Technical Bulletin* 56: 2–4. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 15–17. College Station, TX: Stata Press.

———. 2001a. dm67.1: Enhancements to numbers of missing and present values. *Stata Technical Bulletin* 60: 2–3. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 7–9. College Station, TX: Stata Press.

———. 2001b. dm78.1: Describing variables in memory: update to Stata 7. *Stata Technical Bulletin* 60: 3. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, p. 17. College Station, TX: Stata Press.

———. 2002. Speaking Stata: How to face lists with fortitude. *Stata Journal* 2: 202–222.

———. 2003a. Software update for nmissing and npresent. *Stata Journal* 3: 349.

———. 2003b. Stata tip 2: Building with floors and ceilings. *Stata Journal* 3: 446–447.

———. 2005. Software update for nmissing and npresent. *Stata Journal* 5: 607.

———. 2007a. Speaking Stata: Counting groups, especially panels. *Stata Journal* 7: 571–581.

———. 2007b. Speaking Stata: Making it count. *Stata Journal* 7: 117–130.

———. 2007c. Stata tip 50: Efficient use of summarize. *Stata Journal* 7: 438–439.

———. 2007d. Stata tip 51: Events in intervals. *Stata Journal* 7: 440–443.

Rising, B. 2010. Stata tip 86: The missing() function. *Stata Journal* 10: 303–304.

Weiss, M. 2008. Stata tip 66: ds—A hidden gem. *Stata Journal* 8: 448–449.

About the author

Nicholas Cox is a statistically minded geographer at Durham University. He contributes talks, postings, FAQs, and programs to the Stata user community. He has also coauthored 15 commands in official Stata. He wrote several inserts in the *Stata Technical Bulletin* and is an editor of the *Stata Journal*.

The Stata Journal (2010)
10, Number 2, pp. 297–302

Review of Multivariable Model-building: A Pragmatic Approach to Regression Analysis Based on Fractional Polynomials for Modeling Continuous Variables, by Royston and Sauerbrei

William D. Dupont
Department of Biostatistics
Vanderbilt University School of Medicine
Nashville, TN
william.dupont@vanderbilt.edu

Abstract. This article reviews *Multivariable Model-building: A Pragmatic Approach to Regression Analysis Based on Fractional Polynomials for Modeling Continuous Variables*, by Patrick Royston and Willi Sauerbrei.

Keywords: gn0050, applied statistics, nonlinear regression, fractional polynomials

1 Introduction

Royston and Sauerbrei (2008) provide an excellent introduction to building nonlinear regression models. Their preferred approach is to use fractional polynomial models, a technique that they have largely developed and on which they are undisputed authorities (Royston and Altman 1994; Sauerbrei and Royston 1999). This technique is applicable to models in which the response variable is continuous or dichotomous, to survival models, and to any generalized linear model.

The text is full of practical examples that Royston and Sauerbrei use to illustrate their model-building approach. They make extensive use of smoothed residual plots to evaluate their models and to guide model selection. Their approach is suitable for epidemiological studies in which the number of observations is at least an order of magnitude larger than the number of model covariates. For such data, the problems of multiple comparisons and the overfitting of models are not an overwhelming concern. The problem of model-building for genomic or other studies in which the number of covariates greatly exceeds the number of study subjects is not addressed in the text.

Fractional polynomial models are a subset of generalized linear models in which various powers of the covariates of interest are entered into the linear predictor. A fractional polynomial regression model of order 1 (FP1) is one in which the linear predictor takes the form

$$\beta_0 + \beta_1 x^p$$

where p takes one of the values in $S = \{-2,\ -1,\ -0.5,\ 0,\ 0.5,\ 1,\ 2,\ 3\}$ and $x > 0$. For example, the linear regression model

$$y = \beta_0 + \beta_1/\sqrt{x}$$

is an example of an FP1 model, as is the logistic regression model

$$\text{logit}(\pi[x]) = \beta_0 + \beta_1 x^2$$

A fractional polynomial model of order 2 (FP2) is one in which the linear predictor takes the form

$$\beta_0 + \beta_1 x^{p_1} + \beta_2 x^{p_2}$$

or

$$\beta_0 + \beta_1 x^{p} + \beta_2 x^{p} \log x$$

for p_1, p_2, and p in S and $x > 0$. This definition generalizes in a natural way: in an mth order model (FPm), the linear predictor equals

$$\beta_0 + \sum_{j=1}^{m} \beta_j h_j\,(x)$$

where

$$h_j\,(x) = \begin{cases} x^{p_j}, & p_j \neq p_{j-1} \\ h_{j-1}\,(x) \log x, & p_j = p_{j-1} \end{cases}$$

and $p_1, p_2, \ldots, p_m$ are chosen from S.

The fact that these models all involve linear combinations of the model parameters means that the extensive theory of generalized linear models applies and that many sophisticated programs for building and evaluating such models are already available.

2 Contents

Royston and Sauerbrei's text (2008) starts with an introductory chapter that explains the need for robust ways to fit nonlinear regression models. The authors consider polynomial regression models and find them wanting for many situations. Then they introduce fractional polynomial models and illustrate the use of smoothed residual plots to evaluate model fit. They recommend using the simplest model that fits the data well. Royston and Sauerbrei also illustrate the application of the fractional polynomial approach to Cox proportional hazards regression analysis. They contrast models using age at entry as a continuous covariate with two models with categorical age intervals and with FP1 and FP2 models. They make a convincing argument in favor of the optimal FP2 model over these other alternatives.

Royston and Sauerbrei briefly discuss other modeling approaches, and they make an important distinction between global influence models, such as fractional polynomial models, and local influence models, such as those using restricted cubic splines. They

discuss different types of residuals along with those residuals' roles in guiding model selection.

Chapter 2 introduces the authors' approach to multivariable model selection. Their overall preference is to use backward elimination based on repeated significance tests. They stress the importance of assessing the stability of the selected model by bootstrapping. They distinguish between the modeling goals of prediction and explanation. They place considerable emphasis on reducing the often daunting complexity of observational data and finding comparatively simple models that identify prognostically and diagnostically important variables. Royston and Sauerbrei discuss the pros and cons of different stepwise approaches to model selection, and they describe Akaike's and the Bayesian information criteria (AIC and BIC, respectively). They also discuss shrinkage methods to reduce the effects of selection bias.

Chapter 3 details how to handle categorical and continuous predictors. It contains good advice that is applicable to any exploratory multivariable regression analysis. The multiple-comparisons problems associated with choosing an optimal cutpoint for a continuous covariate are nicely illustrated. Royston and Sauerbrei provide an interesting discussion of the pros and cons of local-influence models, such as lowess regression or cubic splines, and global models, such as those using fractional polynomials.

Chapters 4 and 5 describe in detail the use of fractional polynomials for one variable. Royston and Sauerbrei give the shapes of FP1 and FP2 curves along with their justification of the set of powers, S, that they consider. They explain both naïve and bootstrapped confidence intervals for the linear predictor; the latter adjust for overfitting because of the model-selection algorithm. They discuss methods of graphical and tabular presentation of results from fractional polynomial models along with a worked example on the relationship between systolic blood pressure and all-cause mortality. The authors also describe transformations to improve the robustness of fractional polynomial models.

Chapter 6 introduces multivariable model-building with fractional polynomials—what Royston and Sauerbrei describe as the heart of their text. They present their algorithm for selecting multivariable models, which they illustrate with an example. In essence, they use backward elimination while allowing significant covariates to be fit with an optimal fractional polynomial model. They use functional plots to describe the effect of a covariate on the response variable adjusted for other variables in the model. They consider graphical analysis of residuals from multivariable models. And they use an R^2-like statistic to evaluate the contribution of individual variables.

Chapter 7 describes adding interaction terms to multivariable fractional polynomial models. Royston and Sauerbrei urge the use of graphical checks, sensitivity, and stability analyses as well as a cautious interpretation of the results of such models. Chapter 8 describes the use of bootstrap analyses to assess the stability of complex models.

Chapter 9 compares multivariable fractional polynomial models with spline models. Restricted cubic spline models are a major alternative to fractional polynomial models. They require the specification of three or more knots and fit curves that are cubic

polynomials between adjacent knots, are straight lines before the first and after the last knot, and have the property that the splines and their first and second derivatives are continuous at all knots. This latter property makes restricted cubic splines fairly insensitive to the precise location of their knots. Royston and Sauerbrei present an algorithm for fitting multivariate models with restricted cubic splines. They analyze several datasets with this algorithm and find that it gives models that are roughly comparable with those obtained using multivariable fractional polynomial models.

Chapter 10 provides further guidance on fitting multivariable fractional polynomial models. Chapter 11 describes hazard regression models with time-varying hazard ratios and other topics. Chapter 12, an epilogue, summarizes Royston and Sauerbrei's major recommendations as to how to build useful multivariable models with fractional polynomials.

3 Strengths and weaknesses

This book's greatest strength is its lucid writing style and practical guidance on how to build complex multivariable models. It contains many examples with models that are extensively evaluated using smoothed residual plots and partial predictor plots. I was also intrigued by Royston and Sauerbrei's approach to automated model fitting. Many statisticians have a negative attitude toward such methods because of the risk of overfitting and because of multiple-comparisons concerns (Harrell 2001). I found Royston and Sauerbrei's emphasis on bootstrap analyses to assess model stability to be reassuring. Software to implement their methods is available in Stata, either as part of Stata 11 or as user-contributed programs that can be downloaded over the Internet.

Weaknesses are few. Given that restricted cubic splines are, perhaps, the major competitor to fractional polynomial models, I would have preferred that the authors give a more thorough evaluation of this approach, particularly in regard to fitting univariate models.

4 Fractional polynomials versus restricted cubic splines

I must start this section with a disclaimer: I have been an advocate of restricted cubic splines for several years (Dupont 2009), while my knowledge of fractional polynomial models was limited prior to reading this book. This perhaps gives me a bias in favor of the former over the latter technique.

An argument that the authors make in favor of fractional polynomial models is their simplicity. FP1 models are simpler than restricted cubic spline models, and for this reason, I would recommend an FP1 model whenever it fits the data well. For FP2 models, I am not really sure. My best guess is that most statisticians, and virtually all medical scientists, will lack a visual image of what, say, a linear predictor of the form $\beta_0 + \beta_1 x^2 + \beta_2/\sqrt{x}$ looks like without drawing it. A restricted cubic spline with three knots takes the form $\beta_0 + \beta_1 x + \beta_2 f_2(x)$. Now I must admit the function $f_2(x)$ is neither

pretty nor edifying, but it is readily calculated by any computer. Just as some sausages are best appreciated by eating them without too much knowledge of their contents, a restricted cubic spline can be best understood by drawing it, without paying too much attention to the form of $f_2(x)$. Thus I would argue that FP2 models and three-knot restricted cubic spline models have similar complexity. For data that can be fit well by an FP2 model, my sense is that the two approaches give roughly comparable results. For more complicated data, I believe that restricted cubic splines may have an edge. In particular, my experiments fitting restricted cubic splines to bimodal data have worked well, while fractional polynomial models have either missed the bimodal nature of the data or have provided poor fits to very high or very low values of x.

Another advantage of restricted cubic splines is that the linear model is always nested within more complex models. This means that it is always possible to conduct a Wald or likelihood-ratio test of whether the linear predictor is, in fact, linear in x.

5 Conclusions

This is a very well-written book that provides a thoughtful approach to fitting nonlinear models. The authors are very experienced biostatisticians who have worked extensively in observational and experimental medical science. They make a convincing argument that fractional polynomials can be a valuable tool for building such models in many situations. Their many examples and excellent illustrations make their book accessible to a broad audience within statistical science. Their software will make this book particularly useful to the Stata community. I highly recommend this text.

6 References

Dupont, W. D. 2009. *Statistical Modeling for Biomedical Researchers: A Simple Introduction to the Analysis of Complex Data*. 2nd ed. Cambridge: Cambridge University Press.

Harrell Jr., F. E. 2001. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis*. New York: Springer.

Royston, P., and D. G. Altman. 1994. Regression using fractional polynomials of continuous covariates: Parsimonious parametric modelling (with discussion). *Applied Statistics* 43: 429–467.

Royston, P., and W. Sauerbrei. 2008. *Multivariable Model-building: A Pragmatic Approach to Regression Analysis Based on Fractional Polynomials for Modelling Continuous Variables*. Chichester, UK: Wiley.

Sauerbrei, W., and P. Royston. 1999. Building multivariable prognostic and diagnostic models: Transformation of the predictors by using fractional polynomials. *Journal of the Royal Statistical Society, Series A* 162: 71–94.

About the author

William D. Dupont is a professor of biostatistics and preventive medicine at Vanderbilt University School of Medicine. His interests include the epidemiology of benign breast disease, power and sample-size calculations, statistical graphics, and teaching intermediate-level biostatistics to physician scientists. He is the author of *Statistical Modeling for Biomedical Researchers* (Cambridge University Press, 2009), which uses Stata to teach biostatistics to this audience.

The Stata Journal (2010)
10, Number 2, pp. 303–304

Stata tip 86: The missing() function

Bill Rising
StataCorp
College Station, TX
brising@stata.com

Stata's treatment of missing numeric values in expressions is clear: numeric values behave like infinity. However, some care is needed whenever numeric missing values could appear in a relational expression. Typically, missing values are included or excluded explicitly by a segment of Stata code. But there is a better way to deal with missing values: the `missing()` function.

Suppose that we have a dataset containing the body mass index (BMI) and age for a sample of people. As happens in real-world datasets, some observations contain missing values for one or both of the variables. We would like to create an indicator variable that marks all the obese adults, meaning those who are at least 20 years old and have a BMI of at least 30. On the surface, the command we should type is

```
. generate obese_adult = age>=20 & bmi>=30
```

This will not yield the desired results, however, because the relational operators do not treat missing values as anything special—they are just considered to be very large numbers and so are still included. Hence, the values for `obese_adult` will be filled according to the following table:

	`age<20`	`age>=20`, nonmissing	`age` missing
`bmi<30`	0	0	0
`bmi>=30`, nonmissing	0	1	1
`bmi` missing	0	1	1

The solution to this problem is to assign the value only for those observations that have no missing values. Because missing values are large, we could type

```
. generate obese_adult = age>=20 & bmi>=30 if age<. & bmi<.
```

Although correct, this is not the best solution. In particular, it is only readable to Stata users—other people would have no clue what `age<.` means.

A better way to work with missing values is to use the `missing()` function. Its syntax and behavior are simple: `missing(`*exp1*[`,`*exp2*`,...,`*expN*]`)` evaluates to `1` if any of the expressions is missing and `0` if none is missing. Thus we could rewrite our previous correct `generate` command as

```
. generate obese_adult = age>=20 & bmi>=30 if !missing(age,bmi)
```

This is much more readable.

Another advantage of using the `missing()` function is that it treats string and numeric variables in the same fashion. Now suppose that we have a dataset containing

a string-valued identifier named `ssn` and a numeric identifier named `whodat`. If we want to mark all the observations for which at least one identifier is missing, we can do this quite easily:

```
. generate byte badid = missing(ssn,whodat)
```

Not only is this readable but it also does not require knowing the data type of the two variables.

One thing that can trip up users new to `missing()` is its arguments: they are a comma-separated list of expressions, not a varlist. If you are interested in using varlists, you can write your own ado-file:

```
*! version 1.0.0 February 18, 2010 @ 08:48:04
*! makes a comma-separated varlist
program define vcomma, rclass
version 11
        syntax [varlist]
   local varlist : subinstr local varlist " " ",", all
   return local varlist "`varlist´"
end
```

You can then give `vcomma` a varlist, and it will return the comma-separated list in `r(varlist)`. So, for example, if you were interested in finding any observation in a dataset that had missing values for any variable, you could do this with the following two lines:

```
vcomma
list if missing(r(varlist))
```

This is certainly much easier than using loops or a long `if` qualifier.

Working with missing values is necessary in most datasets. As you have learned in this tip, using the `missing()` function makes working with missing values less onerous.

The Stata Journal (2010)
10, Number 2, pp. 305–308

Stata tip 87: Interpretation of interactions in nonlinear models

Maarten L. Buis
Department of Sociology
Tübingen University
Tübingen, Germany
maarten.buis@uni-tuebingen.de

When fitting a nonlinear model such as `logit` (see [R] **logit**) or `poisson` (see [R] **poisson**), we often have two options when it comes to interpreting the regression coefficients: compute some form of marginal effect or exponentiate the coefficients, which will give us an odds ratio or incidence-rate ratio. The marginal effect is an approximation of how much the dependent variable is expected to increase or decrease for a unit change in an explanatory variable; that is, the effect is presented on an additive scale. The exponentiated coefficients give the ratio by which the dependent variable changes for a unit change in an explanatory variable; that is, the effect is presented on a multiplicative scale. An extensive overview is given by Long and Freese (2006).

Sometimes, we are also interested in how the effect of one variable changes when another variable changes, called the interaction effect. Because there is more than one way in which we can define an effect in a nonlinear model, there must also be more than one way in which we can define an interaction effect. This tip deals with how to interpret these interaction effects when we want to present effects as odds ratios or incidence-rate ratios, which can be an attractive alternative to interpreting interactions effects in terms of marginal effects.

The motivation for this tip is many recent discussions on how to interpret interaction effects when we want to interpret them in terms of marginal effects (Ai and Norton 2003; Norton, Wang, and Ai 2004; Cornelißen and Sonderhof 2009). (A separate concern about interaction effects in nonlinear models that is often mentioned is the possible influence of unobserved heterogeneity on these estimates; for example, see Williams [2009]. But I will not deal with that potential problem here.) These authors point out a common mistake, interpreting the first derivative of the multiplicative term between two explanatory variables as the interaction effect. The problem with this is that we want the interaction effect between two variables (x_1 and x_2) to represent how much the effect of x_1 changes for a unit change in x_2. The effect of x_1, in the marginal effects metric, is the first derivative of the expected value of the dependent variable ($E[y]$) with respect to x_1, which is an approximation of how much $E[y]$ changes for a unit change in x_1. The interaction effect should thus be the cross partial derivative of $E[y]$ with respect to x_1 and x_2—that is, an approximation of how much the derivative of $E[y]$ with respect to x_1 changes for a unit change in x_2. In nonlinear models, this is typically different from the first derivative of $E[y]$ with respect to the multiplicative term $x_1 \times x_2$. This is where programs like `inteff` by Norton, Wang, and Ai (2004) and `inteff3` by Cornelißen and Sonderhof (2009) come in.

Fortunately, we can interpret interactions without referring to any additional program by presenting effects as multiplicative effects (for example, odds ratios, incidence-rate ratios, hazard ratios). However, the marginal effects and the multiplicative effects answer subtly different questions, and thus it is a good idea to have both tools in your toolbox.

The interpretation of results is best explained using an example. Here we study whether the effect of having a college degree (`collgrad`) on the odds of obtaining a "high" job (`high_occ`) differs between black and white women.

```
. sysuse nlsw88
(NLSW, 1988 extract)
. generate byte high_occ = occupation < 3 if occupation < .
(9 missing values generated)
. generate byte black = race == 2 if race < .
. drop if race == 3
(26 observations deleted)
. generate byte baseline = 1
. logit high_occ black##collgrad baseline, or noconstant nolog
Logistic regression                             Number of obs   =       2211
                                                Wald chi2(4)    =     504.62
Log likelihood = -1199.4399                     Prob > chi2     =     0.0000
```

high_occ	Odds Ratio	Std. Err.	z	P>|z|	[95% Conf.	Interval]
1.black	.4194072	.0655069	-5.56	0.000	.3088072	.5696188
1.collgrad	2.465411	.293568	7.58	0.000	1.952238	3.113478
black# collgrad 1 1	1.479715	.4132536	1.40	0.161	.8559637	2.558003
baseline	.3220524	.0215596	-16.93	0.000	.2824512	.3672059

If we were to interpret these results in terms of marginal effects, we would typically look at the effect of the explanatory variables on the probability of attaining a high job. However, this example uses a `logit` model together with the `or` option, so the dependent variable is measured in the odds metric rather than in the probability metric. Odds have a bad reputation for being hard to understand, but they are just the expected number of people with a high job for every person with a low job. For example, the baseline odds—the odds of having a high job for white women without a college degree—is 0.32, meaning that within this category, we expect to find 0.32 women with a high job for every woman with a low job. The trick I have used to display the baseline odds is discussed in an earlier tip (Newson 2003). The odds ratio for `collgrad` is 2.47, which means that the odds of having a high job is 2.47 times higher for women with a college degree. There is also an interaction effect between `collgrad` and `black`, so this effect of having a college degree refers to white women. The effect of college degree for black women is 1.48 times that for white women. So the interaction effect tells how much the effect of `collgrad` differs between black and white women, but it does so in multiplicative terms. The results also show that this interaction is not significant.

This example points to the difference between marginal effects and multiplicative effects. Now we can compute the marginal effect as the difference between the expected odds of women with and without a college degree, rather than as the derivative of the expected odds with respect to `collgrad`. The reason for computing the marginal effect as a difference is that `collgrad` is a categorical variable, so this discrete difference corresponds more closely with what would actually be observed. Although it is a slight abuse of terminology, I will continue to call it the marginal effect.

The `margins` command below shows the odds of attaining a high job for every combination of `black` and `collgrad`. The odds of attaining a high job for white women without a college degree is 0.32, while the odds for white women with a college degree is 0.79. The marginal effect of `collgrad` for white women is thus 0.47. The marginal effect of `collgrad` for black women is only 0.36. The marginal effect of `collgrad` is thus larger for white women than for black women, while the multiplicative effect of `collgrad` is larger for black women than for white women.

```
. margins, over(black collgrad) expression(exp(xb())) post
Predictive margins                                Number of obs   =       2211
Model VCE    : OIM

Expression   : exp(xb())
over         : black collgrad
```

	Margin	Delta-method Std. Err.	z	P>|z|	[95% Conf.	Interval]
black# collgrad						
0 0	.3220524	.0215596	14.94	0.000	.2797964	.3643084
0 1	.7939914	.078188	10.15	0.000	.6407457	.9472371
1 0	.1350711	.0190606	7.09	0.000	.097713	.1724292
1 1	.4927536	.1032487	4.77	0.000	.29039	.6951173

```
. lincom 0.black#1.collgrad - 0.black#0.collgrad
 ( 1)  - 0bn.black#0bn.collgrad + 0bn.black#1.collgrad = 0
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
(1)	.471939	.081106	5.82	0.000	.3129742	.6309038

```
. lincom 1.black#1.collgrad - 1.black#0.collgrad
 ( 1)  - 1.black#0bn.collgrad + 1.black#1.collgrad = 0
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
(1)	.3576825	.1049933	3.41	0.001	.1518994	.5634656

The reason for this difference is that the multiplicative effects are relative to the baseline odds in their own category. In this example, these baseline odds differ substantially between black and white women: for white women without a college degree, we expect to find 0.32 women with a high job for every woman with a low job, while for

black women without a college degree, we expect to find only 0.14 women with a high job for every woman with a low job. So even though the increase in odds as a result of getting a college degree is higher for white women than for black women, this increase as a percentage of the baseline value is less for white women than for black women. The multiplicative effects control in this way for differences between the groups in baseline odds. However, notice that marginal and multiplicative effects are both accurate representations of the effect of a college degree. Which effect one wants to report depends on the substantive question, whether or not one wants to control for differences in the baseline odds.

The example here is relatively simple with only binary variables and no controlling variables. However, the basic argument still holds when using continuous variables and when controlling variables are added. Moreover, the argument is not limited to results obtained from `logit`. It applies to all forms of multiplicative effects, and so, for example, to odds ratios from other models such as `ologit` (see [R] **ologit**) and `glogit` ([R] **glogit**); relative-risk ratios ([R] **mlogit**); incidence-rate ratios (for example, [R] **poisson**, [R] **nbreg**, and [R] **zip**); or hazard ratios (for example, [ST] **streg** and [R] **cloglog**).

Acknowledgment

I thank Richard Williams for helpful comments.

References

Ai, C., and E. C. Norton. 2003. Interaction terms in logit and probit models. *Economics Letters* 80: 123–129.

Cornelißen, T., and K. Sonderhof. 2009. Partial effects in probit and logit models with a triple dummy-variable interaction term. *Stata Journal* 9: 571–583.

Long, J. S., and J. Freese. 2006. *Regression Models for Categorical Dependent Variables Using Stata*. 2nd ed. College Station, TX: Stata Press.

Newson, R. 2003. Stata tip 1: The eform() option of regress. *Stata Journal* 3: 445.

Norton, E. C., H. Wang, and C. Ai. 2004. Computing interaction effects and standard errors in logit and probit models. *Stata Journal* 4: 154–167.

Williams, R. 2009. Using heterogenous choice models to compare logit and probit coefficients across groups. *Sociological Methods & Research* 37: 531–559.

The Stata Journal (2010)
10, Number 2, pp. 309–312

Stata tip 88: Efficiently evaluating elasticities with the margins command

Christopher F. Baum
Department of Economics
Boston College
Chestnut Hill, MA
baum@bc.edu

The new `margins` command, available in Stata 11, greatly simplifies many postestimation computations. Discussions of this command have justifiably highlighted its capabilities to work with factor variables and expressions involving factor-variable operators, such as `c.x#c.x`. But you may find another aspect of `margins` very useful: its ability to compute quantities such as elasticities and semielasticities with a much simpler command syntax than that previously available.

As discussed in [R] **margins**, under the section titled *Expressing derivatives as elasticities*, the elasticity of y with respect to x is the proportional change in y for a proportional change in x. A semielasticity such as `eydx` is the proportional change in y from a unit change in x.

For instance, we might estimate a linear regression equation for per capita expenditures on gasoline in the U.S. market from a time-series dataset:

```
. use http://fmwww.bc.edu/ec-p/data/Greene2008/GasolineMarket
. tsset Year, yearly
        time variable:  Year, 1953 to 2004
                delta:  1 year
. generate gaspc = GasExp/(Gasp*(Pop/1e6))
. generate lngaspc = log(gaspc)
. local allreg Income Gasp PNC PUC PPT PD PN PS
```



```
. regress gaspc `allreg´ Year
      Source |       SS       df       MS              Number of obs =      52
-------------+------------------------------           F(  9,    42) =  530.82
       Model |  56.7083042     9  6.30092268            Prob > F      =  0.0000
    Residual |   .49854905    42  .011870215           R-squared     =  0.9913
-------------+------------------------------           Adj R-squared =  0.9894
       Total |  57.2068532    51    1.121703           Root MSE      =  .10895

------------------------------------------------------------------------------
       gaspc |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      Income |   .0002157   .0000518     4.17   0.000     .0001113    .0003202
        Gasp |  -.0110838   .0039781    -2.79   0.008     -.019112   -.0030557
         PNC |   .0005774   .0128441     0.04   0.964    -.0253432    .0264979
         PUC |  -.0058746   .0048703    -1.21   0.234    -.0157033    .0039541
         PPT |   .0069073   .0048361     1.43   0.161    -.0028524     .016667
          PD |   .0012289   .0118818     0.10   0.918    -.0227495    .0252072
          PN |   .0126905    .012598     1.01   0.320    -.0127333    .0381142
          PS |  -.0280278   .0079962    -3.51   0.001    -.0441649   -.0118907
        Year |   .0725037   .0141828     5.11   0.000     .0438816    .1011257
       _cons |  -140.4213   27.19985    -5.16   0.000    -195.3128    -85.5298
------------------------------------------------------------------------------

. estimates store a
```

Let us say that we wish to calculate elasticities for the year 2004, using the regressors' values in that year. Using pre–Stata 11 syntax, we must store the regression estimates (as above) so that we can use the e-class command `mean` to evaluate the regressors' values in that year and store them in a row vector, `x2004`. We can then restore the regression estimates and use `mfx compute, eyex` to compute the elasticities, with its `at()` option specifying the point in the regressors' space at which they are to be evaluated:

```
. // elasticities at means: compute at t=2004 the old way (with mfx)
. quietly mean `allreg´ Year if Year==2004

. matrix x2004 = e(b)

. estimates restore a
(results a are active now)

. mfx compute, eyex at(x2004)
Elasticities after regress
      y  = Fitted values (predict)
         =  6.1726971
------------------------------------------------------------------------------
variable |      ey/ex   Std. Err.      z    P>|z|   [    95% C.I.   ]      X
---------+--------------------------------------------------------------------
  Income |   .9476599     .2263     4.19   0.000    .504127  1.39119    27113
    Gasp |  -.2224796     .08093   -2.75   0.006   -.381102 -.063857   123.901
     PNC |   .0125245     .2786     0.04   0.964   -.533521   .55857    133.9
     PUC |  -.1268632     .10488   -1.21   0.226   -.332432  .078706    133.3
     PPT |   .2339837     .16441    1.42   0.155    -.08826  .556228    209.1
      PD |   .0228545     .22098    0.10   0.918   -.410256  .455965    114.8
      PN |   .3540265     .35281    1.00   0.316   -.337474  1.04553    172.2
      PS |  -1.011648     .29332   -3.45   0.001   -1.58654 -.436759    222.8
    Year |   23.53872    4.63929    5.07   0.000    14.4459  32.6316     2004
------------------------------------------------------------------------------
```

In contrast, what if we used `margins` to make this computation? We need not store the regression estimates, and one command does the job:

```
. // elasticities at means: compute at t=2004 the new way (with margins)
. margins if Year==2004, eyex(_all) at((means) _all)
Conditional marginal effects                    Number of obs   =          1
Model VCE    : OLS

Expression   : Linear prediction, predict()
ey/ex w.r.t. : Income Gasp PNC PUC PPT PD PN PS Year
at           : Income          =       27113 (mean)
               Gasp            =     123.901 (mean)
               PNC             =       133.9 (mean)
               PUC             =       133.3 (mean)
               PPT             =       209.1 (mean)
               PD              =       114.8 (mean)
               PN              =       172.2 (mean)
               PS              =       222.8 (mean)
               Year            =        2004 (mean)
```

	Delta-method ey/ex	Std. Err.	z	P>|z|	[95% Conf.	Interval]
Income	.9476599	.2262966	4.19	0.000	.5041268	1.391193
Gasp	-.2224796	.0809311	-2.75	0.006	-.3811017	-.0638575
PNC	.0125245	.2786	0.04	0.964	-.5335214	.5585704
PUC	-.1268632	.1048839	-1.21	0.226	-.332432	.0787055
PPT	.2339837	.1644132	1.42	0.155	-.0882602	.5562276
PD	.0228545	.2209787	0.10	0.918	-.4102557	.4559647
PN	.3540265	.3528127	1.00	0.316	-.3374737	1.045527
PS	-1.011648	.2933159	-3.45	0.001	-1.586536	-.4367592
Year	23.53872	4.639292	5.07	0.000	14.44587	32.63156

We merely indicate, with an `if` *exp* clause, that we want marginal effects for the year 2004; that elasticities are to be computed for `_all` regressors; and that they are to be computed at the regressors' means (which are, of course, single values for that year).

We would find it just as straightforward to compute elasticities over the range of regressors' values, for instance, at deciles of their respective distributions. The resulting estimates could be stored in a Stata matrix:

```
. // calculate elasticities at deciles over full sample:
. forvalues i=10(10)90 {
  2.         quietly margins, eyex(_all) at((p`i´) _all)
  3.         local rn "`rn´ p`i´"
  4.         matrix nu = nullmat(nu) \ r(b)
  5. }
. matrix rownames nu = `rn´
```

(*Continued on next page*)

```
. matrix list nu, format(%6.3f) ti("Elasticities")
nu[9,9]:  Elasticities
     Income    Gasp     PNC     PUC     PPT      PD      PN      PS    Year
p10   0.637  -0.063   0.009  -0.045   0.045   0.014   0.124  -0.196  43.934
p20   0.604  -0.057   0.008  -0.045   0.045   0.013   0.112  -0.192  38.315
p30   0.614  -0.051   0.007  -0.040   0.044   0.011   0.104  -0.187  31.412
p40   0.620  -0.052   0.006  -0.039   0.052   0.011   0.115  -0.214  27.265
p50   0.668  -0.098   0.008  -0.063   0.068   0.016   0.172  -0.334  26.614
p60   0.762  -0.149   0.011  -0.119   0.136   0.024   0.243  -0.547  26.836
p70   0.798  -0.149   0.012  -0.122   0.157   0.024   0.264  -0.650  25.338
p80   0.814  -0.150   0.013  -0.136   0.206   0.025   0.301  -0.791  25.029
p90   0.852  -0.150   0.013  -0.144   0.220   0.025   0.310  -0.854  23.397
```

We might also want to hold some regressors' values fixed and vary others across their decile ranges. This can be easily implemented with the `at()` option. For instance,

```
margins, eyex(_all) at((p25) Income (median) Gasp PNC PUC PPT PD PN PS)
```

would compute elasticities at the 25th percentile of `Income` and at the median values of other regressors.

The Stata Journal (2010)
10, Number 2, p. 313

Software Updates

st0173_1: Robust regression in Stata. V. Verardi and C. Croux. *Stata Journal* 9: 439–453.

Previously in the `mcd` command, the former `outlier` option would automatically generate two variables, `MCD_outlier` and `Robust_distance`, respectively flagging the outliers and returning robust distances. This option has been replaced by the `generate(`*newvar1 newvar2*`)` option, which instead allows you to choose the name of these variables. Furthermore, a minor bug in the code related to the use of one variable has been fixed.

st0099_1: Goodness-of-fit test for logistic regression fitted using survey sample data. K. J. Archer, S. Lemeshow, and M. I. Lichter. *Stata Journal* 6: 97–105.

This update of `svylogitgof` corrects some deficiencies in the original. It was assumed that all observations in the dataset were included in the model fit and that the model fit resulted in 10 deciles of risk. Neither assumption necessarily holds in general. For example, observations with missing values are omitted from the model fit. Additionally, when a model is fit where the independent variables are such that there are few distinct covariate patterns, there may be fewer than 10 deciles of risk. `svylogitgof` will now work when no weights are specified. Improved programming practice is followed by using temporary variables and temporary names rather than creating new permanent variables and scalars. After the `svylogitgof` call, the results of the preceding estimation command are restored.

