The Stata Journal

Volume 11 Number 1 2011

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 webpage

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®)
- Scopus™
- 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 listed below include both a printed and an electronic copy unless otherwise mentioned.

Subscriptions mailed to U.S. and Canadian addresses:

1-year subscription	$ 79
2-year subscription	$155
3-year subscription	$225
3-year subscription (electronic only)	$210
1-year student subscription	$ 48
1-year university library subscription	$ 99
2-year university library subscription	$195
3-year university library subscription	$289
1-year institutional subscription	$225
2-year institutional subscription	$445
3-year institutional subscription	$650

Subscriptions mailed to other countries:

1-year subscription	$115
2-year subscription	$225
3-year subscription	$329
3-year subscription (electronic only)	$210
1-year student subscription	$ 79
1-year university library subscription	$135
2-year university library subscription	$265
3-year university library subscription	$395
1-year institutional subscription	$259
2-year institutional subscription	$510
3-year institutional subscription	$750

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 11 Number 1 2011

The Stata Journal

Articles and Columns 1

A procedure to tabulate and plot results after flexible modeling of a quantitative covariate . . . . . N. Orsini and S. Greenland 1

Nonparametric item response theory using Stata . . . . . J.-B. Hardouin, A. Bonnaud-Antignac, and V. Sébille 30

Visualization of social networks in Stata using multidimensional scaling . . . . . R. Corten 52

Pointwise confidence intervals for the covariate-adjusted survivor function in the Cox model . . . . . M. Cefalu 64

Estimation of hurdle models for overdispersed count data . . . . . H. Farbmacher 82

Right-censored Poisson regression model . . . . . R. Raciborski 95

Stata utilities for geocoding and generating travel time and travel distance information . . . . . A. Ozimek and D. Miles 106

eq5d: A command to calculate index values for the EQ-5D quality-of-life instrument . . . . . J. M. Ramos-Goñi and O. Rivero-Arias 120

Speaking Stata: MMXI and all that: Handling Roman numerals within Stata . . . . . N. J. Cox 126

Notes and Comments 143

Stata tip 94: Manipulation of prediction parameters for parametric survival regression models . . . . . T. Boswell and R. G. Gutierrez 143

Stata tip 95: Estimation of error covariances in a linear model . . . . . N. J. Horton 145

Stata tip 96: Cube roots . . . . . N. J. Cox 149

Software Updates 155

The Stata Journal (2011)
11, Number 1, pp. 1–29

A procedure to tabulate and plot results after flexible modeling of a quantitative covariate

Nicola Orsini
Division of Nutritional Epidemiology
National Institute of Environmental Medicine
Karolinska Institutet
Stockholm, Sweden
nicola.orsini@ki.se

Sander Greenland
Departments of Epidemiology and Statistics
University of California–Los Angeles
Los Angeles, CA

Abstract. The use of flexible models for the relationship between a quantitative covariate and the response variable can be limited by the difficulty in interpreting the regression coefficients. In this article, we present a new postestimation command, `xblc`, that facilitates tabular and graphical presentation of these relationships. Cubic splines are given special emphasis. We illustrate the command through several worked examples using data from a large study of Swedish men on the relation between physical activity and the occurrence of lower urinary tract symptoms.

Keywords: st0215, xblc, cubic spline, modeling strategies, logistic regression

1 Introduction

In many studies, it is important to identify, present, and discuss the estimated relationship between a quantitative or continuous covariate (also called predictor, independent, or explanatory variable) and the response variable. In health sciences, the covariate is usually an exposure measurement or a clinical measurement. Regression models are widely used for contrasting responses at different values of the covariate. Their simplest forms assume a linear relationship between the quantitative covariate and some transformation of the response variable. The linearity assumption makes the regression coefficient easy to interpret (constant change of the predicted response per unit change of the covariate), but there is no reason to expect this assumption to hold in most applications.

Modeling nonlinear relationships through categorization of the covariate or adding a quadratic term may have limitations and rely on unrealistic assumptions, leading to distortions in inferences (see Royston, Altman, and Sauerbrei [2006] and Greenland [1995a,c,d]). Flexible alternatives involving more flexible, smooth transformations of the original covariate, such as fractional polynomials and regression splines (linear,

quadratic, or cubic), have been introduced (see Steenland and Deddens [2004]; Royston, Ambler, and Sauerbrei [1999]; Marrie, Dawson, and Garland [2009]; Harrell, Lee, and Pollock [1988]; and Greenland [2008; 1995b]) and are available in Stata (see [R] **mfp** and [R] **mkspline**). Nonetheless, these transformations complicate the contrast of the expected response at different values of the covariate and may discourage their use.

The aim of this article is to introduce the new postestimation command `xblc`, which aids in the interpretation and presentation of a nonlinear relationship in tabular and graphical form. We illustrate the procedure with data from a large cohort of Swedish men. The data examine the relationship between physical activity and the occurrence of lower urinary tract symptoms (LUTS) (Orsini et al. 2006). We focus on cubic-spline logistic regression for predicting the occurrence of a binary response. Nonetheless, the `xblc` command works similarly after any estimation command and regardless of the strategy used to model the quantitative covariate.

The rest of this article is organized as follows: section 2 provides an introduction to different types of cubic splines (not necessarily restricted); section 3 shows how to obtain point and interval estimates of measures of association between the covariate and the response; section 4 describes the syntax of the postestimation command `xblc`; section 5 presents several worked examples showing how to use the `xblc` command after the estimation of different types of cubic-spline models and how to provide intervals for the predicted response rather than differences between predicted responses; and section 6 compares other approaches (categories, linear splines, and fractional polynomials) with model nonlinearity, which can also use the `xblc` command.

2 Cubic splines

Cubic splines are generally defined as piecewise-polynomial line segments whose function values and first and second derivatives agree at the boundaries where they join. The boundaries of these segments are called knots, and the fitted curve is continuous and smooth at the knot boundaries (Smith 1979).

To avoid instability of the fitted curve at the extremes of the covariate, a common strategy is to constrain the curve to be a straight line before the first knot or after the last knot. The `mkspline` command can make both linear and restricted cubic splines since Stata 10.0 (see [R] **mkspline**). In some situations, restricting splines to be linear in both tails is not a warranted assumption. Therefore, we next show how to specify a linear predictor for a quantitative covariate X with neither tail restricted, only the left tail restricted, only the right tail restricted, or both tails restricted.

A common strategy for including a nonlinear effect of a covariate X is to replace it with some function of $X, g(X)$. For example, $g(X)$ could be $b_1X + b_2X^2$ or $b_1 \ln(X)$. For the (unrestricted) cubic-spline model, $g(X)$ is a function of the knot values k_i, $i = 1, \ldots, n$, as follows:

$$g(X) = b_0 + b_1X + b_2X^2 + b_3X^3 + \sum_{i=1}^{n} b_{3+i}\max(X - k_i, 0)^3$$

where the math function $\max(X-k_i, 0)$, known as the "positive part" function $(X-k_i)_+$, returns the maximum value of $X - k_i$ and 0. A model with only the left tail restricted to be linear implies that $b_1 = b_2 = 0$, so we drop X^2 and X^3:

$$g(X) = b_0 + b_1X + \sum_{i=1}^{n} b_{1+i}\max(X - k_i, 0)^3$$

A model with the right tail restricted to be linear is equal to the left-tail restricted model based on $-X$ with knots in reversed order and with the opposite sign of the ones based on the original X, which simplifies to

$$g(X) = b_0 + b_1(-X) + \sum_{i=1}^{n} b_{1+i}\max(k_i - X, 0)^3$$

A model with both tails restricted has $n - 1$ coefficients for transformations of the original exposure variable X,

$$g(X) = b_0 + b_1X_1 + b_2X_2 + \cdots + b_{n-1}X_{n-1}$$

where the first spline term, X_1, is equal to the original exposure variable X, whereas the remaining spline terms, $X_2, \ldots, X_{n-1}$, are functions of the original exposure X, the number of knots, and the spacing between knots, defined as follows:

$$\begin{aligned} u_i =& \max(X - k_i, 0)^3 \quad \text{with} \quad i = 1, \ldots, n \\ X_i =& \{u_{i-1} - u_{n-1}(k_n - k_{i-1})/(k_n - k_{n-1}) + u_n(k_{n-1} - k_{i-1})/(k_n - k_{n-1})\}/ \\ & (k_n - k_1)^2 \\ & \text{with} \quad i = 2, \ldots, n-1 \end{aligned}$$

More detailed descriptions of splines can be found elsewhere (see Greenland [2008]; Smith [1979]; Durrleman and Simon [1989]; Harrell [2001]; and Wegman and Wright [1983]).

3 Measures of association, p-values, and interval estimation

Modeling a quantitative covariate using splines or other flexible tools does not modify the way measures of covariate–response associations are defined.

An estimate of a measure of association between two variables usually ends up being a comparison of the predicted (fitted) value of a response variable (or some function of it) across different groups represented by the covariate. For example, the estimated association between gender and urinary tract symptoms compares the predicted urinary tract symptoms for men with the expected urinary tract symptoms for women. Such a comparison can take the form of computing the difference between the predicted values but can also take the form of computing the ratio.

For a quantitative covariate, such as age in years or pack-years of smoking, there can be a great many groups because each unique value of that covariate represents, in principle, its own group. We can display those using a graph, or we can create a table of a smaller number of comparisons between "representative" groups to summarize the relationship between the variables.

Contrasting predicted responses in the presence of nonlinearity is more elaborate because it involves transformations of the covariate. We illustrate the point using the restricted cubic-spline model; similar considerations apply to other types of covariate transformations. The linear predictor at the covariate values z_1 and z_2 is given by

$$g(X = z_1) = b_0 + b_1 X_1(z_1) + b_2 X_2(z_1) + \cdots + b_{n-1} X_{n-1}(z_1)$$
$$g(X = z_2) = b_0 + b_1 X_1(z_2) + b_2 X_2(z_2) + \cdots + b_{n-1} X_{n-1}(z_2)$$

so that

$$g(X = z_1) - g(X = z_2) =$$
$$b_1 \{X_1(z_1) - X_1(z_2)\} + b_2 \{X_2(z_1) - X_2(z_2)\} + \cdots + b_{n-1} \{X_{n-1}(z_1) - X_{n-1}(z_2)\}$$

The interpretation of the quantity $g(X = z_1) - g(X = z_2)$ depends on the model for the response. For example, within the family of generalized linear models, the quantity $g(X = z_1) - g(X = z_2)$ represents the difference between two mean values of a continuous response in a linear model (see [R] **regress**); the difference between two log odds (the log odds-ratio [OR]) of a binary response in a logistic model (see [R] **logit**); or the difference between two log rates (the log rate-ratio) of a count response in a log-linear Poisson model with the log of time over which the count was observed as an offset variable (see [R] **poisson**).

Commands for calculating p-values and predictions are derived using standard techniques available for simpler parametric models (Harrell, Lee, and Pollock 1988). For example, to obtain the p-value for the null hypothesis that there is no association between the covariate X and the response in a restricted cubic-spline model, we test the joint null hypothesis

$$b_1 = b_2 = \cdots = b_{n-1} = 0$$

The linear-response model is nested within the restricted cubic-spline model ($X_1 = X$), and the linear response to X corresponds to the constraint

$$b_2 = \cdots = b_{n-1} = 0$$

The p-value for this hypothesis is thus a test of linear response. Assuming this constraint, one can drop the spline terms $X_2, \ldots, X_{n-1}$, which simplifies the above comparison to

$$g(X = z_1) - g(X = z_2) = b_1 \{X_1(z_1) - X_1(z_2)\}$$

The quantity $b_1 \{X_1(z_1) - X_1(z_2)\}$ is the contrast between two predicted responses associated with a $z_1 - z_2$ unit increase of the covariate X throughout the covariate range (linear-response assumption). Therefore, modeling the covariate response as linear assumes a constant difference in the linear predictor regardless of where we begin the increase (z_2).

Returning to the general case, an approximate confidence interval (CI) for the difference in the linear predictors at the covariate values z_1 and z_2, $g(X = z_1) - g(X = z_2)$, can be calculated from the standard error (SE) for this difference, which is computable from the covariate values z_1 and z_2 and the covariance matrix of the estimated coefficients:

$$\begin{aligned}&[b_1\{X_1(z_1) - X_1(z_2)\} + b_2\{X_2(z_1) - X_2(z_2)\} + \cdots + b_{n-1}\{X_{n-1}(z_1) - X_{n-1}(z_2)\}] \\ &\pm z_{(\alpha/2)} \times \text{SE}[b_1\{X_1(z_1) - X_1(z_2)\} + b_2\{X_2(z_1) - X_2(z_2)\} + \cdots \\ &\quad + b_{n-1}\{X_{n-1}(z_1) - X_{n-1}(z_2)\}]\end{aligned}$$

where $z_{(\alpha/2)}$ denotes the $100(1 - \alpha/2)$ percentile of a standard normal distribution (1.96 for a 95% CI). The postestimation command `xblc` carries out these computations with the `lincom` command (see [R] **lincom**). In health-related fields, the value of the covariate $X = z_2$ is called a reference value, and it is used to compute and interpret a set of comparisons of subpopulations defined by different covariate values.

4 The xblc command

4.1 Syntax

`xblc` *varlist*`, at(`*numlist*`) covname(`*varname*`)` [`reference(#) pr eform`
`format(%`*fmt*`) level(#) equation(`*string*`)`
`generate(`*newvar1 newvar2 newvar3 newvar4*`)`]

4.2 Description

`xblc` computes point and interval estimates for predictions or differences in predictions of the response variable evaluated at different values of a quantitative covariate modeled using one or more transformations of the original variable specified in *varlist*. It can be used after any estimation command.

4.3 Options

`at(`*numlist*`)` specifies the values of the covariate specified in `covname()`, at which `xblc` evaluates predictions or differences in predictions. The values need to be in the current dataset. Covariates other than the one specified with the `covname()` option are fixed at zero. This is a required option.

`covname(`*varname*`)` specifies the name of the quantitative covariate. This is a required option.

`reference(`#`)` specifies the reference value for displaying differences in predictions.

`pr` computes and displays predictions (that is, mean response after linear regression, log odds after logistic models, and log rate after Poisson models with person-time as offset) rather than differences in predictions. To use this option, check that the previously fit model estimates the constant `_b[_cons]`.

`eform` displays the exponential value of predictions or differences in predictions.

`format(%`*fmt*`)` specifies the display format for presenting numbers. `format(%3.2f)` is the default; see [D] **format**.

`level(`#`)` specifies the confidence level, as a percentage, for CIs. The default is `level(95)` or as set by `set level`.

`equation(`*string*`)` specifies the name of the equation when you have previously fit a multiple-equation model.

`generate(`*newvar1 newvar2 newvar3 newvar4*`)` specifies that the values of the original covariate, predictions or differences in predictions, and the lower and upper bounds of the CI be saved in *newvar1*, *newvar2*, *newvar3*, and *newvar4*, respectively. This option is very useful for presenting the results in a graphical form.

5 Examples

As an illustrative example, we analyze in a cross-sectional setting a sample of 30,377 men (`pa_luts.dta`) in central Sweden aged 45–79 years who completed a self-administered lifestyle questionnaire that included international prostate symptom score (IPSS) questions and physical activity questions (work/occupation, home/household work, walking/bicycling, exercise, and leisure-time such as watching TV/reading) (Orsini et al. 2006). The range of the response variable, the IPSS score, is 0 to 35. According to the American Urological Association, the IPSS score (variable `ipss2`) is categorized in two levels: mild or no symptoms (scores 0–7) and moderate to severe LUTS (scores 8–35). The main covariate of interest is a total physical activity score (variable `tpa`), which comprises a combination of intensity and duration for a combination of daily activities and is expressed in metabolic equivalents (MET) (kcal/kg/hour).

The proportion of men reporting moderate to severe LUTS is 6905/30377 = 0.23. The odds in favor of experiencing moderate to severe LUTS are $0.23/(1 - 0.23) = 6905/23472 = 0.29$; this means that on average, for every 100 men with mild or no symptoms, we observed 29 other men with moderate to severe LUTS, written as 29:100 (29 to 100 odds). Examining the variation of the ratio of cases/noncases (odds) of moderate to severe LUTS according to subpopulations of men defined by intervals of total physical activity (variable `tpac`) is our first step in describing the shape of the covariate–response association (figure 1).

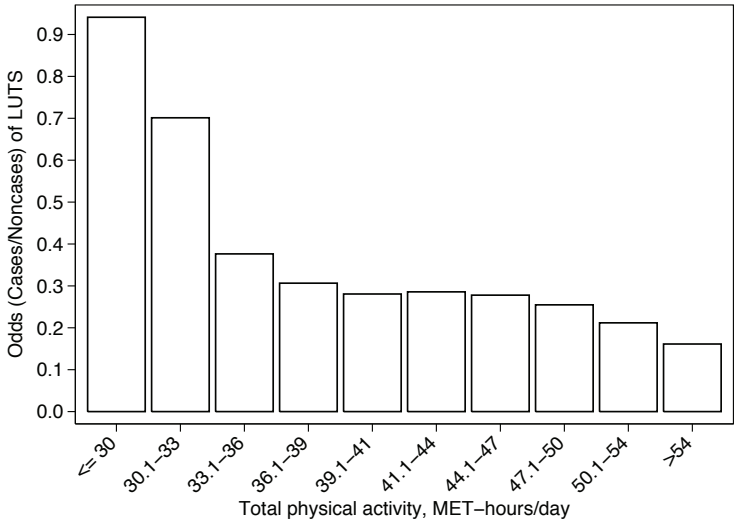


Figure 1. Observed odds (ratio of cases/noncases) of moderate to severe LUTS by categories of total physical activity (MET-hours/day) in a cohort of 30,377 Swedish men.

The occurrence of moderate to severe LUTS decreases more rapidly at the low values of the covariate distribution. There is a strong reduction of the odds of moderate to severe LUTS, going from 94:100 at the minimum total physical activity interval ($\geq$ 30 MET-hours/day) down to 38:100 at the interval 33.1 to 36 MET-hours/day. It follows a more gradual decline in the odds of moderate to severe LUTS to 16:100 in men at the highest total physical activity interval ($>$ 54 MET-hours/day).

Table 1 provides a tabular presentation of the data (total number of men, sum of the cases, range and median value of the covariate) by intervals of total physical activity. About 99% of the participants and 99% of the cases of moderate to severe LUTS are within the range 29 to 55 MET-hours/day. Therefore, results are presented within this range.

Table 1. Tabular presentation of data, unadjusted and age-adjusted ORs with 95% CI for the association of total physical activity (MET-hours/day) and occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men.

No. of subjects	No. of cases	Exposure range	Exposure median	Unadjusted OR [95% CI]*	Age-adjusted OR [95% CI]*
66	32	≤ 30	29	1.00	1.00
427	176	30.1–33	32	0.71 [0.55, 0.93]	0.85 [0.65, 1.12]
2761	755	33.1–36	35	0.42 [0.30, 0.58]	0.60 [0.43, 0.84]
7524	1765	36.1–39	38	0.31 [0.23, 0.43]	0.47 [0.34, 0.66]
5074	1112	39.1–41	40	0.31 [0.23, 0.43]	0.45 [0.32, 0.62]
5651	1256	41.1–44	43	0.31 [0.23, 0.43]	0.41 [0.30, 0.57]
4782	1040	44.1–47	45	0.30 [0.22, 0.41]	0.40 [0.29, 0.55]
2359	479	47.1–50	48	0.27 [0.20, 0.37]	0.39 [0.28, 0.54]
1373	240	50.1–54	52	0.24 [0.17, 0.33]	0.37 [0.27, 0.52]
360	50	> 54	55	0.21 [0.15, 0.30]	0.36 [0.25, 0.51]

* Total physical activity expressed in MET-hours/day was modeled by right-restricted cubic splines with four knots (37.2, 39.6, 42.3, and 45.6) at percentiles 20%, 40%, 60%, and 80% in a logistic regression model. The value of 29 MET-hours/day, as the median value of the lowest reference range of total physical activity, was used to estimate all ORs.

5.1 Unrestricted cubic splines

We first create unrestricted cubic splines with four knots at fixed and equally spaced percentiles (20%, 40%, 60%, and 80%). Varying the location of the knots (for instance, using percentiles 5%, 35%, 65%, and 95% as recommended by Harrell's book [2001]) had negligible influence on the estimates.

```
. generate all = 1
. table all, contents(freq p20 tpa p40 tpa p60 tpa p80 tpa)
```

all	Freq.	p20(tpa)	p40(tpa)	p60(tpa)	p80(tpa)
1	30,377	37.2	39.6	42.3	45.6

```
. generate tpa2 = tpa^2
. generate tpa3 = tpa^3
. generate tpap1 = max(0,tpa-37.2)^3
. generate tpap2 = max(0,tpa-39.6)^3
. generate tpap3 = max(0,tpa-42.3)^3
. generate tpap4 = max(0,tpa-45.6)^3
```

Ideally, the number of knots and their placement will result in categories with reasonably large numbers of both cases and noncases in each category. While there are no simple and foolproof rules, we recommend that each category have at least five and preferably more cases and noncases in each category and that the number of cases and number of noncases each are at least five times the number of model parameters. Further discussion on the choice of location and number of knots can be found in section 2.4.5 of Harrell's book (2001). Harrell also discusses more general aspects of model selection for dose–response (trend) analysis, as do Royston and Sauerbrei (2007).

We first fit a logistic regression model with unrestricted cubic splines for physical activity and no other covariate.

```
. logit ipss2 tpa tpa2 tpa3 tpap1 tpap2 tpap3 tpap4
Iteration 0:   log likelihood = -16282.244
Iteration 1:   log likelihood = -16187.593
Iteration 2:   log likelihood = -16185.014
Iteration 3:   log likelihood = -16185.014
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(7)      =     194.46
                                                  Prob > chi2     =     0.0000
Log likelihood = -16185.014                       Pseudo R2       =     0.0060
```

ipss2	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
tpa	9.759424	3.383661	2.88	0.004	3.12757	16.39128
tpa2	-.2985732	.0986663	-3.03	0.002	-.4919556	-.1051909
tpa3	.0029866	.0009553	3.13	0.002	.0011143	.0048589
tpap1	-.009595	.0035502	-2.70	0.007	-.0165532	-.0026368
tpap2	.0094618	.0052404	1.81	0.071	-.0008093	.0197328
tpap3	-.0049394	.0040546	-1.22	0.223	-.0128863	.0030074
tpap4	.0027824	.0019299	1.44	0.149	-.0010001	.0065649
_cons	-104.8292	38.52542	-2.72	0.007	-180.3376	-29.32074

Because the model omits other covariates, it is called uncontrolled or unadjusted analysis, also known as "crude" analysis.

The one-line postestimation command `xblc` is used to tabulate and plot contrasts of covariate values. It allows the user to specify a set of covariate values (here 29, 32, 35, 38, 40, 43, 45, 48, 52, and 55) at which it computes the ORs, using the value of 29 MET-hours/day as a referent.

```
. xblc tpa tpa2 tpa3 tpap1 tpap2 tpap3 tpap4, covname(tpa)
> at(29 32 35 38 40 43 45 48 52 55) reference(29) eform generate(pa or lb ub)
tpa           exp(xb)    (95% CI)
29            1.00       (1.00-1.00)
32            0.71       (0.55-0.93)
35            0.41       (0.30-0.57)
38            0.31       (0.23-0.43)
40            0.31       (0.23-0.42)
43            0.30       (0.22-0.41)
45            0.30       (0.22-0.41)
48            0.28       (0.20-0.38)
52            0.22       (0.16-0.31)
55            0.19       (0.13-0.28)
```

We specify the `eform` option of `xblc` because we are interested in presenting ORs rather than the difference between two log odds of the binary response. For plotting the ORs, a convenient `xblc` option is `generate()`, which saves the above four columns of numbers in the current dataset. The following code produces a standard two-way plot, as shown in figure 2:

```
. twoway (rcap lb ub pa, sort) (scatter or pa, sort), legend(off)
> scheme(s1mono) xlabel(29 32 35 38 40 43 45 48 52 55) ylabel(.2(.2)1.2,
> angle(horiz)  forma t(%2.1fc)) ytitle("Unadjusted Odds Ratios of LUTS")
> xtitle("Total physical activity, MET-hours/day")
```

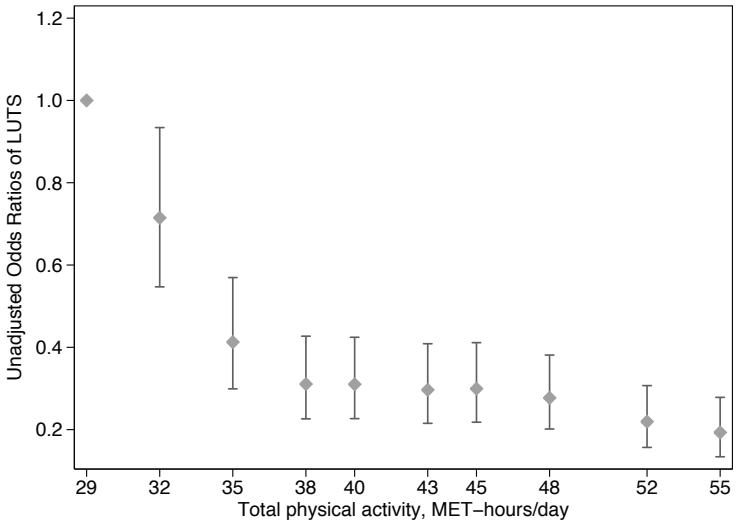


Figure 2. This graph shows unadjusted ORs (dots) with 95% CI (capped spikes) for the relation of total physical activity (MET-hours/day) to the occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men. Total physical activity was modeled by unrestricted cubic splines with four knots (37.2, 39.6, 42.3, and 45.6) at percentiles 20%, 40%, 60%, and 80% in a logistic regression model. The reference value is 29 MET-hours/day.

To get a better idea of the dose–response relation, one can compute the ORs and 95% confidence limits of moderate to severe LUTS for any subpopulation of men defined by a finer grid of values (using, say, a 1 MET-hour/day increment) across the range of interest (figure 3).

```
. capture drop pa or lb ub
. xblc tpa tpa2 tpa3 tpap*, covname(tpa) at(29(1)55) reference(29) eform
> generate(pa or lb ub)
```

(*output omitted*)

```
. twoway (rcap lb ub pa, sort) (scatter or pa, sort), legend(off)
> scheme(s1mono) xlabel(29(2)55) xmtick(29(1)55)
> ylabel(.2(.2)1.2, angle(horiz) format(%2.1fc))
> ytitle("Unadjusted Odds Ratios of LUTS")
> xtitle("Total physical activity, MET-hours/day")
```

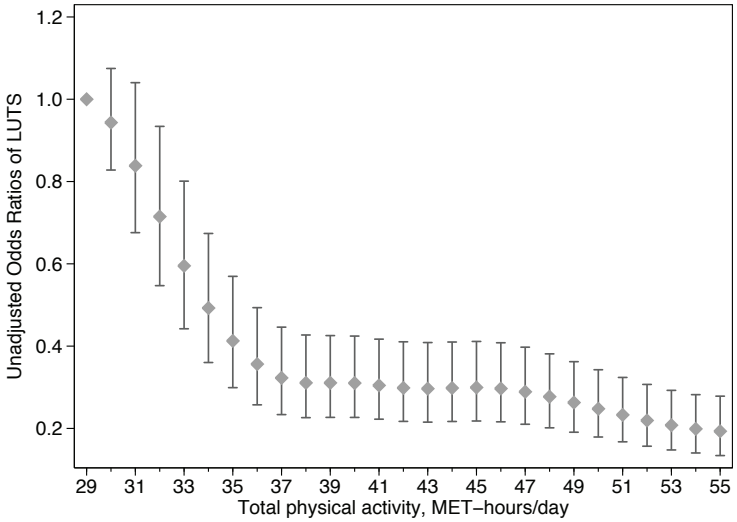


Figure 3. This graph shows unadjusted ORs (dots) with 95% CI (capped spikes) for the relation of total physical activity (MET-hours/day) to the occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men. Total physical activity was modeled by unrestricted cubic splines with four knots (37.2, 39.6, 42.3, and 45.6) at percentiles 20%, 40%, 60%, and 80% in a logistic regression model. The reference value is 29 MET-hours/day.

To produce a smooth graph of the relation, one can estimate all the differences in the log odds of moderate to severe LUTS corresponding to the 315 distinct observed exposure values, and then control how the point estimates and CIs are to be connected (figure 4).

```
. capture drop pa or lb ub
. quietly levelsof tpa, local(levels)
. quietly xblc tpa tpa2 tpa3 tpap*, covname(tpa) at(`r(levels)´) reference(29)
> eform generate(pa or lb ub)
. twoway (line lb ub pa, sort lc(black black) lp(- -))
> (line or pa, sort lc(black) lp(l)) if inrange(pa,29,55), legend(off)
> scheme(s1mono) xlabel(29(2)55) xmtick(29(1)55)
> ylabel(.2(.2)1.2, angle(horiz)  format(%2.1fc))
> ytitle("Unadjusted Odds Ratios of LUTS")
> xtitle("Total physical activity, MET-hours/day")
```

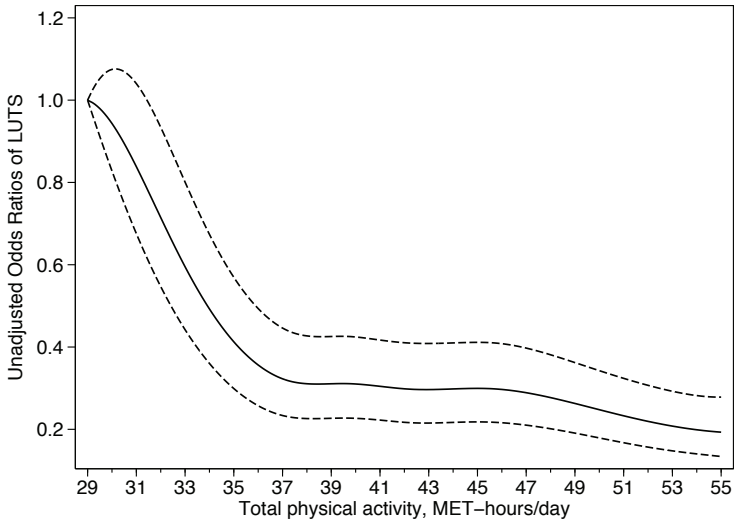


Figure 4. This graph shows unadjusted ORs (solid line) with 95% CI (dashed lines) for the relation of total physical activity (MET-hours/day) to the occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men. Total physical activity was modeled by unrestricted cubic splines with four knots (37.2, 39.6, 42.3, and 45.6) at percentiles 20%, 40%, 60%, and 80% in a logistic regression model. The reference value is 29 MET-hours/day.

5.2 Cubic splines with only one tail restricted

The observed odds of moderate to severe LUTS decreases more rapidly on the left tail of the physical activity distribution (see figure 1), which suggests that restricting the curve to be linear before the first knot placed at 37.2 MET-hours/day (20th percentile) is probably not a good idea. On the other hand, the right tail of the distribution above 45.6 MET-hours/day (80th percentile) shows a more gradual decline of the odds of moderate to severe LUTS, suggesting that restriction there is not unreasonable.

The left-tail restricted cubic-spline model just drops the quadratic and cubic terms of the previously fit unrestricted model. Given that the model that is left-tail restricted is nested within the unrestricted model, a Wald-type test for nonlinearity beyond the first knot is given by

```
. testparm tpa2 tpa3
 ( 1)  [ipss2]tpa2 = 0
 ( 2)  [ipss2]tpa3 = 0
           chi2(  2) =   18.42
         Prob > chi2 =    0.0001
```

The small p-value of the Wald-type test with two degrees of freedom indicates nonlinearity beyond the first knot. We show how to fit the model and then present the results:

```
. logit ipss2 tpa tpap1 tpap2 tpap3 tpap4
Iteration 0:   log likelihood = -16282.244
Iteration 1:   log likelihood = -16195.263
Iteration 2:   log likelihood = -16194.212
Iteration 3:   log likelihood = -16194.212
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(5)      =     176.07
                                                  Prob > chi2     =     0.0000
Log likelihood = -16194.212                       Pseudo R2       =     0.0054
------------------------------------------------------------------------------
       ipss2 |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         tpa |  -.0989318   .0101559    -9.74   0.000     -.118837   -.0790267
       tpap1 |   .0042758   .0009338     4.58   0.000     .0024457    .0061059
       tpap2 |  -.0095806   .0027518    -3.48   0.000     -.014974   -.0041873
       tpap3 |   .0060958    .003147     1.94   0.053    -.0000722    .0122638
       tpap4 |  -.0004933   .0017884    -0.28   0.783    -.0039985     .003012
       _cons |   2.549523   .3753997     6.79   0.000     1.813753    3.285293
------------------------------------------------------------------------------
```

Similarly to what we did after the estimation of the unrestricted cubic-spline model, we use the postestimation command `xblc` to present a set of ORs with 95% confidence limits. The only difference in the syntax of this `xblc` command is the list of transformations used to model physical activity.

```
. xblc tpa tpap*, covname(tpa) at(29 32 35 38 40 43 45 48 52 55) reference(29)
> eform
tpa             exp(xb)     (95% CI)
29              1.00        (1.00-1.00)
32              0.74        (0.70-0.79)
35              0.55        (0.49-0.62)
38              0.41        (0.34-0.49)
40              0.37        (0.31-0.45)
43              0.40        (0.34-0.47)
45              0.39        (0.33-0.46)
48              0.35        (0.29-0.42)
52              0.29        (0.24-0.35)
55              0.25        (0.20-0.32)
```

When assuming linearity only in the right tail of the covariate distribution, as explained in section 2, we first generate the cubic splines based on the negative of the original exposure. We then fit the model:

```
. generate tpan = -tpa
. generate tpapn1 = max(0,45.6-tpa)^3
. generate tpapn2 = max(0,42.3-tpa)^3
. generate tpapn3 = max(0,39.6-tpa)^3
. generate tpapn4 = max(0,37.2-tpa)^3
. logit ipss2 tpan tpapn*
Iteration 0:   log likelihood = -16282.244
Iteration 1:   log likelihood = -16189.534
Iteration 2:   log likelihood = -16187.088
Iteration 3:   log likelihood = -16187.087
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(5)      =     190.31
                                                  Prob > chi2     =     0.0000
Log likelihood = -16187.087                       Pseudo R2       =     0.0058
```

ipss2	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
tpan	.0325003	.0074057	4.39	0.000	.0179853	.0470153
tpapn1	-.0007537	.0005094	-1.48	0.139	-.0017521	.0002446
tpapn2	.0018099	.0023017	0.79	0.432	-.0027014	.0063212
tpapn3	.0029923	.0041652	0.72	0.473	-.0051714	.0111561
tpapn4	-.006665	.0032405	-2.06	0.040	-.0130163	-.0003136
_cons	.1675475	.3437625	0.49	0.626	-.5062146	.8413095

Once again, the postestimation command `xblc` facilitates the presentation, interpretation, and comparison of the results arising from different models.

```
. xblc tpan tpapn*, covname(tpa) at(29 32 35 38 40 43 45 48 52 55)
> reference(29) eform
tpa           exp(xb)    (95% CI)
29            1.00       (1.00-1.00)
32            0.71       (0.55-0.93)
35            0.42       (0.30-0.58)
38            0.31       (0.23-0.43)
40            0.31       (0.23-0.43)
43            0.31       (0.23-0.43)
45            0.30       (0.22-0.41)
48            0.27       (0.20-0.37)
52            0.24       (0.17-0.33)
55            0.21       (0.15-0.30)
```

The right-restricted cubic-spline model provides very similar ORs to the unrestricted model, but uses fewer coefficients.

5.3 Cubic splines with both tails restricted

To create a cubic spline that is restricted to being linear in both tails is more complicated, but the `mkspline` command facilitates this task.

```
. mkspline tpas = tpa, knots(37.2 39.6 42.3 45.6) cubic
```

The above line creates the restricted cubic splines, automatically named `tpas1`, `tpas2`, and `tpas3` using the defined knots. We then fit a logistic regression model that includes the three spline terms.

```
. logit ipss2 tpas1 tpas2 tpas3
Iteration 0:   log likelihood = -16282.244
Iteration 1:   log likelihood = -16195.572
Iteration 2:   log likelihood = -16194.592
Iteration 3:   log likelihood = -16194.592
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(3)      =     175.30
                                                  Prob > chi2     =     0.0000
Log likelihood = -16194.592                       Pseudo R2       =     0.0054

------------------------------------------------------------------------------
       ipss2 |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       tpas1 |  -.1009415   .0098873   -10.21   0.000    -.1203203   -.0815627
       tpas2 |    .337423   .0515938     6.54   0.000      .236301    .4385449
       tpas3 |  -.8010405    .130892    -6.12   0.000    -1.057584   -.5444968
       _cons |   2.620533   .3663134     7.15   0.000     1.902572    3.338494
------------------------------------------------------------------------------
```

To translate the estimated linear predictor into a set of ORs, we use the `xblc` command, as follows:

```
. xblc tpas*, covname(tpa) at(29 32 35 38 40 43 45 48 52 55) reference(29)
> eform
tpa             exp(xb)    (95% CI)
29              1.00       (1.00-1.00)
32              0.74       (0.70-0.78)
35              0.55       (0.49-0.61)
38              0.40       (0.34-0.48)
40              0.37       (0.30-0.44)
43              0.40       (0.34-0.47)
45              0.38       (0.32-0.45)
48              0.34       (0.29-0.40)
52              0.29       (0.24-0.35)
55              0.26       (0.21-0.32)
```

Figure 5 shows a comparison of the four different types of cubic splines. Given the same number and location of knots, the greatest impact on the curve is given by the inappropriate linear constraint before the first knot. Using Akaike's information criterion (a summary measure that combines fit and complexity), we found that the unrestricted and right-restricted cubic-spline models have a better fit (smaller Akaike's information criterion) compared with the left- and both-tail restricted cubic-spline models.

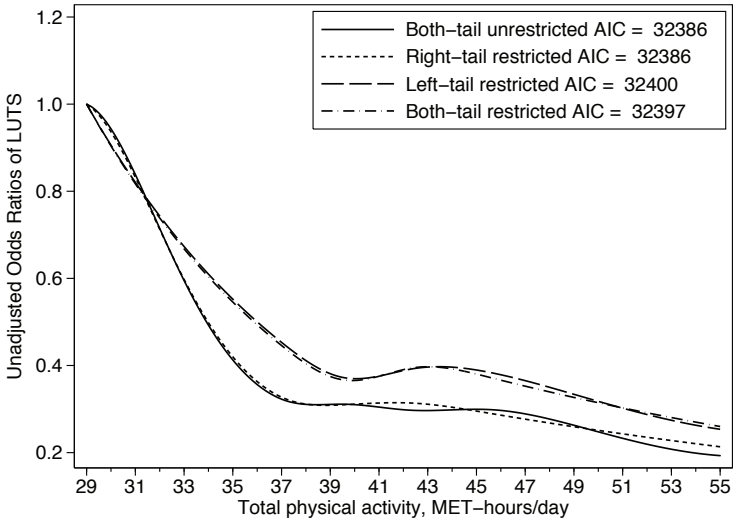


Figure 5. This graph compares unadjusted ORs for the relation of total physical activity (MET-hours/day) with the occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men. Total physical activity was modeled by both-tail unrestricted, left-tail restricted, right-tail restricted, and both-tail restricted cubic splines with four knots (37.2, 39.6, 42.3, and 45.6) at percentiles 20%, 40%, 60%, and 80% in a logistic regression model. The reference value is 29 MET-hours/day.

The right-restricted model has a smaller number of regression coefficients than does the unrestricted model. Hence, we use the right-restricted model for further illustration of the `xblc` command with adjustment for other covariates and for the presentation of adjusted trends and confidence bands for the predicted occurrence of the binary response.

5.4 Adjusting for other covariates

Men reporting different physical activity levels may differ with respect to sociodemographic, biological, anthropometrical, health, and other lifestyle factors, so the crude estimates given above are unlikely to accurately reflect the causal effects of physical activity on the outcome. We now show that adjusting for such variables (known as potential confounders) does not change how the postestimation command `xblc` works.

Consider age, the strongest predictor of urinary problems. Moderate to severe LUTS increases with age and occurs in most elderly men, while total physical activity decreases with age. Therefore, the estimated decreasing odds of moderate to severe LUTS in subpopulations of men reporting higher physical activity levels might be explained by differences in the distribution of age. Thus we include age, centered on the sample mean

of 59 years, in the right-tail restricted cubic-spline model. For simplicity, we assume a linear relation of age to the log odds of moderate to severe LUTS. We could also use splines for age, but it has negligible influence on the main covariate–disease association in our example.

```
. quietly summarize age
. generate agec = age - r(mean)
. logit ipss2 tpan tpapn* agec
Iteration 0:   log likelihood = -16282.244
Iteration 1:   log likelihood = -15533.528
Iteration 2:   log likelihood = -15517.532
Iteration 3:   log likelihood = -15517.526
Iteration 4:   log likelihood = -15517.526
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(6)      =    1529.44
                                                  Prob > chi2     =     0.0000
Log likelihood = -15517.526                       Pseudo R2       =     0.0470
```

ipss2	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
tpan	.0104343	.0076532	1.36	0.173	-.0045657	.0254343
tpapn1	.0004587	.0005237	0.88	0.381	-.0005676	.0014851
tpapn2	-.0015097	.0023617	-0.64	0.523	-.0061385	.003119
tpapn3	.0040955	.0042703	0.96	0.338	-.0042742	.0124651
tpapn4	-.0048478	.0033233	-1.46	0.145	-.0113615	.0016658
agec	.0552749	.0015376	35.95	0.000	.0522612	.0582885
_cons	-.9478404	.3556108	-2.67	0.008	-1.644825	-.2508561

The syntax of the `xblc` command in the presence of another covariate is the same as that used for the unadjusted analysis.

```
. xblc tpan tpapn*, covname(tpa) at(29 32 35 38 40 43 45 48 52 55)
> reference(29) eform
tpa           exp(xb)    (95% CI)
29            1.00       (1.00-1.00)
32            0.85       (0.65-1.12)
35            0.60       (0.43-0.84)
38            0.47       (0.34-0.66)
40            0.45       (0.32-0.62)
43            0.41       (0.30-0.57)
45            0.40       (0.29-0.55)
48            0.39       (0.28-0.54)
52            0.37       (0.27-0.52)
55            0.36       (0.25-0.51)
```

As expected, the age-adjusted ORs of moderate to severe LUTS are generally lower compared with the crude ORs. Thus the association between physical activity and the outcome was partly explained by differences in age (table 1). Entering more covariates in the model does not change the `xblc` postestimation command. To obtain figure 6, the code is as follows:

```
. capture drop pa or lb ub
. quietly levelsof tpa, local(levels)
. quietly xblc tpan tpapn*, covname(tpa) at(`r(levels)´) reference(29) eform
> generate(pa or lb ub)
. twoway (line lb ub pa, sort lc(black black) lp(- -))
> (line or pa, sort lc(black) lp(l)) if inrange(pa,29,55), legend(off)
> scheme(s1mono) xlabel(29(2)55) xmtick(29(1)55)
> ylabel(.2(.2)1.2, angle(horiz) format(%2.1fc))
> ytitle("Age-adjusted Odds Ratios of LUTS")
> xtitle("Total physical activity, MET-hours/day")
```

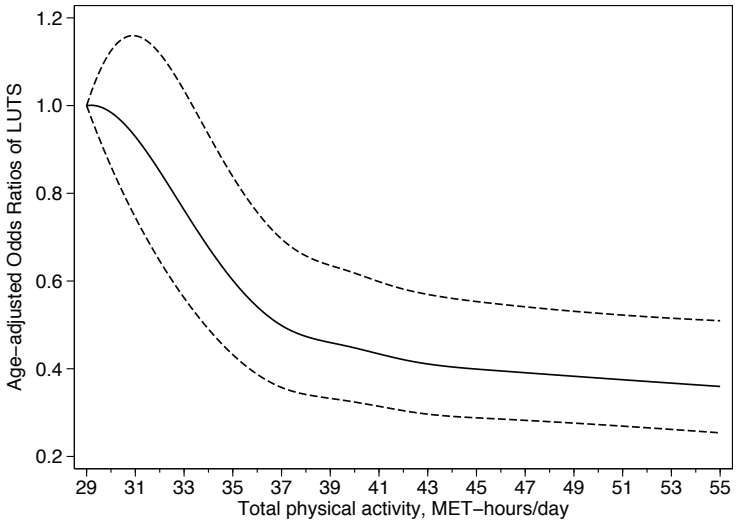


Figure 6. This graph shows age-adjusted ORs (solid line) with 95% CI (dashed lines) for the relation of total physical activity (MET-hours/day) to the occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men. Total physical activity was modeled by right-restricted cubic splines with four knots (37.2, 39.6, 42.3, and 45.6) at percentiles 20%, 40%, 60%, and 80% in a logistic regression model. The reference value is 29 MET-hours/day.

5.5 Uncertainty for the predicted response

So far we have focused on tabulating and plotting ORs as functions of covariate values. It is important to note that the CIs for the ORs that include the sampling variability of the reference value cannot be used to compare the odds of two nonreference values. The problem arises if one misinterprets the CIs of the OR as representing CIs for the odds. Further discussion of this issue can be found elsewhere (Greenland et al. 1999).

Those readers who wish to visualize uncertainty about the odds of the event rather than the ORs may add the `pr` option (predicted response, log odds in our example) in the previously typed `xblc` command.

```
. capture drop pa
. quietly levelsof tpa, local(levels)
. quietly xblc tpan tpapn*, covname(tpa) at(`r(levels)´) reference(29) eform
> generate(pa rcc lbo ubo) pr
. twoway (line lbo ubo pa, sort lc(black black) lp(- -))
> (line rcc pa, sort lc(black) lp(l)) if inrange(pa,29,55), legend(off)
> scheme(s1mono) xlabel(29(2)55) xmtick(29(1)55) ylabel(.2(.1).8, angle(horiz)
> format(%2.1fc)) ytitle("Age-adjusted Odds (Cases/Noncases) of LUTS")
> xtitle("Total physical activity, MET-hours/day")
```

Figure 7 shows that the CIs around the age-adjusted odds of moderate to severe LUTS widen at the extremes of the graph, properly reflecting sparse data in the tails of the distribution of total physical activity.

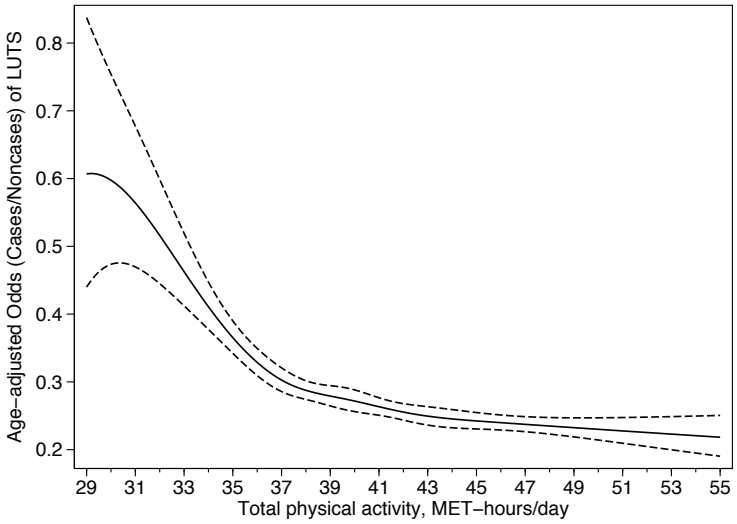


Figure 7. This graph shows age-adjusted odds (ratios of cases/noncases, solid line) with 95% CI (dashed lines) for the relation of total physical activity (MET-hours/day) to the occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men. Total physical activity was modeled by right-restricted cubic splines with four knots (37.2, 39.6, 42.3, and 45.6) at percentiles 20%, 40%, 60%, and 80% in a logistic regression model.

6 Use of xblc after other modeling approaches

A valuable feature of the `xblc` command is that its use is independent of the specific approach used to model a quantitative covariate. The command can be used with alternative parametric models such as piecewise-linear splines or fractional polynomials (Steenland and Deddens 2004; Royston, Ambler, and Sauerbrei 1999; Greenland 2008, 1995b). To illustrate, we next show the use of the `xblc` command with different modeling strategies (categorization, linear splines, and fractional polynomials), as shown in figure 8 (in section 6.3).

6.1 Categorical model

We fit a logistic regression model with $10 - 1 = 9$ indicator variables with the lowest interval (≤ 30 MET-hours/day) serving as a referent.

```
. xi:logit ipss2 i.tpac agec, or
i.tpac            _Itpac_1-10         (naturally coded; _Itpac_1 omitted)
Iteration 0:   log likelihood = -16282.244
Iteration 1:   log likelihood = -15537.912
Iteration 2:   log likelihood = -15521.805
Iteration 3:   log likelihood = -15521.798
Iteration 4:   log likelihood = -15521.798
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(10)     =    1520.89
                                                  Prob > chi2     =     0.0000
Log likelihood = -15521.798                       Pseudo R2       =     0.0467
```

ipss2	Odds Ratio	Std. Err.	z	P>|z|	[95% Conf.	Interval]
_Itpac_2	.8527221	.2336656	-0.58	0.561	.4983777	1.459004
_Itpac_3	.536358	.1386083	-2.41	0.016	.3232087	.8900749
_Itpac_4	.4730301	.1212153	-2.92	0.003	.2862635	.7816485
_Itpac_5	.4108355	.1055984	-3.46	0.001	.2482452	.6799158
_Itpac_6	.39818	.1022264	-3.59	0.000	.2407393	.6585851
_Itpac_7	.3798165	.0976648	-3.76	0.000	.2294556	.6287081
_Itpac_8	.3534416	.091875	-4.00	0.000	.2123503	.5882776
_Itpac_9	.3658974	.0969315	-3.80	0.000	.2177026	.6149715
_Itpac_10	.2925106	.0871772	-4.12	0.000	.1631012	.5245971
agec	1.056869	.0016265	35.94	0.000	1.053686	1.060062

We estimate the age-adjusted odds of the response with the `xblc` command, as shown in figure 8 (in section 6.3).

```
. quietly levelsof tpa, local(levels)
. quietly xblc _Itpac_2- _Itpac_10, covname(tpa) at(`r(levels)´) eform
> generate(pa oddsc lboc uboc) pr
```

The categorical model implies constant odds (ratio of cases/noncases) of moderate to severe LUTS within intervals of physical activity, with sudden jumps between intervals. The advantages of the categorical model are that it is easy to fit and to present in both tabular and graphical forms. The disadvantages (power loss, distortion of trends, and unrealistic dose–response step functions) of categorizing continuous variables have been pointed out several times (Royston, Altman, and Sauerbrei 2006; Greenland 1995a,b,c,d, 2008).

In our example, the differences between the categorical model and splines are greater at the low values of the covariate distribution (< 38 MET-hours/day) where the occurrence of moderate to severe LUTS decreases more rapidly (with a steeper slope) compared with the remaining covariate range. Another difference between the two models is the amount of information used in estimating associations. The odds or ratios of odds from the categorical model are only determined by the data contained in the exposure intervals being compared. One must ignore the magnitude and direction of the association in the remaining exposure intervals. For instance, in the categorical model fit to 30,377 men, the age-adjusted OR comparing the interval 30.1–33 MET-hours/day with the reference interval (≤ 30 MET-hours/day) is 0.85 [95% CI $= 0.50, 1.46$]. We would estimate practically the same adjusted OR and 95% CI by restricting the model to 1.6% of the sample (486 men) belonging to the first two categories of total physical activity being compared.

Not surprisingly, the width of the 95% CI around the fitted OR is greater in categorical models compared with restricted cubic-spline models. The fitted OR from a spline model uses the full covariate information for all individuals, and the CI gradually increases with the distance between the covariate values being compared, as it should.

The large sample size and the relatively large number of cases allow us to categorize physical activity in 10 narrow intervals. Therefore, the fitted trend based on the categorical model is overall not that different from the fitted trends based on splines and fractional polynomials (see table 2 on the next page and figure 8 in section 6.3). However, the shape of the covariate–response relationship in categorical models is sensitive to the location and number of cutpoints used to categorize the continuous covariate—potentially more sensitive than fitted curves with the same number of parameters will be to the choice of knots or polynomial terms.

Table 2. Comparison of age-adjusted OR with 95% CI for the association of total physical activity (MET-hours/day) and occurrence of moderate to severe LUTS estimated with different types of models: categorical, linear spline, and fractional polynomial

Exposure range	Exposure median	Categorical model OR [95% CI] *	Linear spline model OR [95% CI] †	Fractional polynomial model OR [95% CI] ‡
≤ 30	29	1.00	1.00	1.00
30.1–33	32	0.85 [0.50, 1.46]	0.76 [0.71, 0.82]	0.69 [0.62, 0.77]
33.1–36	35	0.54 [0.32, 0.89]	0.58 [0.51, 0.67]	0.53 [0.45, 0.63]
36.1–39	38	0.47 [0.29, 0.78]	0.44 [0.36, 0.54]	0.45 [0.36, 0.55]
39.1–41	40	0.41 [0.25, 0.68]	0.43 [0.35, 0.52]	0.41 [0.33, 0.51]
41.1–44	43	0.40 [0.24, 0.66]	0.41 [0.34, 0.49]	0.37 [0.30, 0.47]
44.1–47	45	0.38 [0.23, 0.63]	0.39 [0.33, 0.47]	0.36 [0.29, 0.45]
47.1–50	48	0.35 [0.21, 0.59]	0.37 [0.31, 0.45]	0.35 [0.28, 0.43]
50.1–54	52	0.37 [0.22, 0.61]	0.35 [0.29, 0.41]	0.34 [0.28, 0.41]
> 54	55	0.29 [0.16, 0.52]	0.33 [0.27, 0.39]	0.35 [0.29, 0.42]

* Nine indicator variables.
† One knot at 38 MET-hours/day.
‡ Degree-2 fractional polynomials with powers (0.5, 0.5).

6.2 Linear splines

The slope of the curve (change in the odds of moderate to severe LUTS per 1 MET-hours/day increase in total physical activity) for the age-adjusted association is much steeper below 38 MET-hours/day when compared with higher covariate levels (see figure 7). For example, assume a simple linear trend for total physical activity where we allow the slope to change at 38 MET-hours/day. We then create a linear spline and fit the model, including both the original MET variable and the spline, to obtain a connected, piecewise-linear curve.

```
. generate tpa38p = max(tpa-38, 0)
. logit ipss2 tpa tpa38p agec
Iteration 0:   log likelihood = -16282.244
Iteration 1:   log likelihood =  -15535.79
Iteration 2:   log likelihood = -15520.305
Iteration 3:   log likelihood = -15520.299
Iteration 4:   log likelihood = -15520.299
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(3)      =    1523.89
                                                  Prob > chi2     =     0.0000
Log likelihood = -15520.299                       Pseudo R2       =     0.0468
------------------------------------------------------------------------------
       ipss2 |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         tpa |  -.0902158   .0113921    -7.92   0.000    -.1125439   -.0678877
      tpa38p |    .072256   .0134459     5.37   0.000     .0459026    .0986094
        agec |   .0551684   .0015265    36.14   0.000     .0521766    .0581602
       _cons |   2.154075   .4203814     5.12   0.000     1.330142    2.978007
------------------------------------------------------------------------------
. xblc tpa tpa38p, covname(tpa) at(29 32 35 38 40 43 45 48 52 55) reference(29)
> eform
tpa           exp(xb)    (95% CI)
29            1.00       (1.00-1.00)
32            0.76       (0.71-0.82)
35            0.58       (0.51-0.67)
38            0.44       (0.36-0.54)
40            0.43       (0.35-0.52)
43            0.41       (0.34-0.49)
45            0.39       (0.33-0.47)
48            0.37       (0.31-0.45)
52            0.35       (0.29-0.41)
55            0.33       (0.27-0.39)
```

The above set of age-adjusted ORs computed with the `xblc` command, based on a linear spline model, is very similar to the one estimated with a more complicated right-restricted cubic-spline model (table 2). The advantage of the linear spline in this example is that it captures the most prominent features of the covariate–response association with just two parameters. The disadvantage is that the linear spline can be thrown off very far if the knot selected is poorly placed; that is, for a given number of knots, it is more sensitive to knot placement than to splines with power terms.

To express the linear trend for two-unit increases before and after the knot, we type

```
. lincom tpa*2, eform
 ( 1)  2*[ipss2]tpa = 0
```

ipss2	exp(b)	Std. Err.	z	P>|z|	[95% Conf.	Interval]
(1)	.8349098	.0190227	-7.92	0.000	.7984461	.8730387

```
. lincom tpa*2 + tpa38p*2, eform
 ( 1)  2*[ipss2]tpa + 2*[ipss2]tpa38p = 0
```

ipss2	exp(b)	Std. Err.	z	P>|z|	[95% Conf.	Interval]
(1)	.9647179	.0073474	-4.72	0.000	.9504242	.9792265

For every 2 MET-hours/day increase in total physical activity, the odds of moderate to severe LUTS significantly decrease by 17% below 38 MET-hours/day and by 4% above 38 MET-hours/day.

6.3 Fractional polynomials

The Stata command `mfp` (see [R] **mfp**) provides a systematic search for the best-fitting (likelihood maximizing) fractional-polynomial function (Royston, Ambler, and Sauerbrei 1999) for the quantitative covariates in the model.

```
. mfp logit ipss2 tpa agec, df(agec:1)
  (output omitted)
Fractional polynomial fitting algorithm converged after 2 cycles.
Transformations of covariates:
-> gen double Itpa__1 = X^-.5-.4908581303 if e(sample)
-> gen double Itpa__2 = X^-.5*ln(X)-.6985894219 if e(sample)
   (where: X = tpa/10)
-> gen double Iagec__1 = agec-1.46506e-07 if e(sample)
Final multivariable fractional polynomial model for ipss2
```

Variable	Initial			Final		
	df	Select	Alpha	Status	df	Powers
tpa	4	1.0000	0.0500	in	4	-.5 -.5
agec	1	1.0000	0.0500	in	1	1

```
Logistic regression                               Number of obs   =      30377
                                                  LR chi2(3)      =    1523.28
                                                  Prob > chi2     =     0.0000
Log likelihood = -15520.604                       Pseudo R2       =     0.0468
```

ipss2	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
Itpa__1	-9.545011	3.716881	-2.57	0.010	-16.82996	-2.260058
Itpa__2	-25.4474	6.240732	-4.08	0.000	-37.67901	-13.21579
Iagec__1	.0555331	.0015258	36.40	0.000	.0525426	.0585237
_cons	-1.353425	.0178889	-75.66	0.000	-1.388487	-1.318363

```
Deviance:31041.208.
```

The algorithm found that the best transformation for total physical activity is a degree-2 fractional polynomial with equal powers (0.5, 0.5). To compute the ORs shown in table 2, we type

```
. xblc Itpa__1 Itpa__2, covname(tpa) at(29 32 35 38 40 43 45 48 52 55)
> reference(29) eform
tpa           exp(xb)    (95% CI)
29            1.00       (1.00-1.00)
32            0.69       (0.62-0.77)
35            0.53       (0.45-0.63)
38            0.45       (0.36-0.55)
40            0.41       (0.33-0.51)
43            0.37       (0.30-0.47)
45            0.36       (0.29-0.45)
48            0.35       (0.28-0.43)
52            0.34       (0.28-0.41)
55            0.35       (0.29-0.42)
```

The advantage of using fractional polynomials is that just one or two transformations of the original covariate can accommodate a variety of possible covariate–response relationships. The disadvantage is that the fitted curve can be sensitive to extreme values of the quantitative covariate (Royston, Ambler, and Sauerbrei 1999; Royston and Sauerbrei 2008).

Figure 8 provides a graphical comparison of the age-adjusted odds of moderate to severe LUTS obtained with the `xblc` command using the different modeling strategies discussed above.

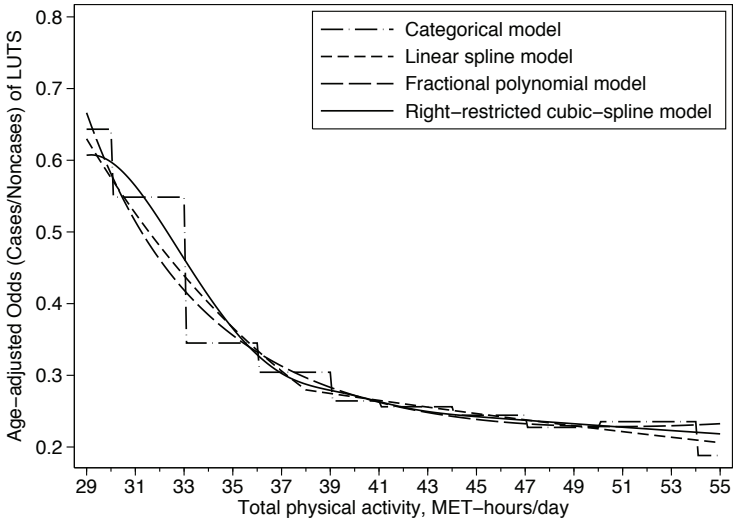


Figure 8. Comparison of covariate models (indicator variables, linear splines with a knot at 38 MET-hours/day, degree-2 fractional polynomial with powers [0.5, 0.5], right-restricted cubic-spline with four knots at percentiles 20%, 40%, 60%, and 80%) for estimating age-adjusted odds for the relation of total physical activity (MET-hours/day) to the occurrence of moderate to severe LUTS in a cohort of 30,377 Swedish men.

7 Conclusion

We have provided a new Stata command, `xblc`, to facilitate the presentation of the association between a quantitative covariate and the response variable. In the context of logistic regression with an emphasis on the use of different type of cubic splines, we illustrated how to present the odds or ORs with 95% confidence limits in tabular and graphical form.

The steps necessary to present the results can be applied to other types of models. The postestimation `xblc` command can be used after the majority of regression analysis (that is, generalized-linear models, quantile regression, survival-time models, longitudinal/panel-data models, meta-regression models) because the way of contrasting predicted responses is similar. The `xblc` command can be used to describe the relation of any quantitative covariate to the outcome using any type of flexible modeling strategy (that is, splines or fractional polynomials). If one is interested in plotting predicted

or marginal effects to a quantitative covariate, one can use the `postrcspline` package (Buis 2008). However, unlike the `xblc` command, the `postrcspline` command works only after fitting a restricted cubic-spline model.

Advantages of flexibly modeling a quantitative covariate include the ability to fit smooth curves efficiently and realistically. The fitted curves still need careful interpretation supported by subject-matter knowledge. Explanations for the observed shape may involve chance, mismeasurement, selection bias, or confounding rather than an effect of the fitted covariate (Orsini et al. 2008; Greenland and Lash 2008). For instance, in our unadjusted analysis, the OR for moderate to severe LUTS is not always decreasing with higher physical activity values. Once we adjust for age, this counterintuitive phenomenon disappears.

This example occurred in a large study in the middle of the exposure distribution where a large number of cases were located. Therefore, the investigator should be aware of the potential problems (instability, limited ability to predict future observations, and increased chance of overinterpretation and overfitting) with methods that can closely fit data (Steenland and Deddens 2004; Greenland 1995b; Royston and Sauerbrei 2007, 2009). Thus, as with any other strategy, subject-matter knowledge is needed when fitting regression models using flexible tools. Other important issues not considered here are how to deal with uncertainty due to model selection, how to assess goodness of fit, and how to handle zero exposure levels (Royston and Sauerbrei 2007, 2008; Greenland and Poole 1995).

In conclusion, the postestimation command `xblc` greatly facilitates the tabular and graphical presentation of results, thus aiding analysis and interpretation of covariate–response relations.

8 References

Buis, M. L. 2008. postrcspline: Stata module containing postestimation commands for models using a restricted cubic spline. Statistical Software Components S456928, Department of Economics, Boston College. http://ideas.repec.org/c/boc/bocode/s456928.html.

Durrleman, S., and R. Simon. 1989. Flexible regression models with cubic splines. *Statistics in Medicine* 8: 551–561.

Greenland, S. 1995a. Avoiding power loss associated with categorization and ordinal scores in dose–response and trend analysis. *Epidemiology* 6: 450–454.

———. 1995b. Dose–response and trend analysis in epidemiology: Alternatives to categorical analysis. *Epidemiology* 6: 356–365.

———. 1995c. Previous research on power loss associated with categorization in dose–response and trend analysis. *Epidemiology* 6: 641–642.

———. 1995d. Problems in the average-risk interpretation of categorical dose–response analyses. *Epidemiology* 6: 563–565.

———. 2008. Introduction to regression models. In *Modern Epidemiology*, ed. K. J. Rothman, S. Greenland, and T. L. Lash, 3rd ed., 381–417. Philadelphia: Lippincott Williams & Wilkins.

Greenland, S., and T. L. Lash. 2008. Bias analysis. In *Modern Epidemiology*, ed. K. J. Rothman, S. Greenland, and T. L. Lash, 3rd ed., 345–380. Philadelphia: Lippincott Williams & Wilkins.

Greenland, S., K. B. Michels, J. M. Robins, C. Poole, and W. C. Willett. 1999. Presenting statistical uncertainty in trends and dose–response relations. *American Journal of Epidemiology* 149: 1077–1086.

Greenland, S., and C. Poole. 1995. Interpretation and analysis of differential exposure variability and zero-exposure categories for continuous exposures. *Epidemiology* 6: 326–328.

Harrell, F. E., Jr. 2001. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis*. New York: Springer.

Harrell, F. E., Jr., K. L. Lee, and B. G. Pollock. 1988. Regression models in clinical studies: Determining relationships between predictors and response. *Journal of the National Cancer Institute* 80: 1198–1202.

Marrie, R. A., N. V. Dawson, and A. Garland. 2009. Quantile regression and restricted cubic splines are useful for exploring relationships between continuous variables. *Journal of Clinical Epidemiology* 62: 511–517.

Orsini, N., R. Bellocco, M. Bottai, A. Wolk, and S. Greenland. 2008. A tool for deterministic and probabilistic sensitivity analysis of epidemiologic studies. *Stata Journal* 8: 29–48.

Orsini, N., B. RashidKhani, S.-O. Andersson, L. Karlberg, J.-E. Johansson, and A. Wolk. 2006. Long-term physical activity and lower urinary tract symptoms in men. *Journal of Urology* 176: 2546–2550.

Royston, P., D. G. Altman, and W. Sauerbrei. 2006. Dichotomizing continuous predictors in multiple regression: A bad idea. *Statistics in Medicine* 25: 127–141.

Royston, P., G. Ambler, and W. Sauerbrei. 1999. The use of fractional polynomials to model continuous risk variables in epidemiology. *International Journal of Epidemiology* 28: 964–974.

Royston, P., and W. Sauerbrei. 2007. Multivariable modeling with cubic regression splines: A principled approach. *Stata Journal* 7: 45–70.

———. 2008. *Multivariable Model-building: A Pragmatic Approach to Regression Analysis Based on Fractional Polynomials for Modelling Continuous Variables*. Chichester, UK: Wiley.

———. 2009. Bootstrap assessment of the stability of multivariable models. *Stata Journal* 9: 547–570.

Smith, P. L. 1979. Splines as a useful and convenient statistical tool. *American Statistician* 33: 57–62.

Steenland, K., and J. A. Deddens. 2004. A practical guide to dose–response analyses and risk assessment in occupational epidemiology. *Epidemiology* 15: 63–70.

Wegman, E. J., and I. W. Wright. 1983. Splines in statistics. *Journal of the American Statistical Association* 78: 351–365.

About the authors

Nicola Orsini is a researcher in the Division of Nutritional Epidemiology at the National Institute of Environmental Medicine, Karolinska Institutet, Stockholm, Sweden.

Sander Greenland is a professor of epidemiology at the UCLA School of Public Health and a professor of statistics at the UCLA College of Letters and Science, Los Angeles, CA.

The Stata Journal (2011)
11, Number 1, pp. 30–51

Nonparametric item response theory using Stata

Jean-Benoit Hardouin
University of Nantes
Faculty of Pharmaceutical Sciences
Biostatistics, Clinical Research, and Subjective Measures in Health Sciences
Nantes, France
jean-benoit.hardouin@univ-nantes.fr

Angélique Bonnaud-Antignac
University of Nantes
Faculty of Medicine
ERT A0901 ERSSCA
Nantes, France

Véronique Sébille
University of Nantes
Faculty of Pharmaceutical Sciences
Biostatistics, Clinical Research, and Subjective Measures in Health Sciences
Nantes, France

Abstract. Item response theory is a set of models and methods allowing for the analysis of binary or ordinal variables (items) that are influenced by a latent variable or latent trait—that is, a variable that cannot be measured directly. The theory was originally developed in educational assessment but has many other applications in clinical research, ecology, psychiatry, and economics.

The Mokken scales have been described by Mokken (1971, *A Theory and Procedure of Scale Analysis* [De Gruyter]). They are composed of items that satisfy the three fundamental assumptions of item response theory: unidimensionality, monotonicity, and local independence. They can be considered nonparametric models in item response theory. Traces of the items and Loevinger's H coefficients are particularly useful indexes for checking whether a set of items constitutes a Mokken scale.

However, these indexes are not available in general statistical packages. We introduce Stata commands to compute them. We also describe the options available and provide examples of output.

Keywords: st0216, tracelines, loevh, gengroup, msp, items trace lines, Mokken scales, item response theory, Loevinger coefficients, Guttman errors

1 Introduction

Item response theory (IRT) (Van der Linden and Hambleton 1997) concerns models and methods where the responses to the items (binary or ordinal variables) of a questionnaire are assumed to depend on nonmeasurable characteristics (latent traits) of the respondents. These models can be applied to measures such as a latent variable (in measurement models) or to investigate influences of covariates on these latent variables.

Examples of latent traits include health status; quality of life; ability or content knowledge in a specific field of study; and psychological traits such as anxiety, impulsivity, and depression.

Most item response models (IRMs) are parametric: they model the probability of response at each category of each item by a function, depending on the latent trait, which is typically considered as a set of fixed effects or as a random variable, and they model the probability of parameters characterizing the items. The most popular IRMs for dichotomous items are the Rasch model and the Birnbaum model, and the most popular IRMs for polytomous items are the partial credit model and the rating scale model. These IRMs are already described for the Stata software (Hardouin 2007; Zheng and Rabe-Hesketh 2007).

Mokken (1971) defines a nonparametric model for studying the properties of a set of items in the framework of IRT. Mokken calls this model the monotonely homogeneous model, but it is generally referred to as the Mokken model. This model is implemented in a stand-alone package for the Mokken scale procedure (MSP) (Molenaar, Sijtsma, and Boer 2000), and codes already have been developed in Stata (Weesie 1999), SAS (Hardouin 2002), and R (Van der Ark 2007) languages. We propose commands under Stata to study the fit of a set of items to a Mokken model. These commands are more complete than the `mokken` command of Jeroen Weesie, for example, which does not offer the possibility of analyzing polytomous items.

The main purpose of the Mokken model is to validate an ordinal measure of a latent variable: for items that satisfy the criteria of the Mokken model, the sum of the responses across items can be used to rank respondents on the latent trait (Hemker et al. 1997; Sijtsma and Molenaar 2002). Compared with parametric IRT models, the Mokken model requires few assumptions regarding the relationship between the latent trait and the responses to the items; thus it generally allows keeping more important items. As a consequence, the ordering of individuals is more precise (Sijtsma and Molenaar 2002).

2 The Mokken scales

2.1 Notation

In the following text, we use the following notation:

- X_j is the random variable (item) representing the responses to the jth item, $j = 1, \ldots, J$.

- X_{nj} is the random variable (item) representing the responses to the jth item, $j = 1, \ldots, J$, for the nth individual, and x_{nj} is the realization of this variable.
- $m_j + 1$ is the number of response categories of the jth item.
- The response category 0 implies the smallest level of the latent trait and is referred to as a negative response, whereas the m_j nonzero response categories $(1, 2, \ldots, m_j)$ increase with increasing levels of the latent trait and are referred to as positive responses.
- M is the total number of possible positive responses across all items: $M = \sum_{j=1}^{J} m_j$
- Y_{jr} is the random-threshold dichotomous item taking the value 1 if $x_{nj} \geq r$ and 0 otherwise. There are M such items ($j = 1, \ldots, J$ and $r = 1, \ldots, m_j$).
- $P(.)$ refers to observed proportions.

2.2 Monotonely homogeneous model of Mokken (MHMM)

The Mokken scales are sets of items satisfying an MHMM (Mokken 1997; Molenaar 1997; Sijtsma and Molenaar 2002). This kind of model is a nonparametric IRM defined by the three fundamental assumptions of IRT:

- unidimensionality (responses to items are explained by a common latent trait)
- local independence (conditional on the latent trait, responses to items are independent)
- monotonicity (the probability of an item response greater than or equal to any fixed value is a nondecreasing function of the latent trait)

Unidimensionality implies that the responses to all the items are governed by a scalar latent trait. A practical advantage of this assumption is the easiness of interpreting the results. For a questionnaire aiming at measuring several latent traits, such an analysis must be realized for each unidimensional latent trait.

Local independence implies that all the relationships between the items are explained by the latent trait (Sijtsma and Molenaar 2002). This assumption is strongly related to the unidimensionality assumption, even if unidimensionality and local independence do not imply one another (Sijtsma and Molenaar 2002). As a consequence, local independence implies that a strong redundancy among the items does not exist.

Monotonicity is notably a fundamental assumption that allows validating the score as an ordinal measure of the latent trait.

2.3 Traces of the items

Traces of items can be used to check the monotonicity assumption. We define the score for each individual as the sum of the individual's responses ($S_n = \sum_{j=1}^{J} X_{nj}$). This score is assumed to represent an ordinal measure of the latent trait. The trace of a dichotomous item represents the proportion of positive responses $\{P(X_j = 1)\}$ as a function of the score. If the monotonicity assumption is satisfied, the trace lines increase. This means that the higher the latent trait, the more frequent the positive responses. In education sciences, if we wish to measure a given ability, this means that a good student will have more correct responses to the items. In health sciences, if we seek to measure a dysfunction through the presence of symptoms, this means that a patient having a high level of dysfunction will display more symptoms. An example trace is given in figure 1.

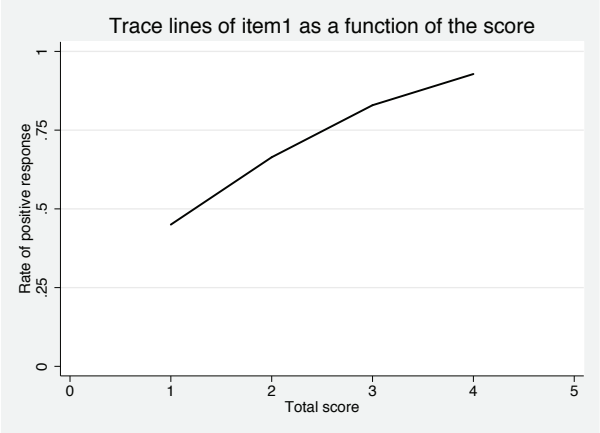


Figure 1. Trace of a dichotomous item as a function of the score

The score and the proportion of positive responses to each item are generally positively correlated, because the score is a function of all the items. This phenomenon can be strong, notably if there are few items in the questionnaire. To avoid the phenomenon, the rest-score (computed as the score of all the other items) is more generally used.

For polytomous items, we represent the proportion of responses to each response category $\{P(X_j = r)\}$ as a function of the score or of the rest-score (an example is given in figure 2).

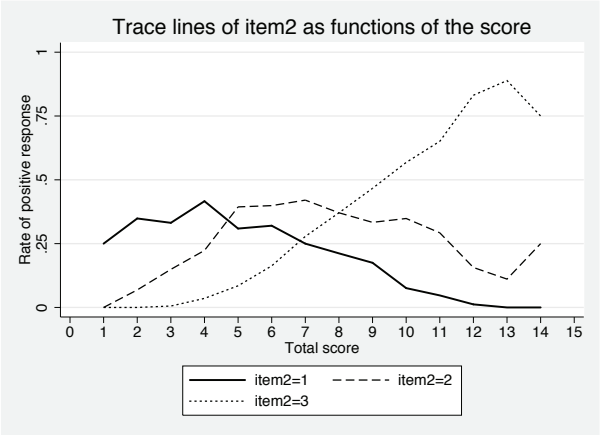


Figure 2. Traces of a polytomous item as functions of the score

Unfortunately, these trace lines are difficult to interpret, because an individual with a moderate score will preferably respond to medium response categories, and an individual with high scores will respond to high response categories, so the trace lines corresponding to each response category do not increase. Cumulative trace lines represent the proportions $P(Y_{jr} = 1) = P(X_j \geq r)$ as a function of the score or of the rest-score. If the monotonicity assumption is satisfied, these trace lines increase. An example is given in figure 3.

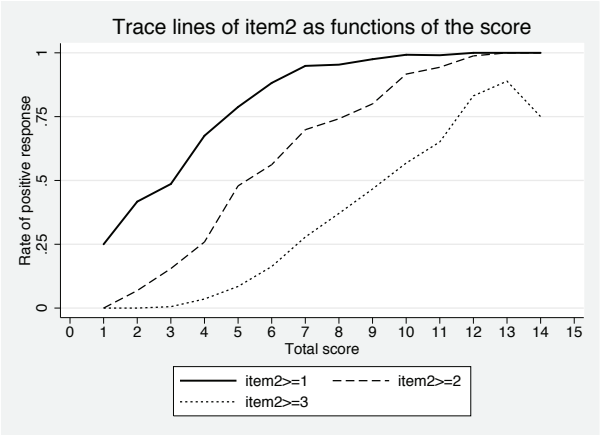


Figure 3. Cumulative trace lines of a polytomous item as functions of the score

2.4 The Guttman errors

Dichotomous case

The difficulty of an item can be defined as its proportion of negative responses. The Guttman errors (Guttman 1944) for a pair of dichotomous items are the number of individuals having a positive response to the more difficult item and a negative response to the easiest item. In education sciences, this represents the number of individuals who correctly responded to a given item but incorrectly responded to an easier item. In health sciences, this represents the number of individuals who present a given symptom but do not present a more common symptom.

We define the two-way tables of frequency counts between the items j and k as

		Item j		
		0	1	
Item k	0	a_{jk}	b_{jk}	$a_{jk}+b_{jk}$
	1	c_{jk}	d_{jk}	$c_{jk}+d_{jk}$
		$a_{jk}+c_{jk}$	$b_{jk}+d_{jk}$	N_{jk}

N_{jk} is the number of individuals with nonmissing responses to the items j and k.

An item j is easier than the item k if $P(X_j = 1) > P(X_k = 1)$—that is to say, if $(b_{jk} + d_{jk}/N_{jk}) > (c_{jk} + d_{jk}/N_{jk})$ (equivalently, if $b_{jk} > c_{jk}$), and the number of Guttman errors e_{jk} in this case is $e_{jk} = N_{jk} \times P(X_j = 0, X_k = 1) = c_{jk}$. More generally, if we ignore the easier item between j and k,

$$e_{jk} = N_{jk} \times \min\{P(X_j = 0, X_k = 1), P(X_j = 1, X_k = 0)\} = \min(b_{jk}, c_{jk}) \quad (1)$$

$e_{jk}^{(0)}$ is the number of Guttman errors under the assumption of independence of the responses to the two items:

$$\begin{aligned} e_{jk}^{(0)} =& N_{jk} \times \min\{P(X_j = 0) \times P(X_k = 1), P(X_j = 1) \times P(X_k = 0)\} \\ =& \frac{(a_{jk} + e_{jk})(e_{jk} + d_{jk})}{N_{jk}} \end{aligned}$$

Polytomous case

The Guttman errors between two given response categories r and s of the pair of polytomous items j and k are defined as

$$\begin{aligned} e_{j(r)k(s)} &= N_{jk} \times \min\{P(X_j \geq r, X_k < s), P(X_j < r, X_k \geq s)\} \\ &= N_{jk} \times \min\{P(Y_{jr} = 1, Y_{ks} = 0), P(Y_{jr} = 0, Y_{ks} = 1)\} \end{aligned}$$

The number of Guttman errors between the two items is

$$e_{jk} = \sum_{r=1}^{m_j} \sum_{s=1}^{m_k} e_{j(r)k(s)}$$

If $m_j = m_k = 1$ (the dichotomous case), this formula is equivalent to (1).

Under the assumption of independence between the responses to these two items, we have

$$e^{(0)}_{j(r)k(s)} = N_{jk} \times P(X_j < r)P(X_k \geq s) = N_{jk} \times P(Y_{jr} = 0)P(Y_{ks} = 1)$$

if $P(X_j \geq r) > P(X_k \geq s)$ and

$$e^{(0)}_{jk} = \sum_{r=1}^{m_j} \sum_{s=1}^{m_k} e^{(0)}_{j(r)k(s)}$$

2.5 The Loevinger's H coefficients

Loevinger (1948) proposed three indexes that can be defined as functions of the Guttman errors between the items.

The Loevinger's H coefficient between two items

H_{jk} is the Loevinger's H coefficient between the items j and k:

$$H_{jk} = 1 - \frac{e_{jk}}{e^{(0)}_{jk}}$$

We have $H_{jk} \leq 1$ with $H_{jk} = 1$ only if there is no Guttman error between the items j and k. If this coefficient is close to 1, there are few Guttman errors, and so the two items probably measure the same latent trait. An index close to 0 signifies that the responses to the two items are independent, and therefore reveals that the two items probably do not measure the same latent trait. A significantly negative value to this index is not expected, and it can be a flag that one or more items have been incorrectly coded or are incorrectly understood by the respondents.

We can test H_0: $H_{jk} = 0$ (against H_a: $H_{jk} > 0$). Under the null hypothesis, the statistic

$$Z = \frac{\mathrm{Cov}(X_j, X_k)}{\sqrt{\frac{\mathrm{Var}(X_j)\mathrm{Var}(X_k)}{N_{jk}-1}}} = \rho_{jk}\sqrt{N_{jk} - 1} \tag{2}$$

follows a standard normal distribution, where ρ_{jk} is the correlation coefficient between items j and k.

The Loevinger's H coefficient measuring the consistency of an item within a scale

Let S be a set of items (a scale), and let j be an item that belongs to this scale ($j \in S$). H_j^S is the Loevinger's H coefficient that measures the consistency of the item j within a scale S.

$$H_j^S = 1 - \frac{e_j^S}{e_j^{S(0)}} = 1 - \frac{\sum_{k\in S,\ k\neq j} e_{jk}}{\sum_{k\in S,\ k\neq j} e_{jk}^{(0)}}$$

If the scale S is a good scale (that is, if it satisfies an MHMM, for example), this index is close to 1 if the item j has a good consistency within the scale S, and this index is close to 0 if it has a bad consistency within this scale.

It is possible to test H_0: $H_j^S = 0$ (against H_a: $H_j^S > 0$). Under the null hypothesis, the statistic

$$Z = \frac{\sum_{k\in S, k\neq j} \mathrm{Cov}(X_j, X_k)}{\sqrt{\sum_{k\in S, k\neq j} \frac{\mathrm{Var}(X_j)\mathrm{Var}(X_k)}{N_{jk}-1}}} \tag{3}$$

follows a standard normal distribution.

The Loevinger's H coefficient of scalability

If S is a set of items, we can compute the Loevinger's H coefficient of scalability of this scale.

$$H^S = 1 - \frac{e^S}{e^{S(0)}} = 1 - \frac{\sum_{j\in S}\sum_{k\in S,\ k>j} e_{jk}}{\sum_{j\in S}\sum_{k\in S,\ k>j} e_{jk}^{(0)}}$$

We have $H^S \geq \min_{j\in S} H_j^S$. If H^S is near 1, then the scale S has good scale properties; if H^S is near 0, then it has bad scale properties.

It is possible to test H_0: $H^S = 0$ (against H_a: $H^S > 0$). Under the null hypothesis, the statistic

$$Z = \frac{\sum_{j\in S}\sum_{k\in S, k\neq j} \mathrm{Cov}(X_j, X_k)}{\sqrt{\sum_{j\in S}\sum_{k\in S, k\neq j} \frac{\mathrm{Var}(X_j)\mathrm{Var}(X_k)}{N_{jk}-1}}} \tag{4}$$

follows a standard normal distribution.

In the MSP software (Molenaar, Sijtsma, and Boer 2000), the z statistics defined in (2), (3), and (4) are approximated by dividing the variances by N_{jk} instead of by $N_{jk} - 1$.

2.6 The fit of a Mokken scale to a dataset

Link between the Loevinger's H coefficient and the Mokken scales

Mokken (1971) showed that if a scale S is a Mokken scale, then $H^S > 0$, but the converse is not true. He proposes the following classification:

- If $H^S < 0.3$, the scale S has poor scalability properties.
- If $0.3 \leq H^S < 0.4$, the scale S is "weak".
- If $0.4 \leq H^S < 0.5$, the scale S is "medium".
- If $0.5 \leq H^S$, the scale S is "strong".

So Mokken (1971) suggests using the Loevinger's H coefficient to build scales that satisfy a Mokken scale. He suggests that there is a threshold $c > 0.3$ such that if $H^S > c$, then the scale S satisfies a Mokken scale. This idea is used by Mokken (1971) and is adapted by Hemker, Sijtsma, and Molenaar (1995) to propose the MSP or automated item selection procedure (AISP) (Sijtsma and Molenaar 2002).

Moreover, the fit of the Mokken scale is satisfactory if $H_j^S > c$ and $H_{jk} > 0$ for all pairs of items j and k from the scale S.

Check of the monotonicity assumption

The monotonicity assumption can be checked by a visual inspection of the trace lines. Nevertheless, the MSP program that Molenaar, Sijtsma, and Boer (2000) proposed calculates indexes to evaluate the monotonicity assumption. The idea of these indexes is to allow the trace lines to have small decreases.

To check for the monotonicity assumption linked to the jth item ($j = 1, \ldots, J$), the population is cut into G_j groups (based on the individual's rest-score for item j as the sum of the individual's responses to the other items). Each group is indexed by $g = 1, \ldots, G_j$ ($g = 1$ represents the individuals with the lower rest-scores, and $g = G_j$ represents the individuals with the larger rest-scores).

Let Z_j be the random variable representing the groups corresponding to the jth item. It is expected that $\forall j = 1, \ldots, J$ and $r = 1, \ldots, m_j$. We have $P(Y_{jr} = 1|Z_j = g) \geq P(Y_{jr} = 1|Z_j = g')$ with $g > g'$. $G_j(G_j - 1)/2$ of such comparisons can be realized for the item j (denoted as $\#\text{ac}_j$ for active comparisons). In fact, only important violations of the expected results are retained, and a threshold minimum violation (minvi) is used to define an important violation $P(Y_{jr} = 1|Z_j = g') - P(Y_{jr} = 1|Z_j = g) >$ minvi. Consequently, it is possible for each item to count the number of important violations ($\#\text{vi}_j$) and to compute the value of the maximum violation (maxvi_j) and the sum of the important violations (sum_j). Lastly, it is possible to test the null hypothesis $H_0 : P(Y_{jr} = 1|Z_j = g) \geq P(Y_{jr} = 1|Z_j = g')$ against the alternative hypothesis $H_a : P(Y_{jr} = 1|Z_j = g) < P(Y_{jr} = 1|Z_j = g')$ $\forall j, r, g, g'$ with $g > g'$.

Consider the table

		Item	Y_{jr}
		0	1
Group	g'	a	b
	g	c	d

Under the null hypothesis, the statistic

$$z = \frac{2\left\{\sqrt{(a+1)(d+1)} - \sqrt{bc}\right\}}{\sqrt{a+b+c+d-1}}$$

follows a standard normal distribution. The maximal value of z for the item j is denoted $z\text{max}_j$, and the number of significant z values is denoted $\#z\text{sig}_j$. The criterion used to check the monotonicity assumption linked to the item j is defined by Molenaar, Sijtsma, and Boer (2000) as

$$\begin{aligned}\text{Crit}_j &= 50(0.30 - H_j) + \sqrt{\#\text{vi}_j} + 100\frac{\#\text{vi}_j}{\#\text{ac}_j} + 100\text{maxvi}_j + 10\sqrt{\text{sum}_j} + 1000\frac{\text{sum}_j}{\#\text{ac}_j} \\ &\quad +5z\text{max}_j + 10\sqrt{\#z\text{sig}_j} + 100\frac{\#z\text{sig}_j}{\#\text{ac}_j} \end{aligned} \quad (5)$$

It is generally considered that a criterion less than 40 signifies that the reported violations can be ascribed to sampling variation. A criterion exceeding 80 casts serious doubts on the monotonicity assumption for this item. If the criterion is between 40 and 80, further analysis must be considered to draw a conclusion (Molenaar, Sijtsma, and Boer 2000).

2.7 The doubly monotonely homogeneous model of Mokken (DMHMM)

The P++ and P−− matrices

The DMHMM is a model where the probabilities $P(X_j \geq l)\ \forall j, l$ produce the same ranking of items for all persons (Mokken and Lewis 1982). In practice, this means that the questionnaire is interpreted similarly by all the individuals, whatever their level of the latent trait.

$\mathbf{P}++$ is an $\mathbf{M} \times \mathbf{M}$ matrix in which each element corresponds to the probability $P(X_j \geq r, X_k \geq s) = P(Y_{jr} = 1, Y_{ks} = 1)$. The rows and the columns of this matrix are ordered from the most difficult threshold item $Y_{jr}\ \forall j, r$ to the easiest one.

$\mathbf{P}--$ is an $\mathbf{M} \times \mathbf{M}$ matrix in which each element corresponds to the probability $P(Y_{jr} = 0, Y_{ks} = 0)$. The rows and the columns of this matrix are ordered from the most difficult threshold item $Y_{jr}\ \forall j, r$ to the easiest one.

A set of items satisfies the doubly monotone assumption if the set satisfies an MHMM, and if the elements of the $\mathbf{P}++$ matrix are increasing in each row and the elements of the $\mathbf{P}--$ matrix are decreasing in each row.

We can represent each column of these matrices in a graph. On the x axis, the response categories are ordered in the same order as in the matrices; and on the y axis, the probabilities contained in the matrices are represented. The obtained curves must be nondecreasing for the $\mathbf{P}++$ matrix and must be nonincreasing for the $\mathbf{P}--$ matrix.

Check of the double monotonicity assumption via the analysis of the P matrices

Consider three threshold items Y_{jr}, Y_{ks}, and Y_{lt} with $j \neq k \neq l$. Under the DMHMM, if $P(Y_{ks} = 1) < P(Y_{lt} = 1)$, then it is expected that $P(Y_{ks} = 1, Y_{jr} = 1) < P(Y_{lt} = 1, Y_{jr} = 1)$. In the set of possible threshold items, we count the number of important violations of this principle among all the possible combinations of three items. An important violation represents a case where $P(Y_{ks} = 1, Y_{jr} = 1) - P(Y_{lt} = 1, Y_{jr} = 1) >$ minvi, where minvi is a fixed threshold. For each item j, $j = 1, \ldots, J$, we count the number of comparisons ($\#\text{ac}_j$), the number of important violations ($\#\text{vi}_j$), the value of maximal important violation (maxvi_j), and the sum of the important violations (sumvi_j). It is possible to test the null hypothesis H_0: $P(Y_{ks} = 1, Y_{jr} = 1) \leq P(Y_{lt} = 1, Y_{jr} = 1)$ against the alternative hypothesis H_a: $P(Y_{ks} = 1, Y_{jr} = 1) > P(Y_{lt} = 1, Y_{jr} = 1)$ with a McNemar test.

Let K be the random variable representing the number of individuals in the sample who satisfy $Y_{jr} = 1$, $Y_{ks} = 0$, and $Y_{lt} = 1$. Let N be the random variable representing the number of individuals in the sample who satisfy $Y_{jr} = 1$, $Y_{ks} = 0$, and $Y_{lt} = 1$, or who satisfy $Y_{jr} = 1$, $Y_{ks} = 1$, and $Y_{lt} = 0$. k and n are the realizations of these two random variables. Molenaar, Sijtsma, and Boer (2000) define the statistic:

$$z = \sqrt{2k + 2 + b} - \sqrt{2n - 2k + b} \quad \text{with} \quad b = \frac{(2k + 1 - n)^2 - 10n}{12n}$$

Under the null hypothesis, z follows a standard normal distribution. It is possible to count the number of significant tests ($\#z\text{sig}$) and the maximal value of the z statistics ($z\text{max}$).

A criterion can be computed for each item as the one used in (5), using the same thresholds for checking the double monotonicity assumption.

2.8 Contribution of each individual to the Guttman errors, H coefficients, and person-fit

From the preceding formulas, the number of Guttman errors induced by each individual can be computed. Let e_n be this number for the nth individual. The number of expected Guttman errors under the assumption of independence of the responses to the item is equal to $e_n^{(0)} = e^{S(0)}/N$. An individual with $e_n > e_n^{(0)}$ is very likely to be an individual whose responses are not influenced by the latent variable, and if e_n is very high, the individual can be considered an outlier.

By analogy with the Loevinger coefficient, we can compute the H_n coefficient in the following way: $H_n = 1 - (e_n/e_n^{(0)})$. A large negative value indicates an outlier, and a positive value is expected (note that $H_n \leq 1$).

It is interesting to note that when there is no missing value,

$$H^S = \frac{\sum_{n=1}^{N} H_n}{N}$$

Emons (2008) defines the normalized number of Guttman errors for polytomous items (G^p_N) as

$$G^p_{Nn} = \frac{e_n}{e_{\max,n}}$$

where $e_{\max,n}$ is the maximal number of Guttman errors obtained with a score equal to S_n. This index can be interpreted as

- $0 \leq G^p_{Nn} \leq 1$
- if G^p_{Nn} is close to 0, the individual n has few Guttman errors
- if G^p_{Nn} is close to 1, the individual n has many Guttman errors

The advantages of the G^p_{Nn} indexes are that they always lie between 0 and 1, inclusive, regardless of the number of items and response categories and that dividing by $e_{\max,n}$ adjusts the index to the observed score S_n. However, there is no threshold standard to use to judge the closeness of the index to 0 or 1.

2.9 MSP or AISP

Algorithm

The MSP proposed by Hemker, Sijtsma, and Molenaar (1995) allows selecting items from a bank of items that satisfy a Mokken scale. This procedure uses Mokken's definition of a scale (Mokken 1971): $H_{jk} > 0$, $H^S_j > c$, and $H^S > c$, for all pairs of items j and k from the scale S.

At the initial step, a kernel of at least two items is chosen (we can select, for example, the pair of items having the maximal significant H_{jk} coefficient). This kernel corresponds to the scale S^0.

At each step $n \geq 1$, we integrate into the scale $S^{(n-1)}$ the item j if that item satisfies these conditions:

- $j \notin S^{(n-1)}$
- $S^{(n)} \equiv S^{(n-1)} \cup j$
- $j = \arg\max_{k \notin S^{(n-1)}} H^{S^{*(n)}}$ with $S^{*(n)} \equiv S^{(n-1)} \cup k$
- $H^{S^{(n)}} \geq c$
- $H^{S^{(n)}}_j \geq c$
- $H^{S^{(n)}}_j$ is significantly positive
- H_{jk} is significantly positive $\forall k \in S^{(n-1)}$

The MSP is stopped as soon as no item satisfies all these conditions, but it is possible to construct a second scale with the items not selected in the first scale, and so on, until there are no more items remaining.

The threshold c is subjectively defined by the user: the authors of this article recommend fixing $c \geq 0.3$. As c gets larger, the obtained scale will become stronger, but it will be more difficult to include an item in a scale.

The Bonferroni corrections

At the initial step, in the general case, we compare all the possible H_{jk} coefficients to 0 using a test: there are $J(J-1)/2$ such tests. At each following step l, we compare $J^{(l)}$ H_j coefficients with 0, where $J^{(l)}$ is the number of unselected items at the beginning of step l.

Bonferroni corrections are used to take into account this number of tests and to keep a global level of significance equal to α (Molenaar, Sijtsma, and Boer 2000). At the initial step, we divide α by $J(J-1)/2$ to obtain the level of significance; and at each step l, we divide α by $\{J(J-1)/2\} + \sum_{m=1}^{l} J^{(m)}$.

When the initial kernel is composed of only one item, only $J-1$ tests are realized at the first step, and the coefficient $J(J-1)/2$ is replaced by $J-1$. When the initial kernel is composed of at least two items, this coefficient is replaced by 1.

Tip for improving the speed of computing

At each step, the items k (unselected in the current scale) that satisfy $H_{jk} < 0$ with an item j already selected in the current scale are automatically excluded.

3 Stata commands

In this section, we present three Stata commands for calculating the indexes and algorithms presented in this article. These commands have been intensively tested and compared with the output of the MSP software with several datasets. Small (and generally irrelevant) differences from the MSP software can persist and can be explained by different ways of approximating the values.

3.1 The tracelines command

Syntax

The syntax of the `tracelines` command (version 3.2 is described here) is

`tracelines` *varlist* [`, score restscore ci test cumulative logistic`
`repfiles(`*directory*`) scorefiles(`*string*`) restscorefiles(`*string*`)`
`logisticfile(`*string*`) nodraw nodrawcomb replace onlyone(`*varname*`)`
`thresholds(`*string*`)`]

Options

`score` displays graphical representations of trace lines of items as functions of the total score. This is the default if neither `restscore` nor `logistic` is specified.

`restscore` displays graphical representations of trace lines of items as functions of the rest-score (total score without the item).

`ci` displays the confidence interval at 95% of the trace lines.

`test` tests the null hypothesis that the slope of a linear model for the trace line is zero.

`cumulative` displays cumulative trace lines for polytomous items instead of classical trace lines.

`logistic` displays graphical representations of logistic trace lines of items as functions of the score: each trace comes from a logistic regression of the item response on the score. This kind of trace is possible only for dichotomous items. All the logistic trace lines are represented in the same graph.

`repfiles(`*directory*`)` specifies the directory where the files should be saved.

`scorefiles(`*string*`)` defines the generic name of files containing graphical representations of trace lines as functions of the score. The name will be followed by the name of each item and by the `.gph` extension. If this option is not specified, the corresponding graphs will not be saved.

`restscorefiles(`*string*`)` defines the generic name of files containing graphical representations of trace lines as functions of the rest-scores. The name will be followed by the name of each item and by the `.gph` extension. If this option is not specified, the corresponding graphs will not be saved.

`logisticfile(`*string*`)` defines the name of the file containing graphical representations of logistic trace lines. This name will be followed by the `.gph` extension. If this option is not specified, the corresponding graph will not be saved.

`nodraw` suppresses the display of graphs for individual items.

`nodrawcomb` suppresses the display of combined graphs but not of individual items.

`replace` replaces graphical files that already exist.

`onlyone(`*varname*`)` displays only the trace of a given item.

`thresholds(`*string*`)` groups individuals as a function of the score or the rest-score. The *string* contains the maximal values of the score or the rest-score in each group.

3.2 The loevh command

Syntax

The syntax of the `loevh` command (version 7.1 is described here) is

`loevh` *varlist* [`, pairwise pair ppp pmm noadjust generror(`*newvar*`) replace graph monotonicity(`*string*`) nipmatrix(`*string*`)`]

`loevh` requires that the commands `tracelines`, `anaoption`, `gengroup`, `guttmax`, and `genscore` be installed.

Options

`pairwise` omits, for each pair of items, only the individuals with a missing value on these two items. By default, `loevh` omits all individuals with at least one missing value in the items of the scale.

`pair` displays the values of the Loevinger's H coefficients and the associated statistics for each pair of items.

`ppp` displays the $\mathbf{P}++$ matrix (and the associated graph with `graph`).

`pmm` displays the $\mathbf{P}--$ matrix (and the associated graph with `graph`).

`noadjust` uses N_{jk} as the denominator instead of the default, $N_{jk}-1$, when calculating test statistics. The MSP software also uses N_{jk}.

`generror(`*newvar*`)` defines the prefix of five new variables. The first new variable (only the prefix) will contain the number of Guttman errors attached to each individual; the second one (the prefix followed by `_0`), the number of Guttman errors attached to each individual under the assumption of independence of the items; the third one (the prefix followed by `_H`), the quantity 1 minus the ratio between the two preceding values; the fourth one (the prefix followed by `_max`), the maximal possible Guttman errors corresponding to the score of the individual; and the last one (the prefix followed by `_GPN`), the normalized number of Guttman errors. With the `graph` option, a histogram of the number of Guttman errors by individual is drawn.

`replace` replaces the variables defined by the `generror()` option.

`graph` displays graphs with the `ppp`, `pmm`, and `generror()` options. This option is automatically disabled if the number of possible scores is greater than 20.

`monotonicity(`*string*`)` displays indexes to check monotonicity of the data (MHMM). This option produces output similar to that of the MSP software. The *string* contains the following suboptions: `minvi()`, `minsize()`, `siglevel()`, and `details`. If you want to use all the default values, type `monotonicity(*)`.

`minvi(`#`)` defines the minimal size of a violation of monotonicity. The default is `monotonicity(minvi(0.03))`.

`minsize(`#`)` defines the minimal size of groups of patients to check the monotonicity (by default, this value is equal to $N/10$ if $N > 500$, to $N/5$ if $250 < N \leq 500$, and to $N/3$ if $N \leq 250$ with the minimal group size fixed at 50).

`siglevel(`#`)` defines the significance level for the tests. The default is `monotonicity(siglevel(0.05))`.

`details` displays more details with polytomous items.

`nipmatrix(`*string*`)` displays indexes to check the nonintersection (DMHMM). This option produces output similar to that of the MSP software. The *string* contains two suboptions: `minvi()` and `siglevel()`. If you want to use all the default values, type `nipmatrix(*)`.

`minvi(`#`)` defines the minimal size of a violation of nonintersection. The default is `nipmatrix(minvi(0.03))`.

`siglevel(`#`)` defines the significance level for the tests. The default is `nipmatrix(siglevel(0.05))`.

Saved results

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

Scalars

`r(pvalH)`	p-value for Loevinger's H coefficient of scalability
`r(zH)`	z statistic for Loevinger's H coefficient of scalability
`r(eGutt0)`	total number of theoretical Guttman errors associated with the scale
`r(eGutt)`	total number of observed Guttman errors associated with the scale
`r(loevh)`	Loevinger's H coefficient of scalability

Matrices

`r(Obs)`	(matrix) number of individuals used to compute each coefficient H_{jk} (if the `pairwise` option is not used, the number of individuals is the same for each pair of items)
`r(pvalHj)`	p-values for consistency of each item with the scale
`r(pvalHjk)`	p-values for pairs of items
`r(zHj)`	z statistics for consistency of each item with the scale
`r(zHjk)`	z statistics for pairs of items
`r(P11)`	$\mathbf{P}++$ matrix
`r(P00)`	$\mathbf{P}--$ matrix
`r(eGuttjk0)`	theoretical Guttman errors associated with each item pair
`r(eGuttj0)`	theoretical Guttman errors associated with the scale
`r(eGuttjk)`	observed Guttman errors associated with each item pair
`r(eGuttj)`	observed Guttman errors associated with the scale
`r(loevHjk)`	Loevinger's H coefficients for pairs of items
`r(loevHj)`	Loevinger's H coefficients for consistency of each item with the scale

3.3 The msp command

Syntax

The syntax of the `msp` command (version 6.6 is described here) is

`msp` *varlist* [`, c(#) kernel(#) p(#) minvalue(#) pairwise nobon notest nodetails noadjust`]

`msp` requires that the `loevh` command be installed.

Options

`c(#)` defines the value of the threshold c. The default is `c(0.3)`.

`kernel(#)` defines the first # items as the kernel of the first subscale. The default is `kernel(0)`.

`p(#)` defines the level of significance of the tests. The default is `p(0.05)`.

`minvalue(#)` defines the minimum value of an H_{jk} coefficient between two items j and k on a same scale. The default is `minvalue(0)`.

`pairwise` omits, for each pair of items, only the individuals with a missing value on these two items. By default, `msp` omits all individuals with at least one missing value in the items of the scale.

`nobon` suppresses the Bonferroni corrections of the levels of significance.

`notest` suppresses testing of the nullity of the Loevinger's H coefficient.

`nodetails` suppresses display of the details of the algorithm.

`noadjust` uses N_{jk} as the denominator instead of the default, $N_{jk}-1$, when calculating test statistics. The MSP software also uses N_{jk}.

Saved results

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

Scalars

`r(dim)`	number of created scales
`r(nbitems#)`	number of selected items in the #th scale
`r(H#)`	value of the Loevinger's H coefficient of scalability for the #th scale

Macros

`r(lastitem)`	when only one item is remaining, the name of that item
`r(scale#)`	list of the items selected in the #th scale (in the order of selection)

Matrices

`r(selection)`	a vector that contains, for each item, the scale where it is selected (or `0` if the item is unselected)

3.4 Output

We present an example of output of these programs with items of the French adaptation of the Ways of Coping Checklist questionnaire (Cousson et al. 1996). This questionnaire measures coping strategies and includes 27 items that compose three dimensions: problem-focused coping, emotional coping, and seeking social support. The sample is composed of 100 women, each with a recent diagnosis of breast cancer.

Output of the loevh command

The `loevh` command allows researchers to obtain the values of the Loevinger's H coefficients. Because the sample was small, it was impossible to obtain several groups of 50 individuals or more. As a consequence, for the `monotonicity()` option, the `minsize()` has been fixed at 30. We studied the emotional dimension composed of nine items (with four response categories per item). The rate of missing data varied from 2% to 15% per item. Only 69 women have a complete pattern of responses, so the `pairwise` option was employed to retain a maximum of information.

```
. use wccemo
. loevh item2 item5 item8 item11 item14 item17 item20 item23 item26, pairwise
> monotonicity(minsize(30)) nipmatrix(*)
                        Observed  Expected                                  Number
           Difficulty   Guttman   Guttman  Loevinger           HO: Hj<=0    of NS
Item   Obs    P(Xj=0)    errors    errors    H coeff   z-stat.   p-value     Hjk
-------------------------------------------------------------------------------
item2   92     0.2935       453    732.03    0.38117   7.4874    0.00000       1
item5   92     0.3261       395    751.61    0.47446   9.5492    0.00000       1
item8   90     0.3667       515    788.65    0.34699   7.6200    0.00000       4
item11  97     0.5670       519    862.50    0.39826   9.2705    0.00000       1
item14  98     0.6327       532    752.63    0.29314   6.8306    0.00000       3
item17  94     0.7660       299    487.40    0.38653   7.4598    0.00000       1
item20  95     0.6632       494    711.53    0.30573   6.7867    0.00000       1
item23  85     0.5412       525    729.72    0.28054   6.1752    0.00000       2
item26  89     0.6517       502    710.59    0.29355   6.3643    0.00000       2
-------------------------------------------------------------------------------
Scale  100                 2117   3263.33    0.35128  15.9008    0.00000

        Summary per item for check of monotonicity
        Minvi=0.030    Minsize=   30  Alpha=0.050

Items  #ac  #vi  #vi/#ac   maxvi     sum  sum/#ac    zmax #zsig  Crit
-------------------------------------------------------------------------------
item2    3    0                                                  -4     graph
item5    3    0                                                  -9     graph
item8    3    0                                                  -2     graph
item11   3    0                                                  -5     graph
item14   3    0                                                   0     graph
item17   2    0                                                  -4     graph
item20   3    0                                                  -0     graph
item23   3    0                                                   1     graph
item26   3    0                                                   0     graph
-------------------------------------------------------------------------------
Total   52    0   0.0000  0.0000  0.0000   0.0000  0.0000           0
-------------------------------------------------------------------------------
```

```
          Summary per item for check of non-Intersection via Pmatrix
          Minvi=0.030    Alpha=0.050
Items    #ac    #vi    #vi/#ac    maxvi      sum    sum/#ac    zmax    #zsig  Crit
item2   1512     49     0.0324   0.0990   2.2005     0.0015  1.6844        1    51
item5   1512     85     0.0562   0.1239   4.1743     0.0028  2.9280        6    81
item8   1512     90     0.0595   0.1105   4.2927     0.0028  2.5221        4    81
item11  1512    120     0.0794   0.1105   5.4429     0.0036  2.5221        6    89
item14  1512     88     0.0582   0.1081   4.1701     0.0028  2.3015        7    88
item17  1512     52     0.0344   0.0865   2.4122     0.0016  2.0662        2    57
item20  1512     52     0.0344   0.0830   2.2127     0.0015  2.3015        1    57
item23  1512     90     0.0595   0.0990   4.2123     0.0028  1.8742        3    77
item26  1512     94     0.0622   0.1239   4.3258     0.0029  2.9280        4    87
```

This scale has a satisfactory scalability ($H^S = 0.35$). Three items (14, 23, 26) display a borderline value for the H_j^S coefficient (0.28 or 0.29). The monotonicity assumption is not rejected because no important violation of this assumption occurred and the criteria are satisfied. This is not the case for the nonintersection of the Pmatrix curves: several criteria are greater than 80 (items 5, 8, 11, 14, 23, 26), showing an important violation of this assumption. The model followed by these data is therefore more an MHMM than a DMHMM. Because the indexes suggest that the MHMM is appropriate, the score computed by summing codes associated with the nine items can be considered a correct ordinal measure of the studied latent trait (the emotional coping), and the three fundamental assumptions of IRT (unidimensionality, local independence, and monotonicity) can be considered verified.

Output of the msp command

The `msp` command runs the Mokken scale procedure.

```
. msp item2 item5 item8 item11 item14 item17 item20 item23 item26, pairwise
Scale: 1
```

```
Significance level: 0.001389
The two first items selected in the scale 1 are item2 and item11 (Hjk=0.6245)
The following items are excluded at this step: item14 item23
Significance level: 0.001220
The item item17 is selected in the scale 1       Hj=0.5304      H=0.5748
The following items are excluded at this step: item8
Significance level: 0.001136
The item item5 is selected in the scale 1        Hj=0.5464      H=0.5588
The following items are excluded at this step: item26
Significance level: 0.001111
The item item20 is selected in the scale 1       Hj=0.3758      H=0.4864
Significance level: 0.001111
There is no more items remaining.
```

```
                        Observed  Expected                                 Number
            Difficulty   Guttman   Guttman   Loevinger          H0: Hj<=0   of NS
Item   Obs    P(Xj=0)     errors    errors     H coeff  z-stat.  p-value     Hjk
-------------------------------------------------------------------------------
item20  95     0.6632        212    339.64     0.37582   5.5460  0.00000       0
item5   92     0.3261        179    376.71     0.52484   7.5735  0.00000       0
item17  94     0.7660        124    233.07     0.46797   5.8889  0.00000       0
item2   92     0.2935        186    367.75     0.49422   6.9525  0.00000       0
item11  97     0.5670        181    400.10     0.54761   8.4434  0.00000       0
-------------------------------------------------------------------------------
Scale  100                   441    858.64     0.48640  10.9364  0.00000
Scale: 2
-------

Significance level: 0.008333
The two first items selected in the scale 2 are item23 and item26 (Hjk=0.4391)
The following items are excluded at this step: item8
Significance level: 0.007143
The item item14 is selected in the scale 2       Hj=0.4276      H=0.4313
Significance level: 0.007143
There is no more items remaining.
                        Observed  Expected                                 Number
            Difficulty   Guttman   Guttman  Loevinger          H0: Hj<=0   of NS
Item   Obs    P(Xj=0)     errors    errors    H coeff   z-stat.  p-value     Hjk
-------------------------------------------------------------------------------
item14  98     0.6327        115    200.89     0.42756   5.4739  0.00000       0
item23  85     0.5412        109    193.44     0.43651   5.2885  0.00000       0
item26  89     0.6517        114    200.00     0.43000   5.4109  0.00000       0
-------------------------------------------------------------------------------
Scale  100                   169    297.17     0.43129   6.5985  0.00000

There is only one item remaining (item8).
```

The AISP creates two groups of items.

On the one hand, five items measure negation or the wish to forget the reason for the stress: `item2`, "Wish that the situation would go away or somehow be over with"; `item5`, "Wish that I can change what is happening or how I feel"; `item11`, "Hope a miracle will happen"; `item17`, "I daydream or imagine a better time or place than the one I am in"; and `item20`, "Try to forget the whole thing". For this set, the scalability coefficient is good (0.49), and there is no problem concerning the monotonicity assumption (maximal criterion per item of -4), nor is there a problem concerning the intersection of the curves (maximal criterion per item of 38). This set seems to satisfy a DMHMM and is composed of 5 of the 11 items composing the "wishful thinking" and "detachment" dimensions proposed by Folkman and Lazarus (1985) in an analysis of the Ways of Coping Checklist questionnaire among a sample of students.

On the other hand, three items measure culpability: `item14`, "Realize I brought the problem on myself"; `item23`, "Make a promise to myself that things will be different next time"; and `item26`, "Criticize or lecture myself". For this set, the scalability coefficient is good (0.43), and there is no problem concerning the monotonicity assumption (maximal criterion per item of -6), nor concerning the intersection of the curves (maximal criterion per item of -6). This set seems to satisfy a DMHMM and is composed of the three items of the "self blame" dimension proposed by Folkman and Lazarus (1985).

In our case, it is possible to choose between a set of items that satisfy an MHMM and two sets of items that each satisfy a DMHMM. Because the three sets of items are interpretable (emotional coping for the set of items satisfying MHMM; negation and culpability for the two other sets of items), there is no problem to choose freely from the available types of measured concepts. Concerning the validation of the questionnaire, it is preferable to choose the set of items containing all items satisfying the emotional coping, which is closer to the output returned by the `loevh` command.

4 References

Cousson, F., M. Bruchon-Schweitzer, B. Quintard, J. Nuissier, and N. Rascle. 1996. Analyse multidimensionnelle d'une échelle de coping: validation française de la W.C.C. (way of coping checklist). *Psychologie Française* 41: 155–164.

Emons, W. H. 2008. Nonparametric person-fit analysis of polytomous item scores. *Applied Psychological Measurement* 32: 224–247.

Folkman, S., and R. S. Lazarus. 1985. If it changes it must be a process: Study of emotion and coping during three stages of a college examination. *Journal of Personality and Social Psychology* 48: 150–170.

Guttman, L. 1944. A basis for scaling qualitative data. *American Sociological Review* 9: 139–150.

Hardouin, J.-B. 2002. *The SAS Macro-program "%LOEVH"*. University of Nantes, http://sasloevh.anaqol.org.

———. 2007. Rasch analysis: Estimation and tests with raschtest. *Stata Journal* 7: 22–44.

Hemker, B. T., K. Sijtsma, and I. W. Molenaar. 1995. Selection of unidimensional scales from a multidimensional item bank in the polytomous Mokken IRT model. *Applied Psychological Measurement* 19: 337–352.

Hemker, B. T., K. Sijtsma, I. W. Molenaar, and B. W. Junker. 1997. Stochastic ordering using the latent trait and the sum score in polytomous IRT models. *Psychometrika* 62: 331–347.

Loevinger, J. 1948. The technic of homogeneous tests compared with some aspects of scale analysis and factor analysis. *Psychological Bulletin* 45: 507–529.

Mokken, R. J. 1971. *A Theory and Procedure of Scale Analysis: With Applications in Political Research*. Berlin: De Gruyter.

———. 1997. Nonparametric models for dichotomous responses. In *Handbook of Modern Item Response Theory*, ed. W. J. van der Linden and R. K. Hambleton, 351–368. New York: Springer.

Mokken, R. J., and C. Lewis. 1982. A nonparametric approach to the analysis of dichotomous item responses. *Applied Psychological Measurement* 6: 417–430.

Molenaar, I. W. 1997. Nonparametric models for polytomous responses. In *Handbook of Modern Item Response Theory*, ed. W. J. van der Linden and R. K. Hambleton, 369–380. New York: Springer.

Molenaar, I. W., K. Sijtsma, and P. Boer. 2000. *User's Manual for MSP5 for Windows: A Program for Mokken Scale Analysis for Polytomous Items (Version 5.0)*. University of Groningen, Groningen, The Netherlands.

Sijtsma, K., and I. W. Molenaar. 2002. *Introduction to Nonparametric Item Response Theory*. Thousand Oaks, CA: Sage.

van der Ark, L. A. 2007. Mokken scale analysis in R. *Journal of Statistical Sofware* 20: 1–19.

van der Linden, W. J., and R. K. Hambleton, ed. 1997. *Handbook of Modern Item Response Theory*. New York: Springer.

Weesie, J. 1999. mokken: Stata module: Mokken scale analysis. Statistical Software Components, Department of Economics, Boston College. http://econpapers.repec.org/software/bocbocode/sjw31.htm.

Zheng, X., and S. Rabe-Hesketh. 2007. Estimating parameters of dichotomous and ordinal item response models with gllamm. *Stata Journal* 7: 313–333.

About the authors

Jean-Benoit Hardouin and Véronique Sébille are, respectively, attached professor and full professor in biostatistics at the Faculty of Pharmaceutical Sciences of the University of Nantes. Their research applies item–response theory in clinical research. Angélique Bonnaud-Antignac is an attached professor in clinical psychology at the Faculty of Medicine of the University of Nantes. Her research deals with the evaluation of quality of life in oncology.

The Stata Journal (2011)
11, Number 1, pp. 52–63

Visualization of social networks in Stata using multidimensional scaling

Rense Corten
Department of Sociology
Interuniversity Center for Social Science Theory and Methodology
Utrecht University
The Netherlands
r.corten@uu.nl

Abstract. I describe and illustrate the use of multidimensional scaling methods for visualizing social networks in Stata. The procedure is implemented in the `netplot` command. I discuss limitations of the approach and sketch possibilities for improvement.

Keywords: gr0048, netplot, mds, social network analysis, visualization, multidimensional scaling

1 Introduction

Social network analysis (SNA) is the study of patterns of interaction between social entities (Wasserman and Faust 1994; Scott 2000). In the past few decades, SNA has emerged as a major research paradigm in the social sciences (including economics) and has also attracted attention in other fields (Newman, Barabási, and Watts 2006). While dedicated software for SNA exists (for example, UCINET [Borgatti, Everett, and Freeman 1999] or Pajek [Batagelj and Mrvar 2009]), Stata currently lacks readily available facilities for SNA. In this article, I illustrate how methods for SNA can be developed in Stata, using network visualization as an example.

Visualization is one of the oldest methods in SNA and is still one of its most important and widely applied tools for uncovering patterns of relations (Freeman 2000). I describe a procedure for network visualization using Stata's built-in procedures for multidimensional scaling (MDS) and describe an implementation as a Stata command. While I believe that network visualization in itself can be highly useful, the example also illustrates how SNA problems can be handled in Stata more generally.

2 Methods

2.1 Some terminology

Network visualization is concerned with showing binary relations between entities. Adopting the terminology of graph theory, I refer to these entities as vertices. Relations between vertices may be considered directed if they can be understood as flowing from

one vertex to another or may be considered nondirected if no such direction can be identified. I refer to directed relations as arcs and to nondirected relations as edges.

A typical representation of a network of relations is an adjacency matrix, as shown in figure 1 for a network of 10 vertices. In this matrix, every cell represents a relation from a vertex (row) to another vertex (column); for nondirected networks, this matrix is symmetrical. Vertices that have no edges or arcs are called isolates. The number of edges connected to a vertex is called the degree of the vertex. Lastly, the distance between two vertices is defined as the shortest path between them. If there is no path between two isolates, I define the distance between them as infinite.

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	0	1	0	0	0	0	0
2	1	0	0	1	0	1	0	1	0	0
3	1	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	1	0	0	0
5	1	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	1	0	0	0
7	0	0	0	1	0	1	0	0	0	0
8	0	1	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	0

Figure 1. An adjacency matrix, $N = 10$

2.2 Data structure

One particular obstacle in analyzing network data in conventional statistics packages such as Stata is the specific structure of relational data. Whereas in conventional datasets one line in the data typically represents an individual entity, observations in relational datasets represent relations between entities.

I assume that data are available as a list of edges or arcs. That is, for a network of k relations, I have a $k \times 2$ data matrix in which every row represents an edge (if the network is nondirected) or an arc (if the network is directed) between two vertices in the cells. The use of edgelists and arclists is often a more economical way to store network data than is an adjacency matrix, especially for networks that are relatively sparse.

I extend the traditional edgelist and arclist formats by allowing the use of missing values. I use missing values to include isolates in the list (figure 2). In figure 1, vertex 10 is isolated; in figure 2, its vertex number appears in one column accompanied by a missing value in the other column. The order of appearance might be reversed, thus a network consisting of k edges and N vertices, of which h isolates, can be represented by a $(k + h) \times 2$ matrix.

	col 1	col 2
1	2	1
2	3	1
3	4	2
4	5	1
5	6	2
6	7	6
7	7	4
8	8	2
9	9	8
10	10	.

Figure 2. Edgelist based on the adjacency matrix in figure 1

2.3 Procedure

The main task in network visualization is to determine the positions of the vertices in a (typically two-dimensional) graphical layout. Obviously, the optimal placement of vertices depends on the purpose of the analysis; however, it is often desirable to centrally locate in the graphic those vertices that have a central position in the SNA and to represent a larger distance in the network by a larger distance in the two-dimensional graph. Various algorithms have been proposed toward this ideal. Among them, those by Kamada and Kawai (1989) and Fruchterman and Reingold (1991) are probably most widely used. Instead, I use MDS to compute coordinates for the vertices. This strategy has the advantage of being available in Stata by default. The use of MDS for network visualization has a long history in SNA and was first used in this way by Laumann and Guttman (1966).

Assuming that I have a relational dataset formatted as an edgelist, I propose visualizing the network by the following procedure:

1. Reshape the data into an adjacency matrix.
2. Compute the matrix of shortest paths (the distance matrix).
3. Arrange the vertices on a circle in a random order, and then compute their coordinates.
4. Using the coordinates circle layout obtained in the previous step as a source of starting positions, use the modern method to compute coordinates for the vertices by `mds`.
5. Draw the graphic by combining the `twoway` plot types `pcspike` or `pcarrow` with `scatter`.

In my implementation, steps 1–3 are performed in Mata. The calculation of the distance matrix (step 2) involves calculating higher powers of the adjacency matrix

and can be rather time consuming for larger networks. More efficient procedures for obtaining distances in a network are feasible, but they are not implemented in my example.

I chose Stata's iterative modern `mds` method for step 4 because it allows for the specification of starting positions and appears to provide better results in tests. In particular, the modern method performs better than the classic method with regard to the placement of vertices that have identical distances to all other vertices (for example, vertices on the periphery of a "star"). Experimentation furthermore suggests that starting with a circular layout provides the best results.[1]

3 Implementation: The netplot command

3.1 Syntax

`netplot` *var1 var2* [*if*] [*in*] [`, type(mds|circle) label arrows iterations(#)`]

The `netplot` command produces a graphical representation of a network stored as an extended edgelist or arclist in *var1* and *var2*.

3.2 Options

`type(mds|circle)` specifies the type of layout. Valid values are `mds` or `circle`.

`mds` calculates positions of vertices using MDS. This is the default if `type()` is not specified.

`circle` arranges vertices on a circle.

`label` specifies that vertices be labeled using their identifiers in *var1* and *var2*.

`arrows` specifies that arrows rather than lines be drawn between vertices. Arrows run from the vertex in *var1* to the vertex in *var2*. This option is useful for arclists that represent directed relations.

`iterations(#)` specifies the maximum number of iterations in the MDS procedure. The default is `iterations(1000)`.

4 Examples

To illustrate the process outlined above, I use the well-known Padgett's Florentine Families dataset, which contains information on relations among 16 families in fifteenth-

1. Internally, my program issues the command `mdsmat` *distance_matrix*`, noplot method(modern) initialize(from(`*circle_matrix*`)) iterate(#)`.

century Florence, Italy (Padgett and Ansell 1993). The part of the data I use represents marital relations between the families. These relations are by nature nondirected. The data are described below:

```
. describe
Contains data from Padgett_marital02_undir.dta
  obs:            21                          Padgett marital data with
                                                undirected ties
 vars:             2                          22 Jan 2010 17:37
 size:           588 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
from            str12  %12s                   family 1 name
to              str12  %12s                   family 2 name
-------------------------------------------------------------------------------
Sorted by:  from  to
. list, sepby(from)
```

	from	to
1.		Pucci
2.	Albizzi	Guadagni
3.	Albizzi	Medici
4.	Barbadori	Medici
5.	Bischeri	Guadagni
6.	Bischeri	Peruzzi
7.	Bischeri	Strozzi
8.	Castellani	Barbadori
9.	Castellani	Strozzi
10.	Ginori	Albizzi
11.	Guadagni	Lamberteschi
12.	Medici	Acciaiuoli
13.	Medici	Salviati
14.	Medici	Tornabuoni
15.	Pazzi	Salviati
16.	Peruzzi	Castellani
17.	Peruzzi	Strozzi
18.	Ridolfi	Medici
19.	Ridolfi	Tornabuoni
20.	Strozzi	Ridolfi
21.	Tornabuoni	Guadagni

The data are in this case formatted as strings that simply use the family names as identifiers for the vertices of the network.

The first example (figure 3) shows the most basic usage of `netplot`. It uses the `netplot from to` command to produce a network plot of the data resulting from MDS.

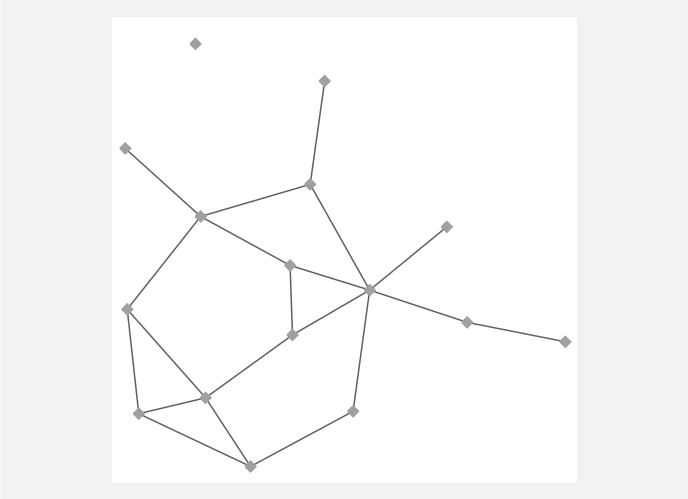


Figure 3. Marital relations among Florentine families, with vertex placement by MDS

In many analyses, it is useful to be able to identify specific vertices in the network. Identification is facilitated by adding labels to the plot using the `label` option (figure 4).[2] I can now observe that this network has a cohesive core formed by the Medici, Ridolfi, and Tornabuoni families, and that the isolated vertex is the Pucci family.

2. The placement of labels outside the plot region is part of the default behavior of `twoway scatter`, which is used by `netplot`. This can be easily adjusted afterward.

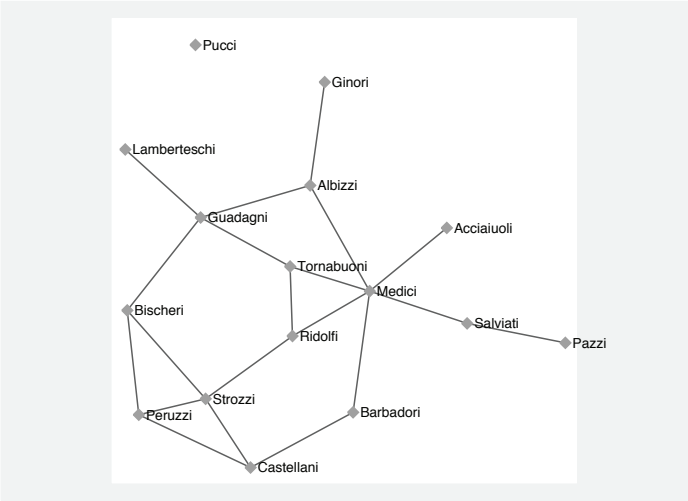


Figure 4. Marital relations among Florentine families, with vertex placement by MDS and labels added

Sometimes it is not necessary to have the relatively complicated plot as produced by MDS. Then a simple view on the data can be produced by the `circle` option (figure 5).

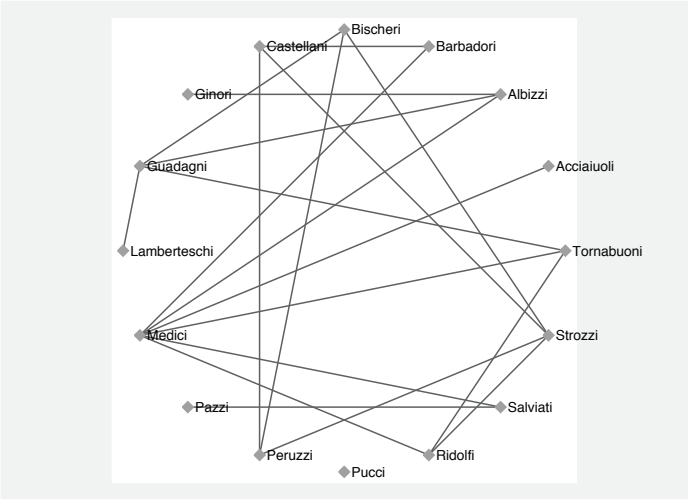


Figure 5. Marital relations among Florentine families, with circular vertex placement and labels

For my final example with these data, I assume that the data are directed. That is, I assume that each line in the data represents a directed relation from one vertex to another vertex. Imagine, for instance, that the data now represent whether a family has

ever sold goods to another family. Such situations can be visualized using the `arrows` option, which draws arrows instead of lines between vertices (figure 6). The graph in this example was slightly adjusted afterward by using the Graph Editor to reduce the sizes of the markers and to make the arrowheads better visible.

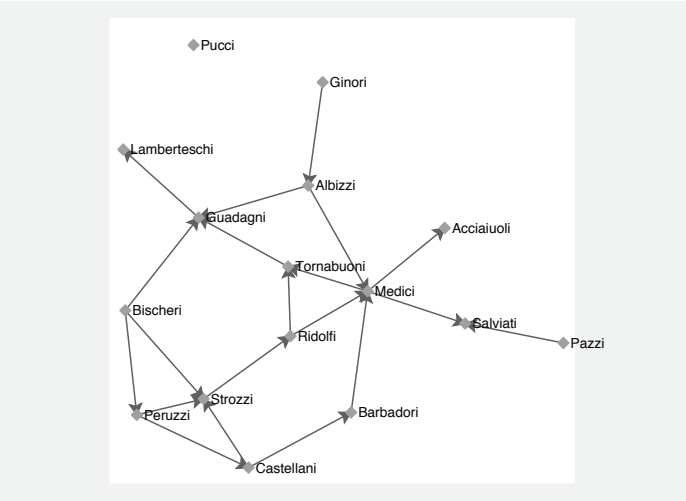


Figure 6. Marital relations among Florentine families, shown as directed relations with vertex placement by MDS and labels

As a final example, I draw a plot of a somewhat larger network of 100 vertices. The data for this example were simulated using the "preferential attachment" algorithm proposed by Barabási and Albert (1999) to construct the network shown in figure 7.[3]

This example highlights two limitations of `netplot`. First, as figure 7 shows, vertex placement can be suboptimal: several vertices in the figure are placed too close together, while others are placed too far from neighboring vertices, which leads to crossings of edges. The reason is that in this particular treelike network structure, there are many vertices that have the exact same distance to all other vertices, which makes placement by MDS difficult. Second (not visible in the figure), the procedure becomes considerably more time consuming with this number of vertices. I discuss this issue in more detail in the next section.

3. The actual simulation was conducted in Mata. The function in which the Albert–Barabási algorithm is implemented is part of a larger library of functions for network analysis under development by the author.

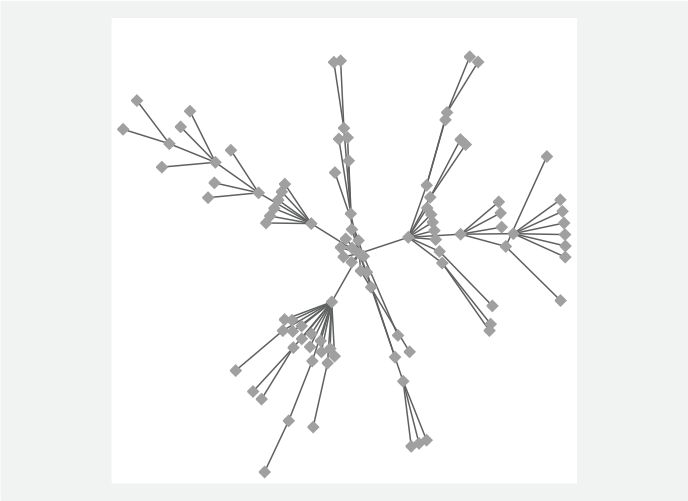


Figure 7. Marital relations among Florentine families, shown as directed relations with vertex placement by MDS

5 Performance

To get a rough idea of the performance of `netplot` in terms of computation time, I conduct two simulated tests. First, I draw plots of networks of increasing network size, keeping network density constant at 0.5. This leads to an exponentially increasing number of edges in the network. To draw the plots, I use `netplot` without any options. The input networks are randomly generated Erdös–Rényi graphs (Erdös and Rényi 1959).[4]

For the second test, I again draw plots with increasing network size but keep average degree constant rather than density. This implies a linear increase in the number of edges in the network. I use an average degree of 3.

In both tests, I look at networks with sizes ranging from 5 to 100 in increments of 5. In addition, I simulate networks of 500 nodes and networks of 1,000 nodes. I keep track of the average time needed to draw a graph over 10 iterations per network size.

The results are shown in figure 8 for the networks of up to 100 nodes. The figure indicates that average time increases quadratically with network size, although time increases more strongly with constant density than with constant degree. For networks of 500 nodes, computation times average 1,545 seconds for networks with a density of 0.5 and 1,182 seconds for networks with an average degree of 3. For networks of 1,000 nodes, the average times are 6,936 seconds and 7,769 seconds, respectively.

4. The tests were run in Stata/SE 11 on a PC with a 2.66-GHz dual-core processor and 1 GB of memory and running the Microsoft Windows XP 32-bit operating system.

Obviously, computation time becomes a major obstacle when using `netplot` on larger networks. In addition, convergence and computational problems of the MDS procedure become more frequent in larger networks. Closer analysis (not reported) of the running time of the different components of the command reveals that the computation of coordinates using MDS after computation of distances is the most time-consuming step in the procedure.

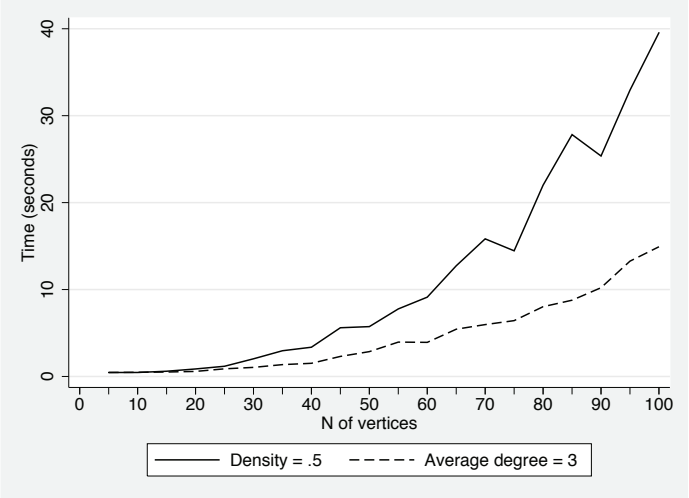


Figure 8. Average computation time by network size

6 Discussion

In this article, I have demonstrated how to use built-in techniques for MDA and graphics to visualize network data in Stata. This method often produces useful results, although not always for all networks. A major drawback is the long computation time needed to compute vertex coordinates on larger networks. As a workaround for this problem, the number of iterations may be limited by using the `iterations()` option.

Visual results could likely be improved by using vertex placement algorithms different from MDS. Good candidates are the often-used "spring embedding" algorithms by Kamada and Kawai (1989) and Fruchterman and Reingold (1991). Given the command architecture of `netplot`, these methods could be added relatively easily, and implementing them would be an obvious target for future development.

A second reason to focus on placement algorithms different from MDS in future development is that the MDS procedure appears to be the major cause of the long computation time needed for large networks. At this moment, however, it is not clear how, for example, the Kamada–Kawai and Fruchterman–Reingold algorithms compare with MDS in terms of computation time.

Another approach to improving efficiency is to use more-efficient methods for computing distances in the network. The simple approach currently implemented, which is based on repeated matrix squaring, computes some quite unneeded information in the process. More-efficient algorithms for computing shortest paths exist (see Cormen et al. [2001]) and might be implemented in the future.

The introduction of Mata with Stata 9 has made matrix programming more effective and more accessible for the average user. This opens up further possibilities for the development of SNA methods in Stata. The fact that Mata can be used interactively makes it easier to use the alternative data structures representing networks common in SNA. The quickly growing interest in social networks in and outside the social sciences certainly justifies the further development of SNA methods for Stata.

7 References

Barabási, A.-L., and R. Albert. 1999. Emergence of scaling in random networks. *Science* 286: 509–512.

Batagelj, V., and A. Mrvar. 2009. *Pajek*. Program for Large Network Analysis. Ljubljana, Slovenia. http://vlado.fmf.uni-lj.si/pub/networks/pajek/.

Borgatti, S. P., M. G. Everett, and L. C. Freeman. 1999. *UCINET*. Program for Social Network Analysis. Lexington, KY. http://www.analytictech.com/ucinet/.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein. 2001. *Introduction to Algorithms*. 2nd ed. Cambridge, MA: MIT Press.

Erdös, P., and A. Rényi. 1959. On random graphs, I. *Publicationes Mathematicae (Debrecen)* 6: 290–297.

Freeman, L. C. 2000. Visualizing social networks. *Journal of Social Structure* 1. http://www.cmu.edu/joss/content/articles/volume1/Freeman.html.

Fruchterman, T. M. J., and E. M. Reingold. 1991. Graph drawing by force-directed placement. *Software—Practice and Experience* 21: 1129–1164.

Kamada, T., and S. Kawai. 1989. An algorithm for drawing general undirected graphs. *Information Processing Letters* 31: 7–15.

Laumann, E. O., and L. Guttman. 1966. The relative associational contiguity of occupations in an urban setting. *American Sociological Review* 31: 169–178.

Newman, M., A.-L. Barabási, and D. Watts, ed. 2006. *The Structure and Dynamics of Networks*. Princeton, NJ: Princeton University Press.

Padgett, J. F., and C. K. Ansell. 1993. Robust action and the rise of the Medici, 1400–1434. *American Journal of Sociology* 98: 1259–1319.

Scott, J. 2000. *Social Network Analysis: A Handbook*. 2nd ed. London: Sage.

Wasserman, S., and K. Faust. 1994. *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.

About the author

Rense Corten is a postdoctoral researcher at the Department of Sociology and the Interuniversity Center for Social Science Theory and Methodology, Utrecht University.

The Stata Journal (2011)
11, Number 1, pp. 64–81

Pointwise confidence intervals for the covariate-adjusted survivor function in the Cox model

Matthew Cefalu
Department of Statistics
Texas A&M University
College Station, TX
mattcefalu@tamu.edu

Abstract. A graphical representation of the pointwise confidence intervals allows a researcher to easily assess the precision of estimators. In the absence of covariates, the official command `sts graph` can be used to plot these intervals for the survivor function or the cumulative hazard function; however, in the presence of covariates, `sts graph` is insufficient. The user-written command `survci` can be used to plot the pointwise intervals for the survivor function after the Cox model. In this article, I describe the current and new features of `survci`. The new features include pointwise confidence intervals for the cumulative hazard function and the support of stratified Cox models, as well as factor variables; available as of Stata 11. I describe the methods used in calculating pointwise confidence intervals in the Cox model for both the covariate-adjusted survivor function and the covariate-adjusted cumulative hazard function. I also demonstrate the syntax of `survci` using Stata's example cancer dataset, `cancer.dta`.

Keywords: st0217, survci, confidence intervals, covariate adjusted survivor function, Cox model

1 Introduction

In analyzing time-to-event data, the Cox proportional hazard model (Cox 1972) is often used to adjust for additional characteristics, other than time, that may affect an individual's outcome. These covariates are of great concern because ignoring them could lead to biased results. The Cox model assumes that the ratio of the hazard rates for two individuals at any time is constant. In other words, the hazard rate for an individual is some baseline hazard rate multiplied by a function of only the covariates.

Because we are interested in confidence intervals of the covariate-adjusted survivor function, we must first have an approximation of the baseline survivor function. The several available estimators include Kalbfleisch and Prentice (2002), which is the estimator used in Stata. However, the methods and formulas presented in this article are valid for any estimate of the baseline survivor function.

Once we have an estimate of the baseline survivor function, we want not only to predict a covariate-adjusted survivor function but also to know how precise our predictions are for each observed time. One approach is to form pointwise confidence intervals at each time and to use those intervals to make any necessary inferences.

In this article, I describe the `survci` command, which plots the pointwise confidence intervals of the survivor function or the cumulative hazard function after Cox regression, and its underlying methodology. The first version of `survci`, written by Yulia Marchenko of StataCorp, provided confidence intervals of only the survivor function. Yulia received a number of requests from users for the confidence intervals of the cumulative hazard function, as well as for the support of stratified Cox models. These requests prompted my work on adding these capabilities to `survci` during my internship at StataCorp in 2009. The new features also include the support of factor variables, available as of Stata 11, and the customization of graphs via the new `plotopts()` and `ciopts()` options.

Although the new features of `survci` require Stata 11, the old features will still be available to Stata 10 users through version control.

2 Methods and formulas

To form confidence intervals, we first need an approximation of the standard error. Marubini and Valsecchi (1995) describe a method derived by Tsiatis (2006), which is described below. An alternative estimator can be found in Hosmer and Lemeshow (1999).

2.1 Variance estimator

Given the vector of estimated coefficients $\widehat{\boldsymbol{\beta}}$ from the Cox model, let $\widehat{S}_0(t)$ be an estimate of the baseline survivor function at time t. Under the assumptions of the Cox model, the estimated survivor function is

$$\widehat{S}\left(t, \mathbf{x}^*\right) = \left\{\widehat{S}_0(t)\right\}^{\exp\left(\widehat{\boldsymbol{\beta}}^T \mathbf{x}^*\right)}$$

where $\mathbf{x}^*$ is a particular covariate vector.

Marubini and Valsecchi (1995) provide an estimator of the variance of $\widehat{S}(t, \mathbf{x}^*)$ that is generalized to consider the presence of a few tied values. Consider a sample of N subjects with a total of J failures, where $J < N$ in the presence of censoring. Let $t_{(1)} < t_{(2)} < \cdots < t_{(J)}$ be the J distinct ordered observed failure times. Let d_j be the number of tied observations and R_j be the set of subjects at risk at time $t_{(j)}$. Then the variance estimator is given by

$$\widehat{\text{var}}\left\{\widehat{S}\left(t,\mathbf{x}^*\right)\right\} = \left\{\widehat{S}(t,\mathbf{x}^*)\right\}^2 \exp\left(2\widehat{\boldsymbol{\beta}}^T\mathbf{x}^*\right) \left[\sum_{t_{(j)}\le t}\frac{d_j}{\left\{\sum_{i\in R_j}\exp\left(\widehat{\boldsymbol{\beta}}^T\mathbf{x}_i\right)\right\}^2} + \widehat{\boldsymbol{\rho}}^T(t,\mathbf{x}^*)I^{-1}\left(\widehat{\boldsymbol{\beta}}\right)\widehat{\boldsymbol{\rho}}(t,\mathbf{x}^*)\right]$$

where $\mathbf{x}_i$ is the vector of observed covariate values for subject i; $I^{-1}\left(\widehat{\boldsymbol{\beta}}\right)$ is the variance–covariance matrix of $\widehat{\boldsymbol{\beta}}$ estimated by the inverse of the observed information matrix; and $\widehat{\boldsymbol{\rho}}(t,\mathbf{x}^*)$ is a vector, with the kth element being

$$\widehat{\rho}_k(t,\mathbf{x}^*) = \sum_{t_{(j)}\le t} d_j \left[\left\{x_k^* - \frac{\sum_{i\in R_j} x_{ki}\exp\left(\widehat{\boldsymbol{\beta}}^T\mathbf{x}_i\right)}{\sum_{i\in R_j}\exp\left(\widehat{\boldsymbol{\beta}}^T\mathbf{x}_i\right)}\right\}\frac{1}{\sum_{i\in R_j}\exp\left(\widehat{\boldsymbol{\beta}}^T\mathbf{x}_i\right)}\right]$$

Related formulas can be used to estimate the variance of the cumulative hazard function if we use the relationship between it and the survivor function. Details can be found in Klein and Moeschberger (2003).

2.2 Pointwise confidence intervals

Several methods have been proposed in the literature to form a $(1-\alpha)\times 100\%$ confidence interval for the survivor function at any time $t_0 \ge 0$. The simplest of these intervals, the normal-based confidence interval, is given by

$$\widehat{S}(t_0,\mathbf{x}^*) \pm Z_{1-\alpha/2}\sigma_{\widehat{S}}(t_0,\mathbf{x}^*)\widehat{S}(t_0,\mathbf{x}^*)$$

where $\sigma_{\widehat{S}}(t_0,\mathbf{x}^*) = \sqrt{\widehat{\text{var}}\left\{\widehat{S}\left(t_0,\mathbf{x}^*\right)\right\}}$ and $Z_{1-\alpha/2}$ is the $1-\alpha/2$ percentile of the standard normal distribution.

A better interval can be formed by finding a confidence interval for the log of the cumulative hazard function and transforming it back to an interval for the survivor function. Because we are first finding an interval for the log of the cumulative hazard function, which is the negative log of the survivor function, this interval is referred to as the log–log-based interval:

$$\left[\widehat{S}(t_0,\mathbf{x}^*)^{1/\theta}, \widehat{S}(t_0,\mathbf{x}^*)^{\theta}\right], \text{where } \theta = \exp\left[\frac{Z_{1-\alpha/2}\sigma_{\widehat{S}}(t_0,\mathbf{x}^*)}{\ln\left\{\widehat{S}(t_0,\mathbf{x}^*)\right\}}\right]$$

Other confidence intervals can be formed by using transformations of $\widehat{S}(t_0,\mathbf{x}^*)$. Details of these other intervals can be found in many sources, one of which is Klein and Moeschberger (2003).

Similar confidence intervals can be formed around the point estimates of the cumulative hazard function. Again, the normal-based confidence intervals are the simplest:

$$\widehat{H}(t_0, \mathbf{x}^*) \pm Z_{1-\alpha/2}\sigma_{\widehat{H}}(t_0, \mathbf{x}^*)$$

where $\widehat{H}(t_0, \mathbf{x}^*)$ is an estimator of the cumulative hazard function (for example, Marubini and Valsecchi [1995]).

As with the survivor function, using transformations of the cumulative hazard function produces better confidence intervals. Using the same transformation as before, the log of the cumulative hazard function, we form the log-based confidence interval for the cumulative hazard function:

$$\left[\widehat{H}(t_0, \mathbf{x}^*)/\phi, \phi\widehat{H}(t_0, \mathbf{x}^*)\right], \text{where } \phi = \exp\left\{\frac{Z_{1-\alpha/2}\sigma_{\widehat{H}}(t_0, \mathbf{x}^*)}{\widehat{H}(t_0, \mathbf{x}^*)}\right\}$$

2.3 Stratified Cox model

The Cox model can be extended to handle the situation when the assumption of proportional hazards is violated for some covariates by stratifying on those covariates such that the assumption is valid within each stratum. Therefore, we fit a separate baseline hazard function for each stratum and assume that the covariate effects are the same regardless of strata. Details can be found in Klein and Moeschberger (2003). If we wish to create confidence intervals for the survivor function in this case, we can use the previous formulas within each stratum.

3 The survci command

3.1 Syntax

```
survci [ if ] [ in ] [ , survival cumhaz at(varname=# [ varname=# ... ])
   at#(varname=# [ varname=# ... ]) citype(loglog|log|normal) level(#)
   outfile(filename [ , replace failonly ]) separate range(#,#)
   plotopts(cline_options) plot#opts(cline_options) ciopts(cline_options)
   ci#opts(cline_options) twoway_options byopts(byopts) ]
```

3.2 Description

`survci` plots the covariate-adjusted pointwise confidence intervals for the survivor function or the cumulative hazard function after `stcox`.

3.3 Options

`survival` specifies that the covariate-adjusted survivor function, along with its pointwise confidence intervals, be plotted. This is the default.

`cumhaz` specifies that the covariate-adjusted cumulative hazard function, along with its pointwise confidence intervals, be plotted. This option may not be combined with `survival`.

`at(`*varname*`=#` [*varname*`=#` ...]`)` specifies the values of the covariates used in `stcox` for which the estimates of the plotted function are to be computed. If left unspecified, continuous covariates will be set to their mean values, and factor variables will be set to their base levels.

`at#(`*varname*`=#` [*varname*`=#` ...]`)` specifies the values of the covariates used in `stcox` for which the estimates of the plotted function for the #th strata are to be computed. By default, continuous covariates will be set to their stratum-specific mean values, and factor variables will be set to their base levels. This option may not be combined with `at()`.

`citype(loglog|log|normal)` specifies the type of confidence interval to use: `loglog` for the log–log-based intervals, `log` for the log-based intervals, and `normal` for the normal-based intervals. `loglog` is the default with `survival`, and `log` is the default with `cumhaz`.

`level(#)` specifies the pointwise confidence level, as a percentage, for confidence intervals. The default is `level(95)` or as set by `set level`.

`outfile(`*filename* [`, replace failonly`]`)` saves the estimates of pointwise confidence intervals, standard errors, and covariate-adjusted functions to *filename*. `replace` indicates that *filename* be overwritten, if it exists. `failonly` requests that only failures be saved in *filename*; otherwise, all observations will be saved.

`separate` specifies that stratified estimates be plotted on separate subgraphs, one per stratum. The default is to overlay stratum-specific curves on one graph. If no strata are present, `separate` is ignored.

`range(#,#)` specifies the range of the time axis to be plotted. If this option is not specified, `survci` plots the desired curve on an interval expanding from the earliest to the latest analysis time in the data.

`plotopts(`*cline_options*`)` affects the rendition of the plotted lines; see [G] ***cline_options***. If a stratified Cox model is used, `plotopts()` applies to all strata.

`plot#opts(`*cline_options*`)` affects the rendition of the #th plotted stratum-specific line; see [G] ***cline_options***. This option may not be combined with `plotopts()` or `separate`.

`ciopts(`*cline_options*`)` affects the rendition of the confidence intervals; see [G] ***cline_options***. If a stratified Cox model is used, `ciopts()` applies to all strata.

`ci#opts(`*cline_options*`)` affects the rendition of the #th stratum-specific confidence interval; see [G] ***cline_options***. This option may not be combined with `ciopts()` or `separate`.

twoway_options are any of the options documented in [G] ***twoway_options***, except `by()`. These include options for titling the graph (see [G] ***title_options***) and for saving the graph to disk (see [G] ***saving_option***).

`byopts(`*byopts*`)` affects the appearance of the combined graph when `separate` is specified, including the overall graph title and the organization of subgraphs. See [G] ***by_option***.

4 Examples

4.1 Basic use

I will demonstrate the use of `survci` using fictional data from patient survival in a drug trial, which is used repeatedly in Stata's documentation. There are 48 observations, each one corresponding to a unique individual. In the study, there were two drugs of interest and one placebo (`drug=1`), stored in the `drug` variable. The `studytime` variable records months until death or the end of the experiment. The `died` variable contains 1 if the patient died and 0 otherwise. The age of the patient is recorded in the `age` variable.

`survci` is a postestimation command and therefore can only be used after `stcox`. Many `stset` options are supported, along with some `stcox` options. To start, let me demonstrate the simplest use of `survci`. We first fit a Cox model with covariates `age` and `drug` using `stcox`, followed by `survci` to plot the survivor function and its confidence intervals:

```
. sysuse cancer
(Patient Survival in Drug Trial)
. stset studytime, failure(died)
     failure event:  died != 0 & died < .
obs. time interval:  (0, studytime]
 exit on or before:  failure
------------------------------------------------------------------------------
       48  total obs.
        0  exclusions
------------------------------------------------------------------------------
       48  obs. remaining, representing
       31  failures in single record/single failure data
      744  total analysis time at risk, at risk from t =         0
                            earliest observed entry t =         0
                                 last observed exit t =        39
```

```
. stcox age i.drug
        failure _d:  died
  analysis time _t:  studytime
Iteration 0:   log likelihood = -99.911448
Iteration 1:   log likelihood = -82.331523
Iteration 2:   log likelihood = -81.676487
Iteration 3:   log likelihood = -81.652584
Iteration 4:   log likelihood = -81.652567
Refining estimates:
Iteration 0:   log likelihood = -81.652567
Cox regression -- Breslow method for ties
No. of subjects =           48                     Number of obs   =        48
No. of failures =           31
Time at risk    =          744
                                                   LR chi2(3)      =     36.52
Log likelihood  =   -81.652567                     Prob > chi2     =    0.0000
```

_t	Haz. Ratio	Std. Err.	z	P>|z|	[95% Conf.	Interval]
age	1.118334	.0409074	3.06	0.002	1.040963	1.201455
drug						
2	.1805839	.0892742	-3.46	0.001	.0685292	.4758636
3	.0520066	.034103	-4.51	0.000	.0143843	.1880305

```
. survci
(age=55.88; drug=1.00)
```

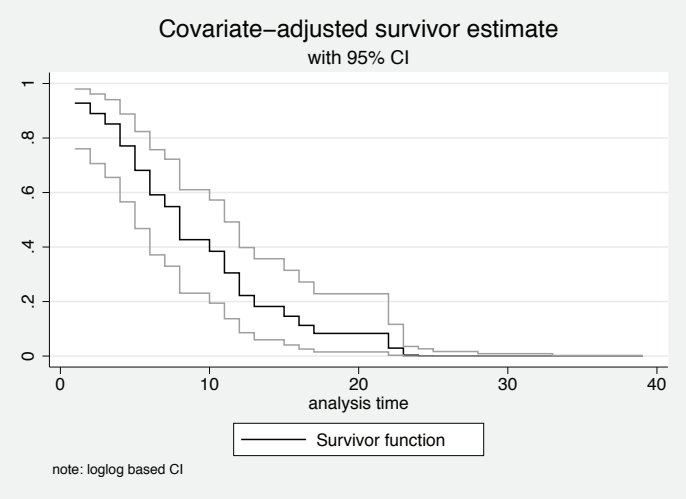


Figure 1. Survivor function

We can see in figure 1 that `survci` defaults to plotting the survivor function and its pointwise confidence interval. `survci` also outputs the values of the covariates at which the survivor function is computed. In this case, because no covariate values were

specified with the `at()` option, `age` is set equal to its mean, `age=55.88`, and a factor variable `drug` is set equal to its base level, `drug=1`.

Now suppose that we are interested in the normal-based interval. By looking at the note at the bottom of figure 1, we can see that the type of confidence interval used was a log–log-based interval. For a description of the difference in the confidence intervals, see section 2.2. To specify the type of confidence interval to be calculated by `survci`, we use the `citype()` option. We can also change its confidence level with the `level()` option.

```
. survci, citype(normal) level(99)
(age=55.88; drug=1.00)
```

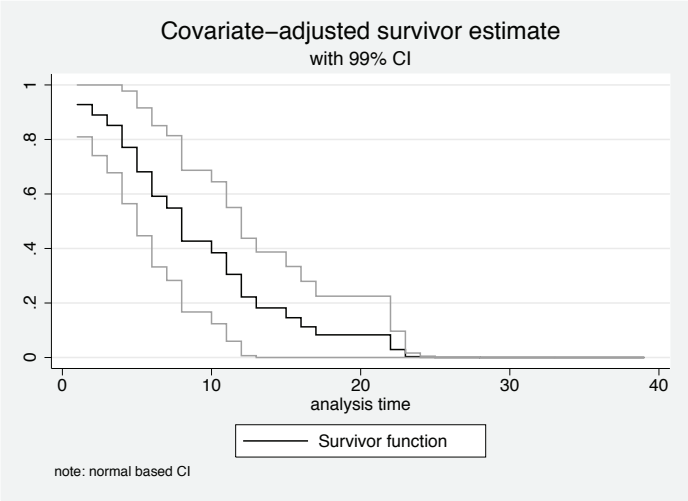


Figure 2. Survivor function, normal-based confidence intervals

4.2 Setting covariate values and levels of factor variables

In many circumstances, it may be of interest to plot the survivor function for specified values of covariates. Suppose we are interested in plotting the survivor estimate and confidence intervals for `age=50`. We can accomplish this in `survci` by using the `at()` option.

```
. survci, at(age=50)
(age=50.00; drug=1.00)
```

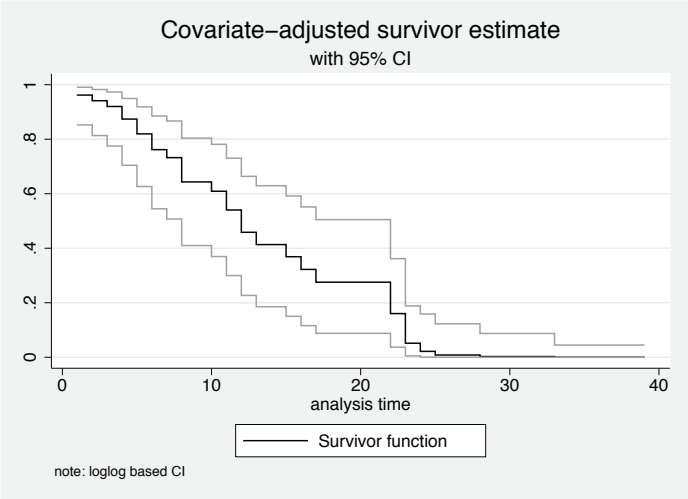


Figure 3. Survivor function at `age=50`

If we now compare figure 1 with figure 3, we will see that, as expected, the survival of a 50-year-old subject is higher than that of a 56-year-old subject.

In all our examples, a value of 1 was used for the `drug` variable. Recall that we included `drug` as a factor variable in `stcox`; see [U] **11.4.3 Factor variables** for details about factor variables. `survci` recognized this factor variable and used the value of the base category, 1, for `drug`. `survci` is written to recognize factor variables and automatically handle them correctly.

As with other covariates, you can use the `at()` option to specify other levels of factor variables. For example, suppose we want to plot the survivor function and its confidence intervals for `drug=2` and `age=50`.

```
. survci, at(age=50 drug=2)
(age=50.00; drug=2.00)
```

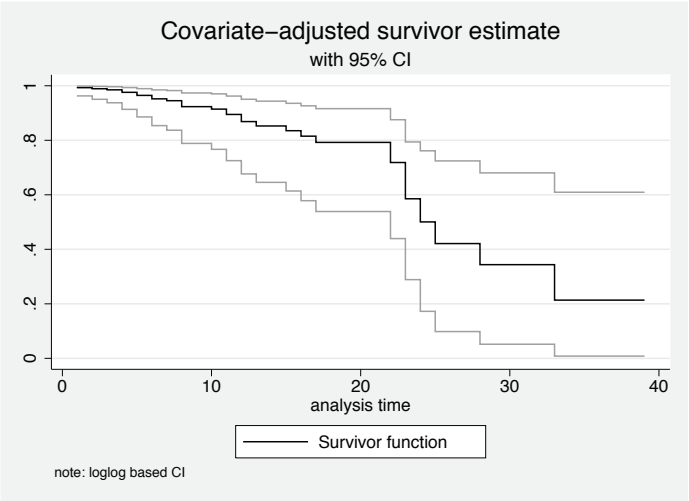


Figure 4. Survivor function at `age=50` and `drug=2`

From figure 3 and figure 4, we can see that the survival of subjects from the `drug=2` group is higher (but also has higher variation) than that of subjects from the placebo group, `drug=1`.

It is important to note that a single value should be assigned to a factor variable in the `at()` option. Individual levels may not be specified:

```
. survci, at(age=50 1.drug=1)
level indicators of factor variables may not be individually set with the at()
option; set one value for the entire factor variable
```

`survci` also uses the factor-variable system in determining the value of an interaction. For the purpose of illustration, let us create a variable, `age_cat`, that breaks `age` into three levels. Then let us use it in interaction with the factor variable `drug` in `stcox`.

```
. generate age_cat = 1
. replace age_cat = 2 if age>55
(25 real changes made)
. replace age_cat = 3 if age>58
(14 real changes made)
. tabulate age_cat
```

age_cat	Freq.	Percent	Cum.
1	23	47.92	47.92
2	11	22.92	70.83
3	14	29.17	100.00
Total	48	100.00	

```
. stcox age_cat#drug
         failure _d:  died
   analysis time _t:  studytime
Iteration 0:   log likelihood = -99.911448
Iteration 1:   log likelihood = -97.925217
Iteration 2:   log likelihood = -92.125619
Iteration 3:   log likelihood = -81.261211
Iteration 4:   log likelihood = -79.713568
Iteration 5:   log likelihood = -79.660143
Iteration 6:   log likelihood = -79.659946
Iteration 7:   log likelihood = -79.659946
Refining estimates:
Iteration 0:   log likelihood = -79.659946
Cox regression -- Breslow method for ties
No. of subjects =           48                     Number of obs   =        48
No. of failures =           31
Time at risk    =          744
                                                   LR chi2(8)      =     40.50
Log likelihood  =   -79.659946                     Prob > chi2     =    0.0000
```

_t	Haz. Ratio	Std. Err.	z	P>|z|	[95% Conf.	Interval]
age_cat#drug						
1 2	.0633389	.0691915	-2.53	0.012	.0074442	.5389197
1 3	.0615073	.0527435	-3.25	0.001	.0114554	.3302521
2 1	1.182804	.7281159	0.27	0.785	.3539375	3.95275
2 2	.6967579	.5537083	-0.45	0.649	.1467704	3.307695
2 3	.0974318	.090084	-2.52	0.012	.0159108	.596636
3 1	4.45363	2.533906	2.63	0.009	1.460227	13.58338
3 2	.8499884	.5786483	-0.24	0.811	.2238406	3.227656
3 3	.0360163	.0446832	-2.68	0.007	.0031657	.4097629

Now that we have an interaction of factor variables in our model, suppose we want to predict the survivor function at the interaction when `age_cat=2` and `drug=2`. In `survci`, we do not specify the interaction but the individual factors that make up the interaction. Therefore, we will just set `age_cat=2` and `drug=2`, and `survci` will recognize that they are involved in the interaction, although there is no indication of this in the output:

```
. survci, at(age_cat=2 drug=2)
(age_cat=2.00; drug=2.00)
```

(*output omitted*)

If several factor variables are involved in the same interaction, all the variables must be specified in the `at()` option:

```
. survci, at(age_cat=2)
the model contains an interaction involving factor variables; the level for
each variable in the interaction must be set with the at() option
```

If no `at()` option is used, all factors are set to their base values:

```
. survci
(age_cat=1.00; drug=1.00)
```
(*output omitted*)

Although `survci` also supports the use of the `xi` prefix with `stcox`, it is recommended that you instead use factor variables. The reason is that `survci` will not recognize the interaction terms as it does when factor variables are used. Therefore, it is your responsibility to specify all the values of the interactions involved in the model if the `xi` prefix is used with `stcox`. Also with `xi`, the individual indicators for the factors must be specified in the `at()` option, as if each were its own variable. As you can see below, getting the names of all the indicators correct can be quite difficult when `xi` is used. This is just another reason to use factor variables instead of `xi`.

```
. xi: stcox i.age_cat*i.drug
```
(*output omitted*)
```
. survci, at(_Iage_cat_2=1 _Iage_cat_3=0 _Idrug_2=1 _Idrug_3=0 _IageXdru_2_2=1
> _IageXdru_2_3=0 _IageXdru_3_2=0 _IageXdru_3_3=0)
(_Iage_cat_2=1.00; _Iage_cat_3=0.00; _Idrug_2=1.00; _Idrug_3=0.00;
  _IageXdru_2_2=1.00; _IageXdru_2_3=0.00; _IageXdru_3_2=0.00;
  _IageXdru_3_3=0.00)
```
(*output omitted*)

4.3 Customizing the look of graphs

Now that we know how to use `survci` to plot the survivor function, we will see how to format the look of the graph. Suppose we want to make the survivor function be a solid line and the confidence intervals be dashed lines. The `plotopts()` option controls the look of the survivor function line, and the `ciopts()` option controls the look of the confidence intervals. Any *cline_options* are allowed within these options; see [G] ***cline_options***. We use `lpattern(solid)` and `lpattern(dash)` to obtain the desired plot.

```
. stcox age i.drug
  (output omitted)
. survci, plotopts(lpattern(solid)) ciopts(lpattern(dash))
(age=55.88; drug=1.00)
```

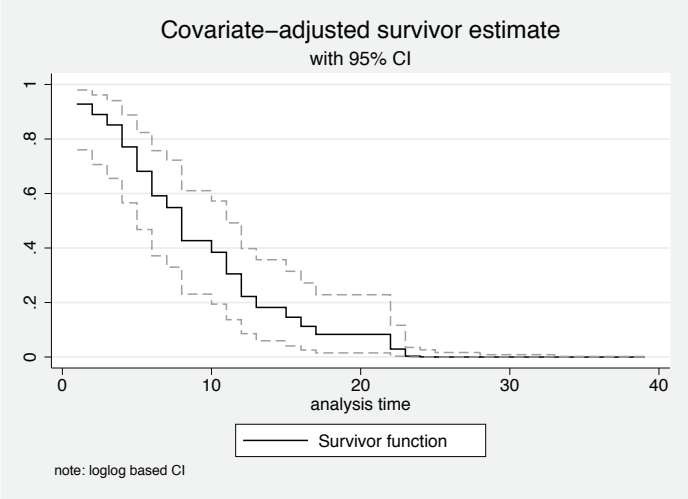


Figure 5. Survivor function, customized graph

If a stratified Cox model is used, `plotopts()` and `ciopts()` apply the specified options to all strata. To further control the look of the graph, any *twoway_options* (except the `by()` option) are allowed in `survci`; see [G] ***twoway_options***.

4.4 Stratified Cox models

Next we will see how `survci` handles stratified Cox models. First, we need to create a strata variable. Once created, we will fit the stratified Cox model with `stcox` and then use `survci` to plot the stratified estimates of the survivor function.

```
. set seed 100
. generate group = round(uniform())
. stcox age i.drug, strata(group)
        failure _d:  died
  analysis time _t:  studytime
Iteration 0:   log likelihood = -78.521191
Iteration 1:   log likelihood = -63.607129
Iteration 2:   log likelihood = -62.768258
Iteration 3:   log likelihood = -62.743298
Iteration 4:   log likelihood = -62.743261
Refining estimates:
Iteration 0:   log likelihood = -62.743261
Stratified Cox regr. -- Breslow method for ties
No. of subjects =           48                     Number of obs   =        48
No. of failures =           31
Time at risk    =          744
                                                   LR chi2(3)      =     31.56
Log likelihood  =   -62.743261                     Prob > chi2     =    0.0000
```

_t	Haz. Ratio	Std. Err.	z	P>|z|	[95% Conf.	Interval]
age	1.102773	.0408935	2.64	0.008	1.025467	1.185908
drug						
2	.168276	.1038296	-2.89	0.004	.0502128	.5639364
3	.0448717	.0353096	-3.94	0.000	.0095976	.2097899

Stratified by group

```
. survci
note: survivor estimates are stratified on group
(group=0: age=55.36; drug=1.00)
(group=1: age=56.43; drug=1.00)
```

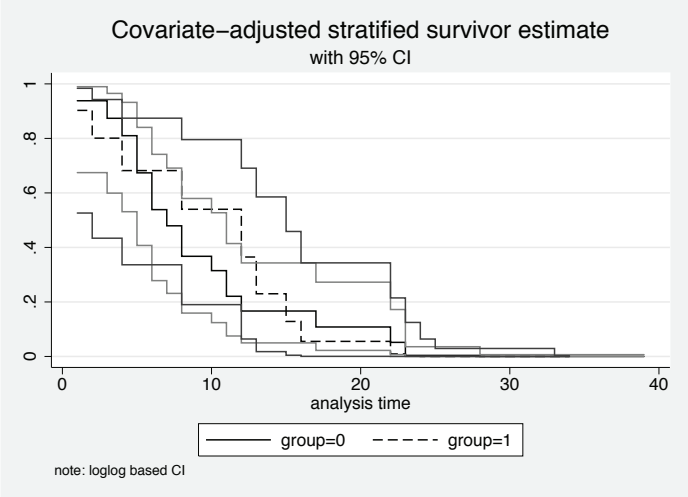


Figure 6. Stratified survivor function

When a stratified Cox model is used, `survci` plots a survivor function and its corresponding pointwise confidence intervals for each stratum in the model. If we look at the output, we see that `survci` uses a different value of the covariate `age` for each stratum. Each value is the mean of `age` within each stratum.

The produced graph with the stratified estimates can be hard to read because `survci` plots all the curves in one graph. We can use the `separate` option to create a plot of each stratum in a different panel. For better output, we create labels for our strata variable, `group`, though doing so is not required in general.

```
. label define names 0 "Group 1" 1 "Group 2"
. label values group names
. survci, separate
note: survivor estimates are stratified on group
(group=Group 1: age=55.36; drug=1.00)
(group=Group 2: age=56.43; drug=1.00)
```

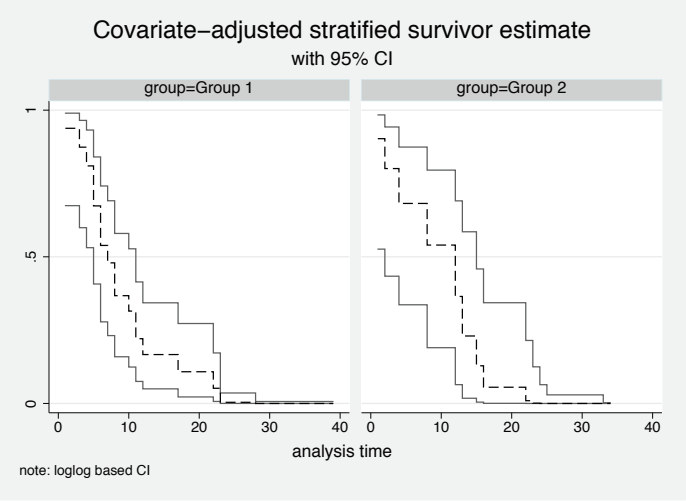


Figure 7. Survivor function, one subgraph per stratum

You can see in both figure 6 and figure 7 that `survci` attempts to label the strata. If value labels are set on the strata variable, then `survci` will use the labels to name the strata. `survci` will also recognize when multiple strata variables are present, but an example of this is not given here.

Just as before, `at()` can be used with a stratified Cox model. However, suppose we want to specify different covariate values for different strata. To do this, we will use the `at#()` option. `at#()` works just as `at()` does, but # identifies the stratum for which the "at" values are to be applied. Suppose we want to plot the survivor function for `age=50` for strata 1 but `age=55` for strata 2. We can use `at#()` as follows:

```
. survci, at1(age=50) at2(age=55) separate
note: survivor estimates are stratified on group
(group=Group 1: age=50.00; drug=1.00)
(group=Group 2: age=55.00; drug=1.00)
```

(*output omitted*)

Similar analogs exist for `plotopts()` and `ciopts()`. `plot#opts()` and `ci#opts()` work the same as `plotopts()` and `ciopts()`, but they are specific to the strata specified by #. If `separate` is specified, `plot#opts()` and `ci#opts()` are not allowed. However, we can still use `plotopts()` and `ciopts()`.

4.5 Saving plotted data

If we wish to save our estimates of the survivor function or cumulative hazard function, its pointwise standard errors, and the corresponding confidence intervals, we can use the `outfile()` option. `outfile()` defaults to saving all observations in our dataset, but we can restrict this to one observation per failure time by using `outfile()`'s suboption `failonly`.

4.6 Cumulative hazard function

All the options previously presented are also valid for the covariate-adjusted cumulative hazard function. To see how to make `survci` plot the cumulative hazard function instead of the survivor function, let us look at the simplest example:

```
. sysuse cancer, clear
(Patient Survival in Drug Trial)
. stset studytime, failure(died)
```

(*output omitted*)

```
. stcox age i.drug
```

(*output omitted*)

```
. survci, cumhaz
(age=55.88; drug=1.00)
```

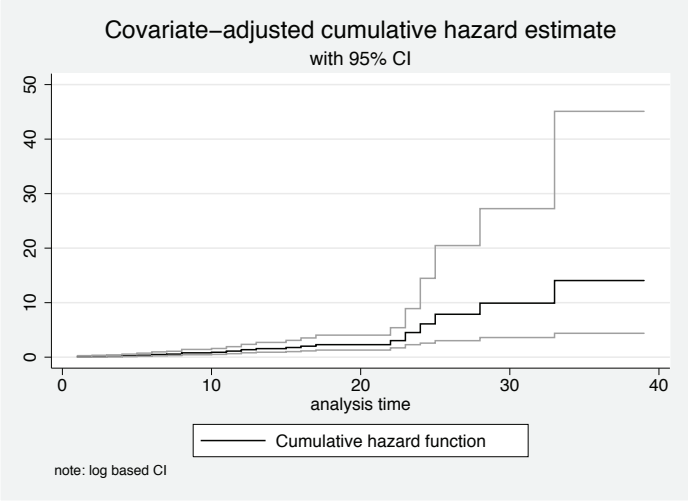


Figure 8. Cumulative hazard function

It is important to note that the default confidence interval for the cumulative hazard function (figure 8) is the log-based interval, while the default for the survivor function is the log–log-based interval (figure 1).

5 Conclusion

I have presented the methods and formulas used to calculate pointwise confidence intervals for the covariate-adjusted survivor function and cumulative hazard functions in the Cox model. I also described the user-written `survci` command, which plots the survivor function or the cumulative hazard function along with the corresponding confidence intervals, and I demonstrated its use in a few simple examples. The presented version of `survci` requires Stata 11, but the functionality of its original version, previously available from `net from http://www.stata.com/users/ymarchenko`, is available under version control.

6 Acknowledgment

This work was done during my internship at StataCorp in Summer 2009. The first version of `survci` was written by Yulia Marchenko of StataCorp.

7 References

Cox, D. R. 1972. Regression models and life-tables (with discussion). *Journal of the Royal Statistical Society, Series B* 34: 187–220.

Hosmer, D. W., Jr., and S. Lemeshow. 1999. *Applied Survival Analysis: Regression Modeling of Time to Event Data*. New York: Wiley.

Kalbfleisch, J. D., and R. L. Prentice. 2002. *The Statistical Analysis of Failure Time Data*. 2nd ed. New York: Wiley.

Klein, J. P., and M. L. Moeschberger. 2003. *Survival Analysis: Techniques for Censored and Truncated Data*. 2nd ed. New York: Springer.

Marubini, E., and M. G. Valsecchi. 1995. *Analysing Survival Data from Clinical Trials and Observational Studies*. Chichester, UK: Wiley.

Tsiatis, A. A. 2006. *Semiparametric Theory and Missing Data*. New York: Springer.

About the author

Matthew Cefalu received his Master's degree from the Department of Statistics at Texas A&M University. He is currently pursuing a PhD degree in biostatistics from the Harvard School of Public Health.

The Stata Journal (2011)
11, Number 1, pp. 82–94

Estimation of hurdle models for overdispersed count data

Helmut Farbmacher
Department of Economics
University of Munich, Germany
helmut.farbmacher@lrz.uni-muenchen.de

Abstract. Hurdle models based on the zero-truncated Poisson-lognormal distribution are rarely used in applied work, although they incorporate some advantages compared with their negative binomial alternatives. I present a command that enables Stata users to estimate Poisson-lognormal hurdle models. I use adaptive Gauss–Hermite quadrature to approximate the likelihood function, and I evaluate the performance of the estimator in Monte Carlo experiments. The model is applied to the number of doctor visits in a sample of the U.S. Medical Expenditure Panel Survey.

Keywords: st0218, ztpnm, count-data analysis, hurdle models, overdispersion, Poisson-lognormal hurdle models

1 Introduction

Hurdle models, first discussed by Mullahy (1986), are very popular for modeling count data. For example, the number of doctor visits or hospitalizations may serve as proxies for demand for health care. These measures may be determined by a two-part decision process. At first, it is up to the patient whether to visit a doctor. After the first contact, though, the physician influences the intensity of treatment (Pohlmeier and Ulrich 1995). Thus the use of a single-index count-data model (such as Poisson or negative binomial models) seems to be inappropriate in many health care applications.

The hurdle model typically combines a binary model to model participation (for example, modeling the patient's decision to visit the doctor) with a zero-truncated count-data model to model the extent of participation for those participating (for example, modeling the number of doctor visits). In contrast with a single-index model, the hurdle model permits heterogeneous effects for individuals below or above the hurdle. In many applications, the hurdle is set at zero and can therefore also solve the problem of excess zeros, that is, the presence of more zeros in the data than what was predicted by single-index count-data models.

There are many possible combinations of binary and truncated count-data models. An often-used model combines a probit or logit model with a zero-truncated negative binomial model (for example, Vesterinen et al. [2010] and Wong et al. [2010]). The zero-truncated negative binomial model is known to account for overdispersion that may be caused by unobserved heterogeneity. In this model, the heterogeneity is introduced at the level of the parent (untruncated) distribution.

Santos Silva (2003) describes an alternative method of estimating hurdle models if unobserved heterogeneity is present. He proposes using a truncated distribution and doing the mixing over this distribution only. Winkelmann's (2004) proposal of a hurdle model based on the zero-truncated Poisson-lognormal distribution follows this method. In many applications, this model seems to fit the data much better than its negative binomial alternative. The command introduced here makes it possible to estimate Poisson-lognormal hurdle models using adaptive Gauss–Hermite quadrature. It can be used with cross-sectional data, but also with panel data if one is willing to pool the data over time.

2 Model

Generally, the probability function of a hurdle model can be written as

$$f(y) = \begin{cases} g(0) & \text{if } y = 0 \\ \frac{1-g(0)}{1-h(0)} h(y) & \text{if } y \geq 1 \end{cases} \tag{1}$$

where the zeros and the positive counts are determined by the probability $g(0)$ and the truncated probability function $h(y|y > 0) = h(y)/\{1 - h(0)\}$, respectively. The numerator in (1) represents the probability of crossing the hurdle $\{1 - g(0)\}$ and is multiplied by a truncated probability function to ensure that the probabilities sum up to one. The likelihood contribution is

$$L_i^H = g(0)^{(1-d_i)} \times \left[\{1 - g(0)\} \frac{h(y)}{1 - h(0)} \right]^{d_i}$$

where d_i indicates whether individual i crosses the hurdle. Assuming that both functions are independent conditional on covariables, the maximization procedure can be divided into two separate parts. Firstly, one can maximize a binary model with d_i as a dependent variable using the full sample. Secondly, the parameters of h can be estimated separately by a truncated regression using only observations with positive counts.

Winkelmann (2004) proposes combining a probit with a zero-truncated Poisson-lognormal model in which the mixing is done over the truncated Poisson distribution. The likelihood function of this model is given by

$$L^H = \prod_{i=1}^{N} \{1 - \Phi(\mathbf{x}_i'\gamma)\}^{(1-d_i)} \times \left\{ \Phi(\mathbf{x}_i'\gamma) P^+(y_i|\mathbf{x}_i, \epsilon_i) \right\}^{d_i}$$

where $P^+(y_i|\mathbf{x}_i, \epsilon_i)$ is the probability function of the zero-truncated part and $\Phi(\mathbf{x}_i'\gamma)$ is the cumulative distribution function of the standard normal distribution. The following discussion is limited to the estimation of the truncated part. The probability function of the zero-truncated Poisson-lognormal model is given by

$$P^+(y_i|\mathbf{x}_i, \epsilon_i) = \frac{\exp(-\lambda_i)\lambda_i^{y_i}}{\{1 - \exp(-\lambda_i)\}\, y_i!}$$

where λ_i is defined by $\exp(\mathbf{x}_i\beta)\zeta_i$ and $\zeta_i = \exp(\epsilon_i)$. $\mathbf{x}_i$ is a vector of observed characteristics and ϵ_i denotes unobserved variables that might cause overdispersion.

Inference is based on the density of y_i conditional on $\mathbf{x}_i$,

$$P^+(y_i|\mathbf{x}_i) = \int_{-\infty}^{\infty} P^+(y_i|\mathbf{x}_i, \epsilon_i) f(\epsilon_i) d\epsilon_i \tag{2}$$

where $f(\epsilon_i)$ is the prior density of the unobserved heterogeneity. Because the mixing is done over the zero-truncated Poisson distribution, the calculation of the likelihood contributions is computationally more demanding than in the negative binomial alternative. There is no analytical solution for the integral in (2), and thus it has to be approximated using simulation or quadrature. The likelihood of the zero-truncated negative binomial model has a closed-form expression because the mixing is done prior to the truncation.

To complete the model, we need an assumption about the prior density of the unobserved heterogeneity. Because there is no analytical solution of the integral in (2), even if we assume a gamma density, we may think about using an assumption that is more theoretically motivated. If, for example, ϵ_i captures many independent variables that cannot be observed by the researcher, then normality of ϵ_i can be established by central limit theorems (Winkelmann 2008). Assuming that ϵ_i is normally distributed (that is, ζ_i is log-normally-distributed), Gauss–Hermite quadrature can be used to approximate the likelihood. Consider a change of variable $\nu_i = \epsilon_i/\sqrt{2}\sigma$:

$$P^+(y_i|\mathbf{x}_i) = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} P^+(y_i|\mathbf{x}_i, \sqrt{2}\sigma\nu_i) \exp(-\nu_i^2) d\nu_i$$

Written in this form, the integral can be evaluated by Gauss–Hermite quadrature to get rid of the unobserved heterogeneity, and the likelihood function becomes

$$L = \prod_{i=1}^{N} \frac{1}{\sqrt{\pi}} \sum_{r=1}^{R} P^+(y_i|\mathbf{x}_i, \sqrt{2}\sigma\nu_r) w_r$$

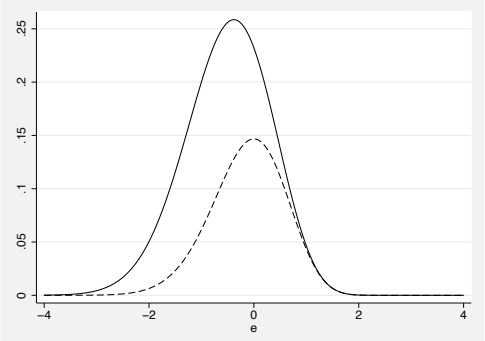
where ν_r and w_r are the nodes and weights for the quadrature.

3 The ztpnm command

3.1 Stata implementation

The zero-truncated Poisson-lognormal model discussed above is implemented in Stata as `ztpnm`. The command enables users to estimate hurdle models based on the zero-truncated Poisson-lognormal distribution using standard or adaptive Gauss–Hermite quadrature. Adaptive quadrature shifts and scales the quadrature points to place them under the peak of the integrand, which most likely improves the approximation (see section 6.3.2 in Skrondal and Rabe-Hesketh [2004] for a detailed discussion). Many Stata commands such as `xtpoisson` implement an approach proposed by Liu and Pierce

(1994). They argue that the mode of the integrand and the curvature at the mode can be used as scaling and shifting factors. Instead of calculating these factors, `ztpnm` uses the corresponding values of the standard (untruncated) Poisson-lognormal model to implement adaptive quadrature. The reason for this approach is a built-in command in Stata that gives the corresponding values for the standard Poisson-lognormal model but not for the zero-truncated model. The integrand, however, is very similar in both models, which indicates that these values are good guesses for the scaling and shifting factors of the zero-truncated model. Figure 1 shows that the integrands are almost identical for higher values of the dependent variable.



(a) y=1, xb=0 (b) y=4, xb=0

Figure 1. Integrands of zero-truncated models (solid curves) and standard models (dashed curves)

3.2 Syntax

`ztpnm` *depvar* [*indepvars*] [*if*] [*in*] [`,` `irr` `nonadaptive` `intpoints(`#`)` `noconstant` `predict(`*newvar*`)` `vce(`*vcetype*`)` `quadcheck` `quadoutput` `vuong` *maximize_options*]

where *depvar* has to be a strictly positive outcome.

3.3 Options

`irr` reports incidence-rate ratios. `irr` is not allowed with `quadcheck`.

`nonadaptive` uses standard Gauss–Hermite quadrature; the default is adaptive quadrature.

`intpoints(#)` chooses the number of points used for the approximation. The default is `intpoints(30)`. The maximum is 195 without `quadcheck` and 146 with `quadcheck`. Generally, a higher number of points leads to a more accurate approximation, but it takes longer to converge. It is highly recommended to check the sensitivity of the results; see `quadcheck`.

`noconstant` suppresses the constant term (intercept) in the model.

`predict(`*newvar*`)` calculates the estimate of the conditional mean of n given $n > 0$; that is, $E(n|n > 0)$, which is $\exp(xb + \epsilon)/\Pr(n > 0|x)$. Gauss–Hermite quadrature is used to calculate the conditional mean.

`vce(`*vcetype*`)` specifies the type of standard error reported, which includes `oim`, `robust`, `cluster` *clustvar*, or `opg`; see [R] ***vce_option***.

`quadcheck` checks the sensitivity of the quadrature approximation by refitting the model with more and fewer quadrature points. `quadcheck` is not allowed with `irr`.

`quadoutput` shows the iteration log and the output of the refitted models.

`vuong` performs a Vuong test of `ztpnm` versus `ztnb`.

maximize_options: `difficult`, `technique(`*algorithm_spec*`)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `nonrtolerance`, and `from(`*init_specs*`)`; see [R] **maximize**. These options are seldom used.

`difficult` is the default.

4 Examples

Santos Silva (2003) points out that in the estimation of mixture models under endogenous sampling, the distribution of the unobserved heterogeneity can be specified in both the actual population and the artificial or truncated population. Independence between the unobservables and the covariables can thus be assumed in both populations. The choice between these two alternatives is not innocuous, and this seemingly slight difference may lead to substantially different results. In hurdle models, the mechanism generating the positive counts is assumed to be different from the one generating the zeros. The population of interest for the positive counts is the truncated population because only observations in this group carry information on the mechanism that generates the positives. Thus covariables and unobservables should be independent in the truncated population.

The popular hurdle model is based on the negative binomial distribution. In this model, the unobserved heterogeneity follows a gamma distribution where covariables and unobservables are assumed to be independent in the actual population. However, independence in the actual population generally rules out independence in the truncated population. The following example illustrates this problem. Here counts are generated

by a Poisson process[1] where λ is assumed to depend on a covariable x and an unobserved variable e. Although x and e are orthogonal in the actual population, they are correlated in the truncated population. The reason is that given a certain value of x, it is more likely to get truncated if the unobserved individual effect is lower than average. As a consequence, x and e are correlated in the truncated population.

```
. set obs 3000
obs was 0, now 3000
. set seed 1234
. generate x = invnormal(runiform())
. generate e = invnormal(runiform())*0.7
. generate lambda = exp(0.5 + 0.5*x + e)
. genpoisson y, mu(lambda)
. pwcorr x e, star(0.1)
             |        x        e
-------------+------------------
           x |   1.0000
           e |   0.0028   1.0000
. drop if y==0
(756 observations deleted)
. pwcorr x e, star(0.001)
             |        x        e
-------------+------------------
           x |   1.0000
           e |  -0.1027*  1.0000
```

In the following Monte Carlo experiment, x and e are orthogonal in the truncated population. Therefore, the estimates of the truncated negative binomial regression applied to the simulated datasets are expected to be different from the true values. See McDowell (2003) for an example of a data-generating process that can be consistently estimated by a zero-truncated negative binomial model. Assume that McDowell's `lambda` additionally contains an appropriate heterogeneity term. The hurdle model described in section 2 assumes that the unobservables are independent of the covariables in the truncated population. This makes it possible to estimate the parameters consistently.

First, results are presented for simulated data, and then results are presented for a sample of the U.S. Medical Expenditure Panel Survey. The artificial count-data variable could be, for example, the number of doctor visits in a certain period. The outcome is generated using the inverse transformation method where the zeros and the positive counts come from different data-generating processes. The program `mc_ztpnm` simulates only the positive counts y. There is one explanatory variable x with parameter $\beta = 0.5$ and a constant of 0.5. The unobserved heterogeneity e is assumed to be normal with a standard deviation of $\sigma = 0.7$. The program simulates the data with `obs()` observations and estimates a regression model of y on x, which has to be specified in `command()`.

1. `genpoisson`, used below, can be obtained from http://www.stata.com/users/rgutierrez/gendist/genpoisson.ado or by typing `findit genpoisson` in Stata.

```
. program define mc_ztpnm, rclass
  1.         syntax, COMmand(string) [obs(integer 3000) Beta(real 0.5)
> CONStant(real 0.5) SIGma(real 0.7) options(string)]
  2.         quietly{
  3.         drop _all
  4.         set obs `obs´
  5.         generate x = invnormal(runiform())
  6.         generate e = invnormal(runiform())*`sigma´
  7.         generate xb = `constant´ + `beta´*x + e
  8.         generate z = runiform()
  9.         generate double fy_cdf=0
 10.         generate y=.
 11.         forvalues k=1/200 {
 12.                 generate double fy`k´ = (exp(xb)^`k´*exp(-exp(xb)))/
> ((1-exp(-exp(xb)))*exp(lnfactorial(`k´)))
 13.                 replace fy_cdf = fy_cdf + fy`k´
 14.                 replace y=`k´ if fy_cdf - fy`k´<z & fy_cdf>z
 15.         }
 16.         }
 17.         `command´ y x, robust `options´
 18. end
```

The `simulate` command is used to replicate `mc_ztpnm` 50 times with 3,000 cross-sectional observations for each replication. The first part of the results, below, shows the estimates from zero-truncated negbin II regressions using `ztnb`, and the second part displays the results from zero-truncated Poisson-lognormal regressions. The average estimate of β from the truncated negbin II models is 0.5834, which is noticeably different from the true value ($\beta = 0.5$). The estimates of `ztnb` are biased if the observed and unobserved variables are independent in the truncated population. The truncated Poisson-lognormal model is estimated using the `ztpnm` command. The default is adaptive quadrature with 30 nodes. The average estimate of β is now almost identical to the true value, and the average log likelihood is highest.

In addition, Vuong (1989) tests are performed to more formally select between both models (`ztnb` and `ztpnm`). Note that the models are overlapping rather than strictly nonnested. The zero-truncated Poisson-lognormal and negbin II models collapse to a zero-truncated Poisson model under the restrictions that σ and α are zero. The Vuong test can only be interpreted if these conditions are rejected. Otherwise, it is not possible to discriminate between the two models. If σ and α are not equal to zero, a test statistic larger than 1.96 indicates that the truncated Poisson-lognormal model is more appropriate. If the value of the statistic is smaller than -1.96, the truncated negbin II model is better. In this example, σ and α are significantly different from zero by definition, and the average value of the Vuong statistic is 1.97, which indicates a better fit of the zero-truncated Poisson-lognormal model.

```
. simulate _b ll=e(ll) N=e(N), reps(50) seed(1234): mc_ztpnm, com(ztnb)
      command:  mc_ztpnm, com(ztnb)
     [_eq3]ll:  e(ll)
      [_eq3]N:  e(N)
Simulations (50)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
. summarize, sep(3)
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       y_b_x |        50    .5834298     .023775   .5415038   .6273451
    y_b_cons |        50    .4490598    .0419869   .3584312   .5225616
lnalpha_b_~s |        50   -.1442036    .1081242  -.4058383   .0702536
-------------+---------------------------------------------------------
     _eq3_ll |        50   -5127.739    71.44369   -5341.06   -4964.11
      _eq3_N |        50        3000           0       3000       3000
. simulate _b ll=e(ll) N=e(N) vuong=e(vuong), reps(50) seed(1234): mc_ztpnm,
> com(ztpnm) options(vuong)
      command:  mc_ztpnm, com(ztpnm) options(vuong)
     [_eq3]ll:  e(ll)
      [_eq3]N:  e(N)
  [_eq3]vuong:  e(vuong)
Simulations (50)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
. summarize, sep(3)
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
     eq1_b_x |        50    .5016896    .0201121   .4668793    .542971
  eq1_b_cons |        50    .4974964    .0277192   .4312099   .5555598
lnsigma_b_~s |        50   -.3553845    .0313651  -.4273075  -.2912054
-------------+---------------------------------------------------------
     _eq3_ll |        50   -5117.823    70.90838  -5324.094  -4952.258
      _eq3_N |        50        3000           0       3000       3000
  _eq3_vuong |        50    1.967658    .6502108   .8654493   3.760262
```

Table 1 displays the simulation results for different values of σ. The higher σ, the further away are the estimates of `ztnb` from the true values. The correlation between x and e in the truncated sample also increases with rising σ. The estimates of `ztpnm` are consistent for all three parameterizations. There is no decision necessary if $\sigma = 0$ because both models collapse to a zero-truncated Poisson model in this case.

Table 1. Simulation results for different values of σ

σ	$\overline{\widehat{\beta}}_{\texttt{ztnb}}$	$\overline{\widehat{\beta}}_{\texttt{ztpnm}}$	$\mathrm{Corr}(x,e)^{+}$
0.3	0.5165	0.5010	−0.0536
0.5	0.5436	0.5004	−0.0856
0.7	0.5834	0.5017	−0.1027

The real-data example analyzes a cross-section sample from the U.S. Medical Expenditure Panel Survey for 2003. The dependent variable is the annual number of doctor visits of individuals 65 years and older (see Cameron and Trivedi [2010] for more information about this dataset). Suppose we are interested in the effect that is attributable to Medicaid insurance on the number of doctor visits. The outcome is analyzed using a hurdle model because there are more zeros in the data than what was predicted by a Poisson count-data model (see figure 2).

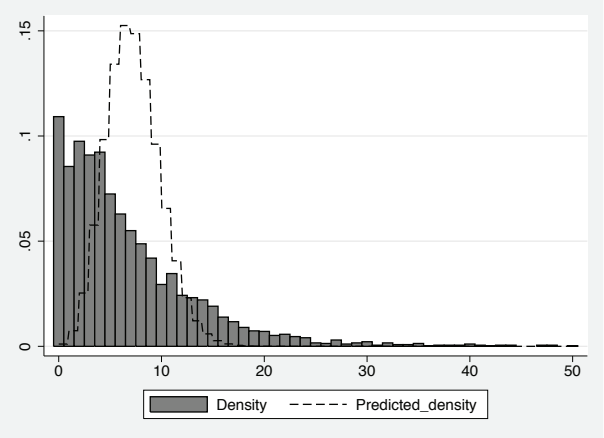


Figure 2. Observed frequencies versus probability mass function (∼ Pois[mean of docvis])

You can think about the demand for health care as a two-part decision process. At first, it is up to the patient to decide whether to visit a doctor. But after the first contact, the physician influences the intensity of treatment (Pohlmeier and Ulrich 1995). Assuming that the error terms of the binary and the truncated models are uncorrelated, the maximization process can be separated. In this case, one can first maximize a binary model with at least one doctor visit as the dependent variable, using the full sample. Second, one can estimate a zero-truncated regression separately using only observations with positive counts.

The following example discusses only the choice of the truncated part. Firstly, a zero-truncated negbin II model is applied. The results indicate a weakly significant increase of 10% in the expected number of doctor visits, conditional on use, which is attributable to Medicaid insurance.

```
. use mus17data
. global xs private medicaid age age2 educyr actlim totchr
. generate ytrunc = docvis
. replace ytrunc = . if ytrunc==0
(401 real changes made, 401 to missing)
```

```
. ztnb ytrunc $xs, robust
  (output omitted)
Zero-truncated negative binomial regression       Number of obs   =       3276
Dispersion     = mean                             Wald chi2(7)    =     474.34
Log likelihood = -9452.899                        Prob > chi2     =     0.0000
```

ytrunc	Coef.	Robust Std. Err.	z	P>|z|	[95% Conf.	Interval]
private	.1095567	.0382086	2.87	0.004	.0346692	.1844442
medicaid	.0972309	.0589629	1.65	0.099	-.0183342	.212796
age	.2719032	.0671328	4.05	0.000	.1403254	.403481
age2	-.0017959	.000445	-4.04	0.000	-.002668	-.0009238
educyr	.0265974	.0050938	5.22	0.000	.0166137	.0365812
actlim	.1955384	.040658	4.81	0.000	.1158503	.2752266
totchr	.2226969	.0135761	16.40	0.000	.1960882	.2493056
_cons	-9.19017	2.517163	-3.65	0.000	-14.12372	-4.256621
/lnalpha	-.5259629	.0544868			-.632755	-.4191708
alpha	.590986	.0322009			.5311265	.6575919

Secondly, a zero-truncated Poisson lognormal model is estimated using the `ztpnm` command. Additionally, the accuracy of the likelihood approximation is checked using `ztpnm`'s `quadcheck` option, which reestimates the model with more and fewer quadrature points. The table below displays the fitted model with 30 quadrature points and, additionally, the results of the reestimated models with 20 and 40 points. The estimates and the likelihoods are almost the same in all three cases, which indicates that the likelihood approximation is good enough to rely on the results. Now the effect that is attributable to Medicaid is around 3% lower than the negbin II results and is no longer significant at the 10% level.

The Vuong test statistic is positive and larger than 1.96, which rejects the truncated negbin II model in favor of the truncated Poisson-lognormal model. Because σ is a boundary parameter, it should be tested separately instead of using the reported t-values. A likelihood-ratio test for $\sigma = 0$ gives $\overline{\chi}^2_{01} = -2(-12998 + 9412) = 7172$. The level of significance is $\Pr(\chi^2_1 > 7172)/2 = 0.000$ (see also Gutierrez, Carter, and Drukker [2001]). The null hypothesis is rejected, which is probably the case in almost all applications.

```
. ztpnm ytrunc $xs, robust vuong quadcheck
Getting starting values from zero-truncated Poisson:
  (output omitted)
Fitting zt-Poisson normal mixture model:
  (output omitted)
Iteration 4:   log pseudolikelihood = -9412.1934
Refitting model intpoints() = 20
Refitting model intpoints() = 40

*****************
Quadrature check:
*****************
```

Variable	qpoints_20	qpoints_30	qpoints_40
eq1			
private	.10347116	.10364263	.10365141
medicaid	.0721706	.07117263	.07111751
age	.26818424	.26861088	.26863663
age2	-.00176665	-.00176924	-.00176939
educyr	.02492124	.02486335	.02486017
actlim	.16483045	.16434532	.16431735
totchr	.22196453	.2220153	.22201903
_cons	-9.2211787	-9.2376583	-9.2386537
lnsigma			
_cons	-.3619317	-.36278788	-.36283427
Statistics			
ll	-9412.1877	-9412.1934	-9412.1935

```
*************
Fitted model:
*************
Number of quadrature points: 30
Zero-truncated Poisson normal mixture model       Number of obs   =       3276
                                                  Wald chi2(7)    =     585.30
Log pseudolikelihood = -9412.1934                 Prob > chi2     =     0.0000
```

ytrunc	Coef.	Robust Std. Err.	z	P>|z|	[95% Conf.	Interval]
private	.1036426	.0330535	3.14	0.002	.038859	.1684262
medicaid	.0711726	.0448151	1.59	0.112	-.0166633	.1590086
age	.2686109	.0577928	4.65	0.000	.1553391	.3818826
age2	-.0017692	.0003837	-4.61	0.000	-.0025214	-.0010171
educyr	.0248634	.0043052	5.78	0.000	.0164253	.0333014
actlim	.1643453	.0342507	4.80	0.000	.0972152	.2314755
totchr	.2220153	.0115839	19.17	0.000	.1993113	.2447193
_cons	-9.237658	2.165506	-4.27	0.000	-13.48197	-4.993345
/lnsigma	-.3627879	.0192127	-18.88	0.000	-.400444	-.3251318
sigma	.695734	.0133669	52.05	0.000	.6700225	.7224321

```
**************
Vuong test of ztpnm vs. ztnb: z =     2.66  Pr>z = 0.0039
```

Finally, the same model is reestimated using nonadaptive quadrature with a very high number of nodes. This, of course, takes much longer to converge. The results are very similar to the estimates from the previous regression that uses adaptive quadrature to approximate the likelihood, but they vary a little bit depending on the number of quadrature points. This indicates that the likelihood approximation is not as accurate as the adaptive approximation.

```
. ztpnm ytrunc $xs, robust intpoints(120) quadcheck nonadaptive
```

(*output omitted*)

```
*****************
Quadrature check:
*****************
```

Variable	qpoints_80	qpoint~120	qpoint~160
eq1			
private	.10250717	.10340271	.10374735
medicaid	.07526089	.07241452	.07044432
age	.26699716	.26804252	.26878583
age2	-.00175964	-.00176579	-.00177027
educyr	.02513045	.02493826	.02482422
actlim	.16622169	.16496933	.16404298
totchr	.2219509	.2219467	.22200684
_cons	-9.1749427	-9.2156541	-9.244254
lnsigma			
_cons	-.35845104	-.36152824	-.36341895
Statistics			
ll	-9411.9925	-9412.1236	-9412.2131

(*output omitted*)

5 Conclusion

Hurdle models based on the zero-truncated Poisson-lognormal distribution are rarely used in applied work, although they incorporate some advantages compared with their negative binomial alternatives. These models are appealing from a theoretical point of view and, additionally, perform much better in many applications. The new Stata `ztpnm` command allows accurate and quick estimation of a zero-truncated Poisson-lognormal model with adaptive quadrature. This command can be used to model the positive counts in a hurdle model.

6 Acknowledgments

I am grateful to Florian Heiss and Rainer Winkelmann for their helpful comments. Financial support by Munich Center of Health Sciences is gratefully acknowledged.

7 References

Cameron, A. C., and P. K. Trivedi. 2010. *Microeconometrics Using Stata*. Rev. ed. College Station, TX: Stata Press.

Gutierrez, R. G., S. Carter, and D. M. Drukker. 2001. sg160: On boundary-value likelihood-ratio tests. *Stata Technical Bulletin* 60: 15–18. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 269–273. College Station, TX: Stata Press.

Liu, Q., and D. A. Pierce. 1994. A note on Gauss–Hermite quadrature. *Biometrika* 81: 624–629.

McDowell, A. 2003. From the help desk: Hurdle models. *Stata Journal* 3: 178–184.

Mullahy, J. 1986. Specification and testing of some modified count data models. *Journal of Econometrics* 33: 341–365.

Pohlmeier, W., and V. Ulrich. 1995. An econometric model of the two-part decision-making process in the demand for health care. *Journal of Human Resources* 30: 339–361.

Santos Silva, J. M. C. 2003. A note on the estimation of mixture models under endogenous sampling. *Econometrics Journal* 6: 46–52.

Skrondal, A., and S. Rabe-Hesketh. 2004. *Generalized Latent Variable Modeling: Multilevel, Longitudinal, and Structural Equation Models*. Boca Raton, FL: Chapman & Hall/CRC.

Vesterinen, J., E. Pouta, A. Huhtala, and M. Neuvonen. 2010. Impacts of changes in water quality on recreation behavior and benefits in Finland. *Journal of Environmental Management* 91: 984–994.

Vuong, Q. H. 1989. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica* 57: 307–333.

Winkelmann, R. 2004. Health care reform and the number of doctor visits—An econometric analysis. *Journal of Applied Econometrics* 19: 455–472.

———. 2008. *Econometric Analysis of Count Data*. 5th ed. Berlin: Springer.

Wong, I. O. L., M. J. Lindner, B. J. Cowling, E. H. Y. Lau, S.-V. Lo, and G. M. Leung. 2010. Measuring moral hazard and adverse selection by propensity scoring in the mixed health care economy of Hong Kong. *Health Policy* 95: 24–35.

About the author

Helmut Farbmacher is a PhD student at the University of Munich with research interest in applied health economics.

The Stata Journal (2011)
11, Number 1, pp. 95–105

Right-censored Poisson regression model

Rafal Raciborski
StataCorp
College Station, TX
rraciborski@stata.com

Abstract. I present the `rcpoisson` command for right-censored count-data models with a constant (Terza 1985, *Economics Letters* 18: 361–365) and variable censoring threshold (Caudill and Mixon 1995, *Empirical Economics* 20: 183–196). I show the effects of censoring on estimation results by comparing the censored Poisson model with the uncensored one.

Keywords: st0219, rcpoisson, censoring, count data, Poisson model

1 Introduction

Models that adjust for censoring are required when the values of the dependent variable are available for a restricted range but the values of the independent variables are always observed; see Cameron and Trivedi (1998) and Winkelmann (2008). For example, a researcher who is interested in alcohol consumption patterns among male college students may define binge drinking as "five or more drinks in one sitting" and may code the dependent variable as 0, 1, 2, ..., 5 or more drinks. In this case, the number of drinks consumed will be censored at five. Some students may have consumed more than five alcoholic beverages, but we will never know. Other examples of censoring include the number of shopping trips (Terza 1985), the number of children in the family (Caudill and Mixon 1995), and the number of doctor visits (Hilbe and Greene 2007).

Applying a traditional Poisson regression model to censored data will produce biased and inconsistent estimates; see Brännäs (1992) for details. Intuitively, when the data are right-censored, large values of the dependent variable are coded as small and the conditional mean of the dependent variable and the marginal effects will be attenuated.

In this article, I introduce the `rcpoisson` command for the estimation of right-censored count data. Section 2 describes the command, section 3 gives an example of cross-border shopping trips, section 4 presents postestimation commands, section 5 presents the results of a simulation study, section 6 presents methods and formulas, section 7 describes saved results, and the conclusion follows.

2 The rcpoisson command

The `rcpoisson` command fits right-censored count data models with a constant (Terza 1985) or a variable censoring threshold (Caudill and Mixon 1995). A variable censoring threshold allows the censoring value to differ across each individual or group—for

instance, in the example above we can add female college students to our study and define binge drinking for females as "3 or more drinks in one sitting".

2.1 Syntax

`rcpoisson` *depvar* [*indepvars*] [*if*] [*in*] [*weight*], `ul`[`(`*#* | *varname*`)`]
[`noconstant` `exposure(`*varname_e*`)` `offset(`*varname_o*`)`
`constraints(`*constraints*`)` `vce(`*vcetype*`)` `level(`*#*`)` `irr` `nocnsreport`
`coeflegend` *display_options* *maximize_options*]

depvar, *indepvars*, *varname_e*, and *varname_o* may contain time-series operators; see [U] **11.4.4 tsvarlist**.

indepvars may contain factor variables; see [U] **11.4.3 fvvarlist**.

`fweight`s, `iweight`s, and `pweight`s are allowed; see [U] **11.1.6 weight**.

`bootstrap`, `by`, `jackknife`, `mi estimate`, `nestreg`, `rolling`, `statsby`, and `stepwise` are allowed; see [U] **11.1.10 prefix**.

Weights are not allowed with the `bootstrap` prefix.

2.2 Options

`ul`[`(`*#* | *varname*`)`] indicates the upper (right) limit for censoring. Observations with *depvar* $\geq$ `ul()` are right-censored. A constant censoring limit is specified as `ul(`*#*`)`, where *#* is a positive integer. A variable censoring limit is specified as `ul(`*varname*`)`; *varname* should contain positive integer values. When the option is specified as `ul`, the upper limit is the maximum value of *depvar*. This is a required option.

`noconstant` suppresses constant terms.

`exposure(`*varname_e*`)` includes ln(*varname_e*) in the model with the coefficient constrained to 1.

`offset(`*varname_o*`)` includes *varname_o* in the model with the coefficient constrained to 1.

`constraints(`*constraints*`)` applies specified linear constraints.

`vce(`*vcetype*`)` specifies the type of standard error reported. *vcetype* may be `oim` (the default), `robust`, or `cluster` *clustvar*.

`level(`*#*`)` sets the confidence level. The default is `level(95)`.

`irr` reports incidence-rate ratios.

`nocnsreport` suppresses the display of constraints.

`coeflegend` displays a coefficient legend instead of a coefficient table.

display_options control spacing and display of omitted variables and base and empty cells. The options include `noomitted`, `vsquish`, `noemptycells`, `baselevels`, and `allbaselevels`; see [R] **estimation options**.

maximize_options control the maximization process. They are seldom used. The options include `difficult`, `technique(`*algorithm_spec*`)`, `iterate(#)`, `[no]log`, `trace`, `gradient`, `showstep`, `hessian`, `showtolerance`, `tolerance(#)`, `ltolerance(#)`, `nrtolerance(#)`, `nonrtolerance`, and `from(`*init_specs*`)`; see [R] **maximize** for details. `from()` must be specified as a vector, for example, `mat b0 = (0,0,0,0)`, `rcpoisson ... from(b0)`.

2.3 Saved results

The censoring threshold will be returned in `e(ulopt)`. If a variable censoring threshold was specified, the macro will contain the name of the censoring variable; otherwise, the macro will contain the user-specified censoring value or the maximum value of the dependent variable.

Scalars

`e(N)`	number of observations
`e(N_rc)`	number of right-censored observations
`e(N_unc)`	number of uncensored observations
`e(k)`	number of parameters
`e(k_eq)`	number of equations
`e(k_eq_model)`	number of equations in model Wald test
`e(k_dv)`	number of dependent variables
`e(k_autoCns)`	number of base, empty, and omitted constraints
`e(df_m)`	model degrees of freedom
`e(r2_p)`	pseudo-R-squared
`e(ll)`	log likelihood
`e(ll_0)`	log likelihood, constant-only model
`e(N_clust)`	number of clusters
`e(chi2)`	χ^2 statistic
`e(p)`	significance
`e(rank)`	rank of `e(V)`
`e(ic)`	number of iterations
`e(rc)`	return code
`e(converged)`	`1` if converged, `0` otherwise

Macros

`e(cmd)`	`rcpoisson`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(ulopt)`	contents of `ul()`
`e(wtype)`	weight type
`e(wexp)`	weight expression
`e(title)`	title in estimation output
`e(clustvar)`	name of cluster variable
`e(offset)`	offset
`e(chi2type)`	`Wald` or `LR`; type of model chi-squared test
`e(vce)`	*vcetype* specified in `vce()`
`e(vcetype)`	title used to label Std. Err.
`e(opt)`	type of optimization
`e(which)`	`max` or `min`; whether optimizer is to perform maximization or minimization
`e(ml_method)`	type of `ml` method
`e(user)`	name of likelihood-evaluator program
`e(technique)`	maximization technique
`e(singularHmethod)`	`m-marquardt` or `hybrid`; method used when Hessian is singular
`e(crittype)`	optimization criterion
`e(properties)`	`b V`
`e(estat_cmd)`	program used to implement `estat`
`e(predict)`	program used to implement `predict`
`e(asbalanced)`	factor variables `fvset` as `asbalanced`
`e(asobserved)`	factor variables `fvset` as `asobserved`

Matrices

`e(b)`	coefficient vector
`e(Cns)`	constraints matrix
`e(ilog)`	iteration log (up to 20 iterations)
`e(gradient)`	gradient vector
`e(V)`	variance–covariance matrix of the estimators
`e(V_modelbased)`	model-based variance

Functions

`e(sample)`	marks estimation sample

3 Example

To illustrate the effect of censoring, we use data on the frequency of cross-border shopping trips from Slovenia. The respondents were asked about the number of shopping trips they made to another European Union country in the previous 12-month period.[1] Out of 1,025 respondents, 780 made no cross-border shopping trips; 131 made one trip; and 114 made two or more trips. Thus a few more than 11% of observations are right-censored. Table 1 describes each variable used in the analysis.

Table 1. Independent variable description

`female`	1 if female, 0 otherwise
`married`	1 if married, remarried, or living with partner; 0 otherwise
`under15`	number of children under 15 years of age
`age`	1 if 15–24, 2 if 25–39, 3 if 40–54, 4 if 55+
`city`	1 if respondent lives in a large town, 0 otherwise
`car`	1 if respondent owns a car
`internet`	1 if respondent has an Internet connection at home

In Stata, we fit the right-censored Poisson as follows:

```
. use eb
(Eurobarometer 69.1: Purchasing in the EU, February-March 2008)
. rcpoisson trips i.female i.married under15 i.age i.city i.car i.internet, ul
> nolog
Right-censored Poisson regression               Number of obs   =       1025
                                                LR chi2(9)      =     124.95
                                                Prob > chi2     =     0.0000
Log likelihood = -738.19296                     Pseudo R2       =     0.0780
```

trips	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
1.female	.1383446	.1084766	1.28	0.202	-.0742656	.3509547
1.married	.0546732	.1365587	0.40	0.689	-.2129769	.3223234
under15	.1104834	.0698027	1.58	0.113	-.0263275	.2472942
age						
2	-.2044181	.166461	-1.23	0.219	-.5306756	.1218393
3	-.6731548	.1896838	-3.55	0.000	-1.044928	-.3013814
4	-.8328144	.195075	-4.27	0.000	-1.215154	-.4504743
1.city	-.1590896	.1371978	-1.16	0.246	-.4279924	.1098133
1.car	.3853947	.2423583	1.59	0.112	-.0896188	.8604081
1.internet	.6339803	.1576153	4.02	0.000	.32506	.9429007
_cons	-1.454013	.2815106	-5.17	0.000	-2.005763	-.9022622

```
  Observation summary:      114 right-censored observations (11.1 percent)
                            911     uncensored observations
```

1. We use question QC2_1 from the Eurobarometer 69.1, Inter-university Consortium for Political and Social Research, Study No. 25163.

In this case, the `ul` option is equivalent to `ul(2)`—`ul` with no argument tells Stata to treat the maximum value of the dependent variable as the censoring value.

The interpretation of parameters in the censored Poisson model is exactly the same as in the uncensored model. For example, the frequency of trips for people who have an Internet connection at home is `exp(.634)` or 1.89 times larger than for those with no Internet connection. Those rates can be obtained by specifying the `irr` option. Alternatively, we can calculate the percent change in the number of expected trips, which is `(exp(.634)-1)*100` or 89%.

We can compare the censored model with the uncensored one:[2]

```
. poisson trips i.female i.married under15 i.age i.city i.car i.internet, nolog
Poisson regression                              Number of obs   =       1025
                                                LR chi2(9)      =     116.04
                                                Prob > chi2     =     0.0000
Log likelihood = -756.63717                     Pseudo R2       =     0.0712
```

trips	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
1.female	.1309279	.1081285	1.21	0.226	-.0810001	.3428559
1.married	.0524751	.1361917	0.39	0.700	-.2144556	.3194059
under15	.0998008	.0698594	1.43	0.153	-.0371211	.2367228
age						
2	-.179913	.1660251	-1.08	0.279	-.5053162	.1454901
3	-.6324862	.1888077	-3.35	0.001	-1.002542	-.2624299
4	-.788986	.1944563	-4.06	0.000	-1.170113	-.4078586
1.city	-.1525254	.13705	-1.11	0.266	-.4211385	.1160876
1.car	.3942109	.2417194	1.63	0.103	-.0795505	.8679723
1.internet	.6157459	.1574147	3.91	0.000	.3072188	.924273
_cons	-1.517786	.2811689	-5.40	0.000	-2.068867	-.9667054

As can be seen, all coefficients but one returned by uncensored Poisson are smaller than those of the censored Poisson, and so are the marginal effects. The estimate for `internet` falls slightly to 0.616, which translates to an 85% increase in the number of expected trips for respondents with an Internet connection at home.

4 Model evaluation

`rcpoisson` supports all the postestimation commands available to `poisson`, including `estat` and `predict`. The only difference is that, by default, `predict` returns the predicted number of events from the right-censored Poisson distribution. If you want to obtain the predicted number of events from the underlying uncensored distribution, specify the `np` option.

2. Equivalently, you can use `rcpoisson` with a `ul()` value greater than the maximum value of the dependent variable. Point estimates will be identical to those obtained with `poisson`, but if you use `margins`, the marginal effects will differ; therefore, I do not advise using `rcpoisson` for the estimation of uncensored models.

Continuing with the censored example above, we can obtain predicted values and marginal effects by typing

```
. predict n
(option n assumed; predicted number of events)
. margins
Predictive margins                                Number of obs   =       1025
Model VCE    : OIM
Expression   : Predicted number of events, predict()
```

	Margin	Delta-method Std. Err.	z	P>|z|	[95% Conf.	Interval]
_cons	.3563163	.0176632	20.17	0.000	.321697	.3909356

```
. margins, dydx(_all)
Average marginal effects                          Number of obs   =       1025
Model VCE    : OIM
Expression   : Predicted number of events, predict()
dy/dx w.r.t. : 1.female 1.married under15 2.age 3.age 4.age 1.city 1.car
               1.internet
```

	dy/dx	Delta-method Std. Err.	z	P>|z|	[95% Conf.	Interval]
1.female	.0458264	.0356648	1.28	0.199	-.0240753	.1157281
1.married	.0181764	.04526	0.40	0.688	-.0705315	.1068843
under15	.0368447	.02326	1.58	0.113	-.008744	.0824335
age						
2	-.0929382	.0774597	-1.20	0.230	-.2447564	.0588801
3	-.2549375	.0769697	-3.31	0.001	-.4057953	-.1040797
4	-.2963611	.0767083	-3.86	0.000	-.4467065	-.1460157
1.city	-.0510675	.0423248	-1.21	0.228	-.1340225	.0318875
1.car	.1124467	.0612255	1.84	0.066	-.0075531	.2324465
1.internet	.1896224	.0423883	4.47	0.000	.1065428	.2727019

```
Note: dy/dx for factor levels is the discrete change from the base level.
```

Thus having an Internet connection at home increases the expected number of cross-border trips by 0.19, holding all the other variables constant.

5 Simulation study

To compare the uncensored and censored Poisson models, we perform a simulation study. True values of the dependent variable y, obtained from a Poisson distribution, are censored at two constant points, and we attempt to recover the true parameters used in the data-generating process. The variables x_1 and x_2 are generated as uncorrelated standard normal variates, and we fix the values of β_1 and β_2 at 1 and -1, respectively. We perform 1,000 replications on sample sizes of 250, 500, and 1,000, and we choose the censoring constants such that the percentages of censored y-values are roughly 10% and 39%. Tables 2 and 3 present the results:

Table 2. Simulation results with about 10% censoring

		Mean		Std. Dev.		Std. Err.		Rej. Rate	
	n	poi	rcpoi	poi	rcpoi	poi	rcpoi	poi	rcpoi
β_{x_1}	250	0.700	1.004	0.072	0.069	0.048	0.069	0.994	0.036
	500	0.694	1.003	0.054	0.048	0.033	0.048	1.00	0.050
	1000	0.689	1.001	0.036	0.034	0.024	0.034	1.00	0.052
β_{x_2}	250	−0.698	−1.003	0.071	0.070	0.048	0.070	0.995	0.036
	500	−0.693	−1.002	0.052	0.049	0.034	0.048	1.00	0.050
	1000	−0.689	−1.000	0.036	0.034	0.023	0.034	1.00	0.045

`poi` and `rcpoi` stand for uncensored and censored Poisson, respectively.

Table 3. Simulation results with about 39% censoring

		Mean		Std. Dev.		Std. Err.		Rej. Rate	
	n	poi	rcpoi	poi	rcpoi	poi	rcpoi	poi	rcpoi
β_{x_1}	250	0.456	1.010	0.053	0.110	0.065	0.110	1.00	0.044
	500	0.453	1.006	0.038	0.083	0.045	0.077	1.00	0.075
	1000	0.453	1.004	0.025	0.053	0.032	0.054	1.00	0.048
β_{x_2}	250	−0.456	−1.012	0.052	0.109	0.064	0.110	1.00	0.039
	500	−0.455	−1.007	0.037	0.081	0.045	0.077	1.00	0.060
	1000	−0.453	−1.004	0.025	0.053	0.032	0.054	1.00	0.045

`poi` and `rcpoi` stand for uncensored and censored Poisson, respectively.

The "Mean" columns report the average of the estimated coefficients over 1,000 simulation runs. The "Std. Dev." columns report the standard deviation of the estimated coefficients, while the "Std. Err." columns report the mean of the standard error of the true parameters. Finally, the "Rej. Rate" columns report the rate at which the true null hypothesis was rejected at the 0.05 level. With the exception of one run, the coverage

is effectively 95% or better for the censored Poisson model. The coverage for the uncensored Poisson model is essentially zero. As can be seen, the bias for the uncensored Poisson model is substantial, even with a small amount of censoring.

6 Methods and formulas

The basics of the censored Poisson model are presented in Cameron and Trivedi (1998) and Winkelmann (2008). Here we assume that the censoring mechanism is independent of the count variable (see Brännäs [1992]).

Consider the probability function of the Poisson random variable:

$$f(y_i;\mu_i) = \frac{e^{-\mu_i}\mu_i^{y_i}}{y_i!}, \qquad i = 1,\dots,n$$

where $\mu_i = \exp(\mathbf{x}_i\boldsymbol{\beta})$, $\mathbf{x}_i$ is a vector of exogenous variables, and $\boldsymbol{\beta}$ is a vector of unknown parameters. In a traditional Poisson setting, we observe all y_i exactly; however, in a censored Poisson model, we observe the true $y_i^\star$ only below a censoring point c_i. Thus

$$y_i = \begin{cases} y_i^\star, & \text{if } y_i^\star < c_i \\ c_i, & \text{if } y_i^\star \geq c_i \end{cases}$$

The censoring point c_i can vary for each observation (Caudill and Mixon 1995). If c is a constant, we have a model with a constant censoring threshold (Terza 1985).

If y_i is censored, we know that

$$\Pr(y_i \geq c_i) = \sum_{j=c_i}^{\infty} \Pr(y_i = j) = \sum_{j=c_i}^{\infty} f(j) = 1 - \sum_{j=0}^{c_i-1} f(j) = 1 - F(c_i - 1)$$

We define an indicator variable d_i such that

$$d_i = \begin{cases} 1, & \text{if } y_i^\star \geq c_i \\ 0, & \text{otherwise} \end{cases}$$

Then the log likelihood function of the sample can be written as

$$\begin{aligned}\mathcal{L}(\boldsymbol{\beta}) &= \log\left[\prod_{i=1}^{n}\left\{f(y_i)\right\}^{1-d_i}\left\{1-F(c_i-1)\right\}^{d_i}\right] \\ &= \sum_{i=1}^{n}\left[(1-d_i)\log f(y_i) + d_i \log\left\{1-F(c_i-1)\right\}\right]\end{aligned}$$

The gradient is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n}\left\{(1-d_i)(y_i-\mu_i) - d_i\, c_i\, \phi_i\right\}\mathbf{x}_i'$$

and the Hessian is

$$\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\beta}^2} = -\sum_{i=1}^{n}\left[(1-d_i)\,\mu_i - d_i\, c_i\, \left\{(c_i-\mu_i)\,\phi_i - c_i\,{\phi_i}^2\right\}\right]\mathbf{x}_i'\mathbf{x}_i$$

where $\phi_i = f(c_i)/1-F(c_i-1)$.

Estimation is by maximum likelihood. The initial values are taken from the uncensored Poisson model unless the user provides initial values using `from()`.

The conditional mean is given by

$$E\left(y_i;\mu_i\right) = c_i - \sum_{j=0}^{c_i-1} f(j)(c_i-j)$$

Numerical derivatives of the conditional mean are used for the calculation of the marginal effects and their standard errors.

7 Conclusion

In this article, I introduced the `rcpoisson` command for censored count data. I illustrated the usage on censored survey responses and provided a comparison with the uncensored Poisson model. I showed, through simulation, that the uncensored Poisson model is unable to recover the true values of the parameters from the underlying distribution—something the censored Poisson model was quite successful with.

8 Acknowledgments

I am grateful to David Drukker for his guidance and support. I also thank Jeff Pitblado for his comments and suggestions. Any remaining errors and omissions are mine.

9 References

Brännäs, K. 1992. Limited dependent Poisson regression. *Statistician* 41: 413–423.

Cameron, A. C., and P. K. Trivedi. 1998. *Regression Analysis of Count Data*. Cambridge: Cambridge University Press.

Caudill, S. B., and F. G. Mixon, Jr. 1995. Modeling household fertility decisions: Estimation and testing of censored regression models for count data. *Empirical Economics* 20: 183–196.

Hilbe, J. M., and W. H. Greene. 2007. Count response regression models. In *Handbook of Statistics 27: Epidemiology and Medical Statistics*, ed. C. R. Rao, J. P. Miller, and D. C. Rao, 210–252. Amsterdam: Elsevier.

Terza, J. V. 1985. A Tobit-type estimator for the censored Poisson regression model. *Economics Letters* 18: 361–365.

Winkelmann, R. 2008. *Econometric Analysis of Count Data*. 5th ed. Berlin: Springer.

About the author

Rafal Raciborski is an econometrician at StataCorp. In the summer of 2009, he worked as an intern at StataCorp. He produced this project during his internship.

The Stata Journal (2011)
11, Number 1, pp. 106–119

Stata utilities for geocoding and generating travel time and travel distance information

Adam Ozimek
Econsult Corporation
Philadelphia, PA
ozimek@econsult.com

Daniel Miles
Econsult Corporation
Philadelphia, PA
miles@econsult.com

Abstract. This article describes `geocode` and `traveltime`, two commands that use Google Maps to provide spatial information for data. The `geocode` command allows users to generate latitude and longitude for various types of locations, including addresses. The `traveltime` command takes latitude and longitude information and finds travel distances between points, as well as the time it would take to travel that distance by either driving, walking, or using public transportation.

Keywords: dm0053, geocode, traveltime, Google Maps, geocoding, ArcGIS

1 Introduction

The location and spatial interaction of data has long been important in many scientific fields, from the social sciences to environmental and natural sciences. The increased use and availability of geographic information systems (GIS) software has allowed researchers in a growing range of disciplines to incorporate space in their models. In addition, businesses, governments, and the nonprofit sector have found spatial information useful in their analysis.

A crucial first step in incorporating space into analysis is to identify the spatial location of the data. For data that represent points on roads, one way to do this is to "geocode" observations. Geocoding is the process of converting addresses or locations (such as "3600 Market Street, Philadelphia, PA" or simply "Philadelphia, PA") into geographic coordinates (such as [39.95581, −75.19466] or [39.95324, −75.16739]). Once the geographic coordinates are known, the data can be mapped and spatial relationships between data points can be incorporated into analyses.

Geocoding can easily be accomplished using ArcGIS or similar high-end mapping software. However, for those users who are unfamiliar with GIS, this process requires a substantial investment of time to learn the basics of the software and its geocoding capabilities, and often the software's expense is prohibitive.

Alternatively, one can geocode using Google's free mapping website, Google Maps.[1] However, the website is designed to be used for one or two addresses at a time, and using it would be an extremely time-consuming way to geocode and find distances between more than a handful of points. The commands discussed in this article combine the convenience of a high-end software package, like ArcGIS, with the free services of Google Maps.

The `geocode` command automates the geocoding service that is included in the Google Geocoding API (application programming interface) to easily and quickly batch geocode a small set of addresses. The `traveltime` command uses Google Maps to calculate the travel time between a set of geographic coordinates (latitude and longitude) using a variety of transportation modes. The use of these commands in tandem will allow researchers who are unfamiliar with the use of GIS or those without access to GIS the ability to quickly and easily incorporate spatial interactions into their research.

2 The geocode command

2.1 Syntax

`geocode,` [`address(`*varname*`)` `city(`*varname*`)` `state(`*varname*`)` `zip(`*varname*`)` `fulladdr(`*varname*`)`]

See *Remarks* for details on specifying options.

2.2 Options

`address(`*varname*`)` specifies the variable containing a street address. *varname* must be a string. Cleaned address names will provide better results, but the program performs some basic cleaning.

`city(`*varname*`)` specifies the variable containing the name or common abbreviation for the city, town, county, metropolitan statistical area (MSA),[2] or equivalent. *varname* must be a string.

`state(`*varname*`)` specifies the variable containing the name or the two-letter abbreviation of the state of the observation. An example of such an abbreviation is Pa for Pennsylvania. *varname* must be a string.

1. http://maps.google.com
2. MSA refers to a geographic entity defined by the United States Office of Management and Budget for use by federal statistical agencies in collecting, tabulating, and publishing federal statistics. An MSA contains a core urban area of 50,000 or more population. Each MSA consists of one or more counties that contain the core urban area as well as any adjacent counties that have a high degree of social and economic integration with the urban core.

`zip(`*varname*`)` specifies the variable containing the standard United States Postal Service 5-digit postal zip code[3] or zip code +4. If zip code +4 is specified, it should be in the form 12345-6789. *varname* must be a string.

`fulladdr(`*varname*`)` allows users to specify all or some of the above options in a single string. *varname* must be a string and should be in a format that would be used to enter an address using http://maps.google.com. Standard formats are listed in section 2.3.

2.3 Remarks

When geocoding within the United States, one or all of the options `address(`*varname*`)`, `city(`*varname*`)`, `state(`*varname*`)`, and `zip(`*varname*`)` may be specified, with more information allowing for a higher degree of geocoding detail. This allows for the geocoding of zip codes, counties, cities, or other geographic areas. In general, when a specific street address is not specified, a latitude and longitude will be provided for a central location within the specified city, state, or zip code. The same option for specifying geographic detail applies using `fulladdr()`.

When geocoding outside the United States, the `fulladdr()` option must be used and the country must be specified. When inputting data using `fulladdr()`, any string that would be usable with http://maps.google.com is in an acceptable format. Acceptable examples for `fulladdr()` in the United States include but are not limited to these formats:

"street address, city, state, zip code"
"street address, city, state"
"city, state, zip code"
"city, state"
"state, zip code"
"state"

Acceptable examples for `fulladdr()` outside the United States include but are not limited to these formats:

"street address, city, state, country"
"street address, city, country"
"city, state, country"
"city, country"

Country should be specified using the full country name. State can be whatever regional entity exists below the country level—for instance, Canadian provinces or Japanese prefectures. Again, format acceptability may be gauged using the Google Maps website.

The `geocode` command queries Google Maps, which allows for a fair degree of tolerance in how addresses can be entered and still be geocoded correctly. The inputs are

3. Zip codes are postal codes used by the United States Postal Service, the independent government agency that provides postal service in the United States.

not case sensitive and are robust to a wide range of abbreviations and spelling errors. For instance, each of the following would be an acceptable way to enter the same street address:

"123 Fake Street"
"123 Fake St."
"123 fake st"

Common abbreviations for cities, states, towns, counties, and other relevant geographies are also often acceptable. For instance, it is fine to use "Phila" for "Philadelphia", "PA" for "Pennsylvania", "NYC" for "New York City", and "UK" for "United Kingdom". The program is also fairly robust to spelling errors; it is capable of accepting "Allantown, PA" for "Allentown, PA". It is recommended that addresses be as accurate as possible to avoid geocoding errors, but the program is as flexible as Google Maps.

The `geocode` command generates four new variables: `geocode`, `geoscore`, `latitude`, and `longitude`. `latitude` and `longitude` contain the geocoded coordinates for each observation in decimal degrees. The `geocode` variable contains a numerical indicator of geocoding success or type of failure, and `geoscore` provides a measure of accuracy. These values and their definitions are provided by Google Maps. For more information, see http://code.google.com/apis/maps/documentation/geocoding/.

`geocode` error definitions:

200 = no errors
400 = incorrectly specified address
500 = unknown failure reason
601 = no address specified
602 = unknown address
603 = address legally or contractually ungeocodable
620 = Google Maps query limit reached

`geoscore` accuracy level:

0 = unknown accuracy
1 = country-level accuracy
2 = region-level (state, province, etc.) accuracy
3 = subregion-level (county, municipality, etc.) accuracy
4 = town-level (city, village, etc.) accuracy
5 = postal code–level (zip code) accuracy
6 = street-level accuracy
7 = intersection-level accuracy
8 = address-level accuracy
9 = premise-level (building name, property name, store name, etc.) accuracy

Google Maps appears to limit the number of queries allowed from a single Internet protocol (IP) address within a 24-hour period. This exact limit is not known, but it is not recommended that more than 10,000 to 15,000[4] observations be geocoded at any one time from a single IP address.

Data acquired using `geocode` are subject to Google's terms of service, specified here: http://code.google.com/apis/maps/terms.html.

2.4 Standard geocoding example

Start with, for example, a dataset of survey respondent addresses for which an analyst wishes to retrieve latitude and longitude coordinates. The data are as follows:

id	resp_street	resp_city	resp_st	resp_zp
1	1500 Market St	Philadelphia	PA	19102
2	2124 Fairmount Ave	Philadelphia	PA	19130
3	2600 Benjamin Franklin Pkwy	Philadelphia	PA	19130
4	1219 S 9th St	Philadelphia	PA	19147
5	420 Chestnut St	Philadelphia	PA	19106
6	8500 Essington Ave	Philadelphia	PA	19153
7	3600 Market St	Philadelphia	PA	19104
8	1455 Franklin Mills Circle	Philadelphia	PA	19154
9	1901 Vine Street	Philadelphia	PA	19103
10	1801 N Broad St	Philadelphia	PA	19122

The analyst can geocode these data using the following command:

```
. geocode, address(resp_street) city(resp_city) state(resp_st) zip(resp_zp)
Geocoding 1 of 10
Geocoding 2 of 10
Geocoding 3 of 10
Geocoding 4 of 10
Geocoding 5 of 10
Geocoding 6 of 10
Geocoding 7 of 10
Geocoding 8 of 10
Geocoding 9 of 10
Geocoding 10 of 10
```

Upon completion, the geocoded data will include four extra variables, as follows:

4. We have successfully geocoded over 15,000 observations on several occasions, but to be conservative, we do not recommend attempting to geocode more than 15,000 at any one time.

id	resp_street	resp_city	resp_st	resp_zp
1	1500 Market St	Philadelphia	PA	19102
2	2124 Fairmount Ave	Philadelphia	PA	19130
3	2600 Benjamin Franklin Pkwy	Philadelphia	PA	19130
4	1219 S 9th St	Philadelphia	PA	19147
5	420 Chestnut St	Philadelphia	PA	19106
6	8500 Essington Ave	Philadelphia	PA	19153
7	3600 Market St	Philadelphia	PA	19104
8	1455 Franklin Mills Circle	Philadelphia	PA	19154
9	1901 Vine Street	Philadelphia	PA	19103
10	1801 N Broad St	Philadelphia	PA	19122

geocode	geoscore	latitude	longitude
200	8	39.95239	-75.16619
200	8	39.96706	-75.17309
200	8	39.96561	-75.18099
200	8	39.93365	-75.15895
200	8	39.94889	-75.14804
200	8	39.89513	-75.22896
200	8	39.95581	-75.19466
200	8	40.09012	-74.95904
200	8	39.9593	-75.1711
200	8	39.98027	-75.15705

The `geocode` score of 200 indicates no errors in geocoding, and the `geoscore` score of 8 indicates that the geocoding was completed at address-level accuracy.

2.5 Using the fulladdr() option

The previous example showed how to geocode when the address, city, state, and zip were each specified in separate string variables. Alternatively, the full address could appear as a single string variable, as shown in the following output:

id	resp_addr
1	1500 Market St, Philadelphia, PA 19102
2	2124 Fairmount Ave, Philadelphia, PA 19130
3	2600 Benjamin Franklin Pkwy, Philadelphia, PA 19130
4	1219 S 9th St, Philadelphia, PA 19147
5	420 Chestnut St, Philadelphia, PA 19106
6	8500 Essington Ave, Philadelphia, PA 19153
7	3600 Market St, Philadelphia, PA 19104
8	1455 Franklin Mills Circle, Philadelphia, PA 19154
9	1901 Vine Street, Philadelphia, PA 19103
10	1801 N Broad St, Philadelphia, PA 19122

These data, which refer to the exact same locations as those in section 2.4, can be geocoded using the following command:

```
geocode, fulladdr(resp_addr)
```

The coordinates, `geocode`s, and `geoscore`s that are produced are identical to those produced in the previous example.

2.6 Geocoding larger geographical areas

Rather than being concerned with specific street addresses, an analyst might be concerned with the geographic location of particular zip codes, cities, counties, or even states. `geocode` can calculate the latitude and longitude of these geographic areas using Google Maps. Google does not explicitly state how the "centers" of these locations are determined, though in general, it appears that central downtown areas are used for cities and towns, and geographic centroids are used for zip codes, states, and other larger regions. If, for example, the analyst needed to know the latitudes and longitudes corresponding to the centers of the zip codes for the 10 previously used observations, the following command could be issued:

```
. geocode, state(resp_st) zip(resp_zp)
Geocoding 1 of 10
Geocoding 2 of 10
Geocoding 3 of 10
Geocoding 4 of 10
Geocoding 5 of 10
Geocoding 6 of 10
Geocoding 7 of 10
Geocoding 8 of 10
Geocoding 9 of 10
Geocoding 10 of 10
```

This command produces the following dataset of coordinates, `geocode`s, and `geoscore`s:

id	resp_street	resp_city	resp_st	resp_zp
1	1500 Market St	Philadelphia	PA	19102
2	2124 Fairmount Ave	Philadelphia	PA	19130
3	2600 Benjamin Franklin Pkwy	Philadelphia	PA	19130
4	1219 S 9th St	Philadelphia	PA	19147
5	420 Chestnut St	Philadelphia	PA	19106
6	8500 Essington Ave	Philadelphia	PA	19153
7	3600 Market St	Philadelphia	PA	19104
8	1455 Franklin Mills Circle	Philadelphia	PA	19154
9	1901 Vine Street	Philadelphia	PA	19103
10	1801 N Broad St	Philadelphia	PA	19122

geocode	geoscore	latitude	longitude
200	5	39.9548	-75.1656
200	5	39.96883	-75.17586
200	5	39.96883	-75.17586
200	5	39.93567	-75.15173
200	5	39.9493	-75.14471
200	5	39.89152	-75.22866
200	5	39.96157	-75.19677
200	5	40.09333	-74.98056
200	5	40.03131	-75.1698
200	5	39.97274	-75.1246

The `geocode` of 200 indicates that there were no errors and the geocoding was performed correctly. The `geoscore` of 5 indicates that the observations were geocoded at zip code–level accuracy, as expected. Notice that the second and third observations are in the same zip code, and so their latitudes and longitudes are identical.

3 The traveltime command

The geographic distance between data points is often important information in applied research. When geographic coordinates are known, estimating straight-line distance between points can be done using either simple Euclidean distance or a more complex formula, such as the Haversine formula, that takes the curvature of the earth into consideration. However, accurately estimating the driving distance (rather than the straight line or as-the-crow-flies distance) is complicated. Some of the factors that must be taken into consideration include the available network of streets, one-way streets, and the shortest choice among alternative routes.

Even more complicated to estimate than driving distance is driving time. An accurate measure of this includes traffic congestion, speed limits, turning time, and stop signs and traffic lights. The problem is exponentially magnified when mode choice—that is, traveling by car versus taking public transportation—is taken into consideration.

These difficulties explain why straight-line distance is an often-used shortcut. However, real driving time or travel time can be integral in many applications, and a straight-line method often introduces errors that can be correlated with other important variables. For instance, in some urban areas, road and traffic congestion density may be inversely correlated with resident income because lower income people are more likely to live in more densely populated areas. Therefore, if an estimate of the impact of income on an individual's willingness to travel used straight-line distance, that estimate would be biased downward. Moreover, drivers have to travel more road miles and travel for more time to drive a straight-line mile in a city than in a suburb or rural area.

A market study that used straight-line distance to estimate which zip codes lie within the catchment area of a particular store location would overestimate the number of urban zip codes and underestimate the number of suburban or rural zip codes. This is particularly problematic because urban, rural, and suburban zip codes may have differences in average demographics, which would bias estimates in ways that are critically altering to the market study.

3.1 Syntax

The `traveltime` command is designed to work in tandem with the `geocode` command discussed above. The `traveltime` command uses the following syntax:

`traveltime, start_x(`*varname*`) start_y(`*varname*`) end_x(`*varname*`)`
`end_y(`*varname*`)` [`mode(`*varname*`) km`]

3.2 Options

`start_x(`*varname*`)` specifies the variable containing the geocoded x coordinate of the starting point. `start_x()` is required.

`start_y(`*varname*`)` specifies the variable containing the geocoded y coordinate of the starting point. `start_y()` is required.

`end_x(`*varname*`)` specifies the variable containing the geocoded x coordinate of the destination. `end_x()` is required.

`end_y(`*varname*`)` specifies the variable containing the geocoded y coordinate of the destination. `end_y()` is required.

`mode(`*varname*`)` specifies the mode choice of the trip. The values are set to `1` for car, `2` for public transportation, and `3` for walking. The default mode is car.

`km` specifies that `traveltime_dist` be reported in kilometers rather than in miles (the default).

3.3 Remarks

Both starting coordinates (`start_x()`, `start_y()`) and ending coordinates (`end_x()`, `end_y()`) are required inputs. `traveltime` queries Google Maps, which requires that the coordinates be in decimal degrees. We suggest using the `geocode` command to geocode both the start and the end points of the trip before proceeding with the `traveltime` command. Beginning with `geocode` will ensure that the coordinates are in the correct format for Google Maps and will reduce the possibility for errors. However, the use of the `geocode` command is not necessary if the start and end coordinates are known and are reported in decimal degrees.

The `mode()` option is optional. The availability of this option is limited by Google Maps, which does not provide the multiple mode choices for all geographic areas. This especially pertains to the public transportation option, which is currently available in only a limited number of places, mainly in the United States.

The `traveltime` command generates four new variables: `days`, `hours`, `mins`, and `traveltime_dist`. The combination of the `days`, `hours`, and `mins` variables contains the days, hours, and minutes portion of the time that it takes to travel between the origin and the destination. The `traveltime_dist` variable contains the distance between the starting and ending points.

3.4 Standard traveltime example

Suppose you need to calculate the travel time between Philadelphia, PA, and other major cities in Pennsylvania. After using the `geocode` command, the data might look like this:

start_city	end_city	start_~g	start_lat	end_long	end_lat
Philadelphia, PA	Allentown, PA	39.95234	-75.16379	40.60843	-75.49018
Philadelphia, PA	York, PA	39.95234	-75.16379	39.9626	-76.72775
Philadelphia, PA	Lancaster, PA	39.95234	-75.16379	40.03788	-76.30551
Philadelphia, PA	Harrisburg, PA	39.95234	-75.16379	40.2737	-76.88441
Philadelphia, PA	Pittsburgh, PA	39.95234	-75.16379	40.44062	-79.99589
Philadelphia, PA	Erie, PA	39.95234	-75.16379	42.12922	-80.08506
Philadelphia, PA	Scranton, PA	39.95234	-75.16379	41.40897	-75.66241
Philadelphia, PA	Wilkes-Barre, PA	39.95234	-75.16379	41.24591	-75.88131
Philadelphia, PA	Johnstown, PA	39.95234	-75.16379	40.32674	-78.92197
Philadelphia, PA	Reading, PA	39.95234	-75.16379	40.33565	-75.92687

You can calculate the travel time between the start and end points for each observation by using the following command:

```
. traveltime, start_x(start_lat) start_y(start_long) end_x(end_lat)
> end_y(end_long)
Processed 1 of 10
Processed 2 of 10
Processed 3 of 10
Processed 4 of 10
Processed 5 of 10
Processed 6 of 10
Processed 7 of 10
Processed 8 of 10
Processed 9 of 10
Processed 10 of 10
```

Upon completion of the `traveltime` command, the data look as follows:

start_city	end_city	start_~g	start_lat	end_long
Philadelphia, PA	Allentown, PA	39.95234	-75.16379	40.60843
Philadelphia, PA	York, PA	39.95234	-75.16379	39.9626
Philadelphia, PA	Lancaster, PA	39.95234	-75.16379	40.03788
Philadelphia, PA	Harrisburg, PA	39.95234	-75.16379	40.2737
Philadelphia, PA	Pittsburgh, PA	39.95234	-75.16379	40.44062
Philadelphia, PA	Erie, PA	39.95234	-75.16379	42.12922
Philadelphia, PA	Scranton, PA	39.95234	-75.16379	41.40897
Philadelphia, PA	Wilkes-Barre, PA	39.95234	-75.16379	41.24591
Philadelphia, PA	Johnstown, PA	39.95234	-75.16379	40.32674
Philadelphia, PA	Reading, PA	39.95234	-75.16379	40.33565

end_lat	days	hours	mins	travel~t
-75.49018	0	1	9	61.5
-76.72775	0	1	56	102
-76.30551	0	1	32	73
-76.88441	0	1	56	107
-79.99589	0	5	15	305
-80.08506	0	6	43	420
-75.66241	0	2	13	125
-75.88131	0	2	2	113
-78.92197	0	4	13	239
-75.92687	0	1	10	57.6

As illustrated in the above table, it takes one hour and nine minutes to drive the 61.5 miles from Philadelphia, PA, to Allentown, PA; it takes five hours and fifteen minutes from Philadelphia, PA, to Pittsburgh, PA; and it takes six hours and forty-three minutes to drive from Philadelphia, PA, to Erie, PA.

The data in the above example did not have a travel mode specified, so by default, Google Maps calculated the travel time for an automobile trip between the start and end points. If you had data on the transportation mode[5] used for each trip, your data might look like this:

start_city	end_city	start_~g	start_lat	end_long	end_lat	mode
Philadelphia, PA	Allentown, PA	39.95234	-75.16379	40.60843	-75.49018	1
Philadelphia, PA	York, PA	39.95234	-75.16379	39.9626	-76.72775	3
Philadelphia, PA	Lancaster, PA	39.95234	-75.16379	40.03788	-76.30551	3
Philadelphia, PA	Harrisburg, PA	39.95234	-75.16379	40.2737	-76.88441	1
Philadelphia, PA	Pittsburgh, PA	39.95234	-75.16379	40.44062	-79.99589	1
Philadelphia, PA	Erie, PA	39.95234	-75.16379	42.12922	-80.08506	3
Philadelphia, PA	Scranton, PA	39.95234	-75.16379	41.40897	-75.66241	3
Philadelphia, PA	Wilkes-Barre, PA	39.95234	-75.16379	41.24591	-75.88131	3
Philadelphia, PA	Johnstown, PA	39.95234	-75.16379	40.32674	-78.92197	1
Philadelphia, PA	Reading, PA	39.95234	-75.16379	40.33565	-75.92687	1

The syntax for the `traveltime` command would then be

```
. traveltime, start_x(start_lat) start_y(start_long) end_x(end_lat)
> end_y(end_long) mode(mode)
Processed 1 of 10
Processed 2 of 10
Processed 3 of 10
Processed 4 of 10
Processed 5 of 10
Processed 6 of 10
Processed 7 of 10
Processed 8 of 10
Processed 9 of 10
Processed 10 of 10
```

This command produces the following dataset of travel times:

5. We use only the automobile transportation mode and walking. We did not include the public transportation mode in the example because, as mentioned before, Google Maps does not currently support travel times using public transportation in many areas.

start_city	end_city	start_~g	start_lat	end_long
Philadelphia, PA	Allentown, PA	39.95234	-75.16379	40.60843
Philadelphia, PA	York, PA	39.95234	-75.16379	39.9626
Philadelphia, PA	Lancaster, PA	39.95234	-75.16379	40.03788
Philadelphia, PA	Harrisburg, PA	39.95234	-75.16379	40.2737
Philadelphia, PA	Pittsburgh, PA	39.95234	-75.16379	40.44062
Philadelphia, PA	Erie, PA	39.95234	-75.16379	42.12922
Philadelphia, PA	Scranton, PA	39.95234	-75.16379	41.40897
Philadelphia, PA	Wilkes-Barre, PA	39.95234	-75.16379	41.24591
Philadelphia, PA	Johnstown, PA	39.95234	-75.16379	40.32674
Philadelphia, PA	Reading, PA	39.95234	-75.16379	40.33565

end_lat	mode	days	hours	mins	travel~t
-75.49018	1	0	1	9	61.5
-76.72775	3	1	5	0	87.2
-76.30551	3	0	21	13	63.3
-76.88441	1	0	1	56	107
-79.99589	1	0	5	15	305
-80.08506	3	5	0	0	362
-75.66241	3	1	16	0	119
-75.88131	3	1	14	0	112
-78.92197	1	0	4	13	239
-75.92687	1	0	1	10	57.6

When the travel mode between Philadelphia, PA, and York, PA, is switched from automobile to walking, the travel time increases from one hour and fifty-six minutes to one day and five hours, while the distance decreases from 102 miles to 87.2 miles. The difference in the distance is due to the fact that when travel mode changes, the travel route might also change. When the mode of travel between Philadelphia, PA, and Scranton, PA, is changed from automobile, the travel time increases from two hours and thirteen minutes to one day and sixteen hours, while the distance decreases from 125 miles to 119 miles.

`traveltime` faces the same queries limit from Google Maps as with the `geocode` command. Although it is not clear what the exact limit is, we do not recommend attempting to use the `traveltime` command with more than 10,000 to 15,000 latitude and longitude pairs at any one time.

4 Conclusions

Geographic information is often important in economic, epidemiological, and sociological research. The driving distance and time between locations and the geocoded addresses are useful in a wide variety of research. When dealing with a small set of addresses or latitude/longitude coordinate pairs, researchers can use Google Maps to find latitude and longitude or to find driving distance and drive time. Likewise, users with ArcGIS

or similar high-end mapping software can get the information for larger numbers of observations. The `geocode` and `traveltime` commands allow users to geocode and estimate travel time and travel distance for datasets within Stata by querying Google Maps. This approach provides users with the convenience of a high-end software package like ArcGIS and the free services of Google Maps.

5 Acknowledgments

We thank Graeme Blair for suggestions and editing, and Econsult Corporation for time and resources to work on this project. We would also like to acknowledge Franz Buscha and Lionel Page, for providing us with their `gmap` command, and Mai Nguyen, Shane Trahan, Patricia Nguyen, and Wafa Handley, whose work integrating Google Maps and SAS was helpful.

About the authors

Daniel Miles and Adam Ozimek are associates at Econsult Corporation, an economics consulting firm from Philadelphia, PA, that provides economic research in support of litigation, as well as economic consulting services.

The Stata Journal (2011)
11, Number 1, pp. 120–125

eq5d: A command to calculate index values for the EQ-5D quality-of-life instrument

Juan Manuel Ramos-Goñi
Canary Islands Health Care Service
Canary Islands, Spain
jramos@sescs.es

Oliver Rivero-Arias
Health Economics Research Centre
University of Oxford
Oxford, UK
oliver.rivero@dphpc.ox.ac.uk

Abstract. The `eq5d` command computes an index value using the individual mobility, self care, usual activities, pain or discomfort, and anxiety or depression responses from the EuroQol EQ-5D quality-of-life instrument. The command calculates index values using value sets from eight countries: the United Kingdom, the United States, Spain, Germany, the Netherlands, Denmark, Japan, and Zimbabwe.

Keywords: st0220, eq5d, EQ-5D, index value

1 Description

The `eq5d` command computes an index value from individual responses to the EQ-5D quality-of-life instrument. The EQ-5D is a generic quality-of-life survey developed by the EuroQol Group and used widely by health economists and epidemiologists conducting applied work (EuroQol Group 1990). The EQ-5D survey includes five questions or domains covering mobility, self care, usual activities, pain or discomfort, and anxiety or depression. Each domain contains three possible responses indicating "no problem", "some problems", or "extreme problems". Therefore, the EQ-5D yields 243 (or 3^5) possible health states that can be converted into an index value or a health-related quality-of-life score using a validated value set normally estimated using time trade-off methods and regression analysis. Initially only available in the United Kingdom, over the last decade, several country-specific value sets have been estimated and compiled by the EuroQol Group in Szende, Oppe, and Devlin (2007).

The EQ-5D index has an upper bound equal to 1 that indicates full health (indicated by "no problem" in all domains), whereas 0 represents death. Negative values are allowed, and the lower bound varies depending on the country-specific value set used.

`eq5d` provides users and programmers working with EQ-5D data in Stata with an easy implementation of the published country-specific value sets.

2 Syntax

`eq5d` *varname*$_1$ *varname*$_2$ *varname*$_3$ *varname*$_4$ *varname*$_5$ [*if*] [*in*]
[, `country(GB|US|ES|DE|NL|DK|JP|ZW)` `saving(`*newvarname*`)` `by(`*groupvar*`)`]

The variables must be introduced in the same order in which they appear in the EQ-5D questionnaire, for example, "mobility" (`eqmob`) for *varname*$_1$, "self-care" (`eqcare`) for *varname*$_2$, "usual activities" (`equact`) for *varname*$_3$, "pain/discomfort" (`eqpain`) for *varname*$_4$, and "anxiety or depression" (`eqanx`) for *varname*$_5$. In addition, the levels of each EQ-5D variable need to be coded as follows: `1` for "no problem", `2` for "some problem", and `3` for "extreme problems". When missing values are present in any of the domains for a particular individual, the index-value calculation for that individual will also be missing.

3 Options

`country(GB|US|ES|DE|NL|DK|JP|ZW)` specifies the country-specific value set to be used in the estimation of the EQ-5D index values. The country code should be specified in capital letters as follows: the United Kingdom (`GB`), the United States (`US`), Spain (`ES`), Germany (`DE`), the Netherlands (`NL`), Denmark (`DK`), Japan (`JP`), and Zimbabwe (`ZW`). The default is `country(GB)`.

`saving(`*newvarname*`)` specifies the name of the new variable under which the index value will be stored.

`by(`*groupvar*`)` specifies the group variable that contains the groups to be used by `eq5d` when reporting descriptive statistics.

4 Example

To illustrate how `eq5d` works, a hypothetical dataset of 20 individuals with information on the five domains of the EQ-5D, along with gender and age, has been simulated. The data have been stored in `eq5d.dta`.

```
. use eq5d
(Example data for eq5d)
. describe
Contains data from eq5d.dta
  obs:            20                          Example data for eq5d
 vars:             8                          18 Feb 2010 13:59
 size:           300 (99.9% of memory free)
```

variable name	storage type	display format	value label	variable label
id	long	%12.0g		Individual identifier
age	byte	%8.0g		Age
gender	byte	%8.0g	gender	Gender
eqmob	byte	%15.0g	mobility	EQ-5D mobility
eqcare	byte	%13.0g	care	EQ-5D self-care
equact	byte	%13.0g	activity	EQ-5D usual activities
eqpain	byte	%13.0g	pain	EQ-5D pain
eqanx	byte	%18.0g	anxiety	EQ-5D anxiety

```
Sorted by:  gender
. list, nolabel
```

	id	age	gender	eqmob	eqcare	equact	eqpain	eqanx
1.	1	49	1	1	1	1	2	1
2.	2	68	1	1	2	1	2	1
3.	3	75	1	2	1	3	2	3
4.	4	66	1	1	1	1	2	1
5.	5	66	1	1	2	2	2	2
6.	6	29	1	1	1	1	1	1
7.	7	35	1	1	1	1	2	1
8.	8	40	1	1	1	1	1	1
9.	9	30	1	1	1	1	1	1
10.	10	49	1	2	1	2	1	3
11.	11	23	2	1	1	1	1	1
12.	12	44	2	2	1	1	2	2
13.	13	85	2	2	2	2	2	2
14.	14	30	2	1	3	1	1	1
15.	15	20	2	1	1	1	1	1
16.	16	46	2	1	1	1	2	1
17.	17	50	2	1	1	1	1	1
18.	18	82	2	2	2	2	2	1
19.	19	49	2	1	1	1	1	1
20.	20	21	2	1	1	1	1	1

The sample data have been sorted by `gender`, where `1` indicates male and `2` indicates female, with women on average enjoying a better quality of life compared with men. The lower quality of life of male individuals is driven by observations 3 and 10, which both feature a level-3 (extreme problems) response on at least one domain. The EQ-5D index value for the whole group using the United States value set is calculated and reported as follows:

```
. eq5d eqmob eqcare equact eqpain eqanx, country(US)
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      _index |         20      .81365    .1910834   .4029999          1
```

`eq5d` displays summary statistics for a group variable with the `by()` option. In the current dataset, for example, we can display summary statistics for the EQ-5D index for the `gender` variable as follows:

```
. eq5d eqmob eqcare equact eqpain eqanx, country(US) by(gender)
-------------------------------------------------------------------------------
-> gender = Male
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      _index |         10        .7875    .2013876   .4029999          1
-------------------------------------------------------------------------------
-> gender = Female
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      _index |         10        .8398    .1870994        .529          1
```

`eq5d` also displays summary statistics for a specific group of observations determined by the `if` and `in` conditions. For example, for a group of patients within a particular age interval, we could explore the summary statistics for the index values as follows:

```
. eq5d eqmob-eqanx if age>32 & age<70, country(US)
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      _index |         11    .8236364    .1432028        .533          1
```

5 Saved results

`eq5d` saves the following in `r()`:

Scalars

`r(Nincluded)`	number of included observations
`r(Ntotal)`	number of total observations
`r(Nvalid)`	number of valid observations
`r(mean)`	mean
`r(Var)`	variance
`r(sd)`	standard deviation
`r(min)`	minimum
`r(max)`	maximum

6 Methods and formulas

`eq5d` applies the additive linear equation $y = \boldsymbol{\beta}\mathbf{X}$ to estimate index values, where $\boldsymbol{\beta}$ is a vector of coefficients representing decrements from full health of the index value and $\mathbf{X}$ is a matrix indicating a set of covariates. The algorithm starts with all individuals in full health (that is, the index value equals 1). Depending on the country-specific

value set selected, the number of items in $\boldsymbol{\beta}$ and $\mathbf{X}$ varies, reflecting the type of model selected to fit the value sets in each particular country. A brief description of the items included in $\boldsymbol{\beta}$ and $\mathbf{X}$ in each country is given as follows:

Denmark, Japan, and Zimbabwe
: $\boldsymbol{\beta}$ represents decrements of the index value associated with the items in the $\mathbf{X}$ matrix. $\mathbf{X}$ is a matrix with the dummy variables for "some problems" and "extreme problems" in each domain of the EQ-5D. $\mathbf{X}$ also has a dummy variable indicating whether the individual is not in full health.

The United Kingdom, Spain, Germany, and the Netherlands
: $\boldsymbol{\beta}$ represents decrements of the index value associated with the items in the $\mathbf{X}$ matrix. $\mathbf{X}$ is a matrix with the dummy variables for "some problems" and "extreme problems" in each domain of the EQ-5D. $\mathbf{X}$ also has a dummy variable indicating whether the individual is not in full health and an additional dummy variable indicating whether "extreme problems" were reported in any of the domains.

The United States
: $\boldsymbol{\beta}$ represents decrements of the index value associated with the items in the $\mathbf{X}$ matrix. $\mathbf{X}$ is a matrix with the following: dummy variables for "some problems" and "extreme problems" in each domain of the EQ-5D, an ordinal variable that represents the number of deviations from full health beyond the first movement away, an ordinal variable that represents the number of domains with "extreme problems" beyond the first movement and its square, and the square of an ordinal variable that represents the number of domains with "some problems" beyond the first movement away.

For a full description of the models fit in each country, the reader is referred to the original research publications. References can be found in the monograph by the EuroQol Group (Szende, Oppe, and Devlin 2007).

Note: Death in the EQ-5D value sets is coded `0`, but `eq5d` will report missing values for deceased patients because no EQ-5D responses are available. Hence, the user needs to recode these values manually if mortality is present in the dataset after implementing `eq5d`.

7 Acknowledgments

We thank researchers and funding bodies who have conducted high-quality studies to estimate country-specific EQ-5D value sets and therefore have made this Stata command possible. We are grateful to Helen Campbell at University of Oxford and to an anonymous reviewer for comments and suggestions on earlier drafts of this manuscript.

8 References

EuroQol Group. 1990. EuroQol—a new facility for the measurement of health-related quality of life. *Health Policy* 16: 199–208.

Szende, A., M. Oppe, and N. Devlin, ed. 2007. *EQ-5D Value Sets: Inventory, Comparative Review and User Guide*. Dordrecht: Springer.

About the authors

Juan Manuel Ramos-Goñi is a biostatistician at the Canary Islands Health Care Service, Spain.

Oliver Rivero-Arias is a senior researcher at the Health Economics Research Centre at the University of Oxford, United Kingdom.

The Stata Journal (2011)
11, Number 1, pp. 126–142

Speaking Stata: MMXI and all that: Handling Roman numerals within Stata

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
n.j.cox@durham.ac.uk

Abstract. The problem of handling Roman numerals in Stata is used to illustrate issues arising in the handling of classification codes in character string form and their numeric equivalents. The solutions include Stata programs and Mata functions for conversion from numeric to string and from string to numeric. Defining acceptable input and trapping and flagging incorrect or unmanageable inputs are key concerns in good practice. Regular expressions are especially valuable for this problem.

Keywords: dm0054, fromroman, toroman, Roman numerals, strings, regular expressions, Mata, data management

1 Introduction

"MMXI" within the title, as you will recognize, is a Roman numeral representing 2011. Although the decline and fall of the Roman Empire is a matter of ancient history, Roman numerals are far from obsolete, even in contemporary science and technology. Recently on Statalist, the distinguished statistician Tony Lachenbruch asked about handling Roman numerals in Stata. Answering his question has proved to be entertaining and enlightening, so here I share my experiences.

The interest of this problem does not depend on how commonly it arises in Stata practice. Rather, how well can Stata meet such a challenge? One hallmark of a statistical environment such as Stata is its extensibility, its scope for adding new functionality that supports new needs. The Roman numeral problem raises several issues typical of conversions between string codes and numeric equivalents. Depending partly on your background, you are likely to be familiar with formal codes for book classification, medical conditions, sectors of economic activity, and so forth. The Roman numeral example is not so trivial that it can be discussed and dismissed in a paragraph, but not so tricky nor so large as to defy careful and complete examination in a column.

As of Stata 11, official Stata provides no specific support for handling Roman numerals, for converting them to Hindu–Arabic decimal numbers, or for representing decimal numbers as Roman numerals using a dedicated display format. Users do not have scope for creating new Stata functions or display formats, so the problem in practice pivots on being able to move back and forth between string representations such as `"MMXI"` and numeric representations such as 2011.

A key principle with any kind of conversion is that conversions either way should be supported if they both make sense. Thus in Stata, string-to-numeric and numeric-to-string functions and commands occur in pairs: `real()` and `string()`, `encode` and `decode`, and `destring` and `tostring`. See an earlier column (Cox 2002) for more on this theme, while noting that `tostring` has become an official command since that column was written. A further motive for implementing both conversions is to provide some consistency checking. Broadly, conversion one way followed by conversion the other way should yield the original. (We will touch later on the possibility of different string encodings of the same numbers.)

With this column are published two new programs, `fromroman` and `toroman`, and two stand-alone Mata functions. We need to discuss not only how to solve the problem but also why it is done that way.

2 Roman numerals

Let us first spell out the rules, or at least one common version of the rules, for Roman numerals and their conversion to decimal numbers.

Character set. The atoms (our term) M, D, C, L, X, V, and I stand for 1000, 500, 100, 50, 10, 5, and 1.

Subtraction rule. The composites (also our term) CM, CD, XC, XL, IX, and IV are used to stand for 900, 400, 90, 40, 9, and 4. Whenever any of these composites occurs, this rule trumps the previous rule.

Order rule. Two or more atoms or composites appearing within a numeral appear in the order implied by their numerical equivalent.

Parsimony rule. The smallest number of elements possible is used to encode any number.

Addition rule. Following conversions under the first two rules, sum the results.

Thus a person knowing these rules would know to parse MMXI as M + M + X + I $= 1000 + 1000 + 10 + 1 = 2011$ and would also parse MCMLII as M + CM + L + I + I $= 1000 + 900 + 50 + 1 + 1 = 1952$.

The subtraction rule is the twist that gives this problem its particular spin. Without it, we would just need to count the occurrences of the seven possible atoms, multiply, and then sum—a much simpler problem.

For those seeking further information on Roman numerals, the popular accounts of Gullberg (1997) and Ifrah (1998) provide much interesting and useful detail. Although these works have some scholarly limitations (Allen 1999; Zebrowski 2001; and Dauben 2002), they remain interesting and useful at the level we need here. The classic monographs of Cajori (1928) and Menninger (1969) remain useful surveys. Such sources underline that what we now know as Roman numerals are in fact a limited and late version. Roman symbols and conventions for numbers greater than 1000, for fractions, and for various multiplications have evidently not survived into widespread current use,

so no more will be said about them. Conversely, the subtraction principle, whereby (for example) CM is interpreted as C subtracted from M, only became popular in late medieval times.

History aside, it is often said that Roman numerals are not especially attractive mathematically. Addition and subtraction with Roman numerals are awkward, and multiplication and division rather more so, although the difficulties can be exaggerated. The subtractive notation that increases the awkwardness is of late popularity, and the abacus was often used for calculations, anyway.

However, the rules do seem a little arbitrary. Clearly, there would be no practical point to allowing, say, DM, LC, or VX, which would just be longer ways of writing D, L, or V. But what is the objection to, say, IM, VM, or XM? Indeed, historical examples of subtractive composites other than those specified above can be found. However, the point is not to question the rules but to state what they are usually reported to be. Either way, Roman numerals are best thought of as presentation numerics rather than computational numerics.

3 Converting variables containing Roman numerals

3.1 Principles

Let us imagine a string variable with values that are Roman numerals such as `"I"`, `"II"`, `"III"`, `"IV"`, `"V"`, and so forth. The initial problem is to convert to decimal numbers such as 1 to 5. If only a few small numbers were so represented, it might be feasible to define value labels that could then be used with the `encode` command (see [D] **encode**), but this approach is not practical otherwise, at least without a tool created for the purpose. More subtly, a value-label approach would fail if there were different possibilities for representation, as if `IIII` were also allowed as an alternative to `IV`, or `XXXX` as an alternative to `XL`, or `CCCC` as an alternative to `CD`. A curiosity is that `IIII` is often shown on clockfaces and watch faces using Roman numerals.

Hence, we will need to set up our own conversion code. The idea that the data come as variables will for the while lie behind discussion. In a later section, we will focus on a Mata-based approach that extends the scope.

One possible starting point is to think through how somebody conversant with the rules would convert a Roman numeral by eye. However, recipes suitable for people are not always those most suitable for programs, as every programmer learns. I have not tried to write Stata code to parse Roman numerals from left to right, as many of us were taught to do in our early education. The approach I have tried looks for subtractive composites first, given that their occurrence trumps atom-by-atom interpretation.

Given a string Roman numeral to be converted to a decimal number, here is the core of the algorithm:

1. Initialize the number to 0.

2. Find any occurrences of CM, CD, XC, XL, IX, and IV. Increment the number in each case by 900, 400, 90, 40, 9, or 4, as appropriate. Blank out those occurrences.

3. Find any occurrences of M, D, C, L, X, V, and I. Increment the number in each case by 1000, 500, 100, 50, 10, 5, or 1, as appropriate. Blank out those occurrences.

4. Whatever remains is regarded as problematic input and is flagged to the user.

   You can easily mimic this algorithm with examples such as `"MMXI"` and `"MCMLII"` by using pencil and paper and crossing out rather than blanking out. In Stata, the function for blanking out is `subinstr()`, which is used in this case to replace substrings by blanks or empty strings (`""`).

   As yet, this algorithm includes no checking for malformed numerals. `"IM"` would be converted to 1001 rather than be rejected as malformed. This problem will be addressed shortly.

   More positively, there are three easy extensions to handle other possibilities.

5. Strings often contain spaces, which should usually just be ignored. In particular, any leading and trailing spaces might just be side effects of data entry. However, if a user had composite strings such as `"II III"`, meaning 2 and 3 as two separate occurrences, then applying the `split` command (see [D] **split**) beforehand would be recommended.

6. The possibility of lowercase numerals (m, d, c, l, x, v, and i) would be easy to handle, because the function `upper()` can be used to convert to uppercase beforehand.

7. Any occurrences of j for i or of u for v—seemingly rare now but mentioned in the literature as historic variants—could be accommodated with other applications of `subinstr()`.

3.2 Code

The Stata code published with this column includes a Stata program, `fromroman`, and a Mata function with the same name for converting Roman numerals. It would be possible to write a program entirely in Stata for this problem, but a Stata program that calls a Mata function is a more attractive solution. Greater speed is sometimes a motive for using Mata, but not for this problem because the computations are trivial in machine time. Convenience of the programmer is a greater motive for using Mata because Mata provides much of the elegance and generality of modern programming languages with the hooks needed to interface with Stata.

The remainder of this subsection presupposes some acquaintance with Mata for a full understanding, but even if Mata is new to you, you might want to skim along. The logic of the problem is not difficult, and the Mata details should make some sense, even if you could not have written them down yourself. Much of the code is mundane, but some sections are more distinctive and worth some comment.

At the heart of `fromroman` is Mata code that converts a string column vector `sin` of Roman numerals to a numeric column vector of decimal numbers `nout`. These column vectors are in turn input from a Stata string variable and intended for output to a Stata numeric variable. `nout` is initialized as all zeros.

```
sin = st_sdata(., varname, usename)
nout = J(rows(sin), 1, 0)
```

Two vectors of string and numeric constants are set up to define the conversion mapping. Anyone wanting to use different conversion rules would need to modify these vectors. It is arbitrary that they are set up as column vectors, because they are not aligned with the other column vectors. Row vectors would also work fine.

```
rom = ("CM", "CD", "XC", "XL", "IX", "IV", "M", "D", "C", "L", "X", "V", "I")´
num = (900, 400, 90, 40, 9, 4, 1000, 500, 100, 50, 10, 5, 1)´
```

There is then a loop over both of these vectors. We must look for any of the composites (CM, CD, XC, XL, IX, and IV) before we look for any of the atoms (M, D, C, L, X, V, and I), so to that extent, the order is important.

In an early version of the program, I wrote the segment below. The code can be much improved, as I will explain.

```
for (i = 1; i <= rows(rom); i++) {
        while (sum(strpos(sin, rom[i]))) {
                nout = nout + num[i] * (strpos(sin, rom[i]) :> 0)
                sin = subinstr(sin, rom[i], "", 1)
        }
}
```

A small point of style here is that the loop is written as over the elements of the vector as from 1 to `rows(rom)`. We could wire in 13 (the number of elements in both vectors) and save ourselves a tiny amount of computation, but the downside is to require anyone trying to understand the code to puzzle out what the constant 13 is. This choice is reinforced by the idea that someone wanting different rules would need to change the vectors. The opposite decision might be made if the problem were, say, looping over the decimal digits 0 to 9, when it should be obvious why 10 is the number of elements.

That said, let us focus on the conversion itself. There is a test of whether each element of `rom[i]` is contained within `sin`, which is done within the line

```
nout = nout + num[i] * (strpos(sin, rom[i]) :> 0)
```

`strpos()` returns the position of that element. To see how that works, follow through as we start with `rom[1]`, which is `"CM"`.

If we were processing `"MCMLII"`, then `strpos("MCMLII", "CM")` would be returned as 2 because the string `"CM"` does occur within `"MCMLII"`, and it starts at the second character of `"MCMLII"`. In contrast, `strpos("MMXI", "CM")` is returned as 0 because the string `"CM"` does not occur within `"MMXI"`. More generally, `strpos(`*str1*`,` *str2*`)` returns a positive integer for the first position of *str2* within *str1*, or it returns 0 if there is no such position. (Longtime users of Stata may have met `strpos()` in Stata under the earlier name `index()`.)

Thus the comparison `strpos(sin, rom[i]) :> 0` yields 1 whenever the element occurs and 0 whenever it does not occur because those cases yield positive and 0 results from `strpos()`, and the latter never returns a negative result. Multiplying 1 or 0 by `num[i]` adds `num[i]` or 0, as appropriate. Thus with `CM`, 900 would be added to 0 (the initial value) for the input `"MCMLII"`, but 0 would be added for the input `"MMXI"`.

Once we have taken numeric account of the first occurrence of each pertinent element of `rom[]`, we can blank it out within `sin`:

```
sin = subinstr(sin, rom[i], "", 1)
```

Within Mata, as typically within Stata, this calculation is vectorized so that Mata deals with all the elements of `sin`, which correspond to the values held within various observations for a string variable. That is why the inequality comparison is elementwise, as shown by the colon prefix in `:>`.

However, as already mentioned, `strpos()` identifies at most the first occurrence of one string within another. That suffices when dealing with the composites such as `"CM"`, which we expect to occur at most once within any numeral. However, we need to be able to handle multiple occurrences of `"M"`, `"C"`, `"X"`, and `"I"`, which are predictable possibilities. These are handled by continuing to process such elements as long as they are found. The two statements we have looked at in detail are within a `while` loop:

```
while (sum(strpos(sin, rom[i]))) {
        nout = nout + num[i] * (strpos(sin, rom[i]) :> 0)
        sin = subinstr(sin, rom[i], "", 1)
}
```

Recall that we blank out each occurrence of the elements of `rom[]` within `sin[]` as we process it. We can monitor whether there is anything left to process by summing `strpos(sin, rom[i])`. The sum, like the individual results from `strpos()`, is positive given any occurrence and 0 otherwise, but this time the check is for occurrences within the entire vector. A `while` loop continues so long as the argument of `while()` is positive. Some might prefer to spell out the logic as

```
while (sum(strpos(sin, rom[i])) > 0)
```

The choice is one of style. The result is the same.

Let us back up and consider another way, which is that now used in `fromroman`. Instead of blanking out occurrences of each element of `rom[i]` one by one, we can blank them all out at once. We can track how many occurrences there were from a comparison of string lengths. `strlen()` returns the length of a string. (Longtime users of Stata may have met `strlen()` in Stata under the earlier name `length()`.) Here is the new code:

```
for (i = 1; i <= rows(rom); i++) {
        sin2 = subinstr(sin, rom[i], "", .)
        nout = nout + num[i] * (strlen(sin) - strlen(sin2)) / strlen(rom[i])
        sin = sin2
}
```

So the number of elements blanked out is calculated from the difference between string lengths before and after. Division by `strlen(rom[i])` is needed because composite elements are two characters long, and atoms are just one character long.

This way, the repetition encoded in `while()` loops becomes quite unnecessary. The more ambitious code is in fact simpler, which is reminiscent of what Pólya (1957, 121) codified as the inventor's paradox: the more ambitious plan may have more chances of success.

3.3 Checking

We have postponed discussion of the need to check that input does actually obey the rules we are using. If it does not, further questions arise for the user: Is the supposedly bad input some kind of data error? Is a variation on the rules needed? The latter would arise if (for example) CCCC, XXXX, or IIII were regarded as acceptable.

A good way to check input is to define acceptable input using a so-called regular expression. Regular expressions are a little language for defining the patterns that strings may take. In practice, they are a little language with many different dialects, implemented in a variety of software. Stata's own regular expression implementation is most fully documented by Turner (2005). Much fuller discussions of regular expressions in general are available: the excellent account by Friedl (2006) is accessible and useful to learners. All that is needed to solve the problem here is contained within its first chapter. Brief introductions to be found in many books (for example, Kernighan and Pike [1984]; Aho, Kernighan, and Weinberger [1988]; Abrahams and Larson [1997]; and Raymond [2004]) may also be useful.

A regular expression for Roman numerals is

```
^M*(C|CC|CCC|CD|D|DC|DCC|DCCC|CM)?(X|XX|XXX|XL|L|LX|LXX|LXXX|XC)?
  (I|II|III|IV|V|VI|VII|VIII|IX)?$
```

This can be parsed as follows:

1. `^` marks the start of the numeral.
2. `M*` means that the character M may occur zero or more times.

3. `(C|CC|CCC|CD|D|DC|DCC|DCCC|CM)?` means that one of C, CC, CCC, CD, D, DC, DCC, DCCC, or CM may occur (meaning precisely, may occur zero or one time).

4. Similarly, `(X|XX|XXX|XL|L|LX|LXX|LXXX|XC)?` and `(I|II|III|IV|V|VI|VII|VIII|IX)?` mean that just one of the items in each parenthesized list may occur.

5. `$` marks the end of the numeral.

Thus the idea here is that `^`, `*`, `?`, `|`, `()`, and `$`—so-called meta-characters—have special syntactic meaning, while the other characters are all to be taken literally.

Much more could be said about regular expressions, which are an important area in their own right, but a few comments will have to do. You should not imagine that the use of meta-characters (which include others beyond those used here) rules out defining regular expressions in which the same characters occur literally. There are simple tricks to meet such needs.

The regular expression here for Roman numerals makes no particular element compulsory. A side effect is that it is satisfied by empty strings, which break none of the rules. This is not a problem so long as we remember to check that a string is not empty or we ensure that no empty strings are fed to the regular expression.

In writing down this regular expression, I plumped for clarity rather than brevity. Following a personal communication from Kerry Kammire,

```
^M*(CM|CD|D?C?C?C?)(XC|XL|L?X?X?X?)(IX|IV|V?I?I?I?)$
```

can be offered as another way to write it down.

Once you have a regular expression, the Stata function `regexm()` yields 1 if it is satisfied and 0 otherwise. Thus we can reject input, or more helpfully, flag it if it fails such a test.

I offer a further aside: In an earlier version of `fromroman`, I went through the possible elements one by one using code based on `if` to test for elements that might occur once and code based on `while` to test for elements that might occur more than once. What lay behind this design was twofold. For a small problem, I like to get a rough program working as quickly as possible. Once code is in front of me, it is often much easier to see how to improve it. More specifically, the underlying idea was to include some degree of checking against malformed numerals. Once code had been written to check input all at once using a regular expression, it became obvious that the repetitive mix of `if` and `while` code could be replaced by a single `for` loop. Once again, the more ambitious code is in fact simpler.

While in a vein of dispensing homespun philosophy, it need only be added that the art of solving large problems is often to reduce them to a series of small problems.

4 Converting variables to Roman numerals

4.1 Principles

Let us now imagine the reverse or inverse problem. We have a variable with decimal numeric values and a need to convert to a string variable containing Roman numerals. In practice, we are concerned with positive integers such as 42; 1952; or 2011. Here is one algorithm for producing Roman numerals:

1. Initialize the numeral to `""`.
2. For each of the elements 1000, 900, 500, 400, 100, 90, 50, 40, 10, 5, 4, and 1, follow these steps:
    a. Try to subtract that element from the number.
    b. If the result is positive or 0, add (concatenate to the right) the corresponding numeral atom or composite, and subtract the element, replacing the number with a new one.
    c. If the result is positive, repeat 2.a and 2.b with the same element and the new number.
    d. If the result is 0, stop.
    e. If the result is negative, proceed to the next element.

Some constraints on the process are satisfied automatically with this procedure. For example, the fact that `"CM"` can occur at most once is satisfied because if any number is less than 1000, then 900 can be subtracted from it at most once. In the same way, other key facts are all consequences of the algorithm. That is, the other subtractive composites, and also `"D"`, `"L"`, and `"V"`, can each occur at most once; and `"C"`, `"X"`, and `"I"` can each occur at most three times.

4.2 Code

The Stata code published with this column includes a Stata program, `toroman` (and a Mata function with the same name), for converting decimal numbers to Roman. As before, comment is restricted to the most distinctive part of the code. If you skipped or skimmed earlier material on Mata, you might want to repeat that now.

In the Mata function at the heart of `fromroman`, a numeric column vector `nin` is mapped to a string column vector `sout`, which is initialized to empty (`""`) in each element. The conversion is defined as before by two aligned vectors. This time, the vectors are ordered largest number first.

This code is an implementation of the algorithm just given:

```
nin = st_data(., varname, usename)
sout = J(rows(nin), 1, "")
rom = ("M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I")'
num = (1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1)'
for (i = 1; i <= rows(rom); i++) {
        toadd = nin :- num[i] :>= 0
        while (sum(toadd)) {
                sout = sout :+ toadd :* rom[i]
                nin = toadd :* (nin :- num[i]) + (!toadd) :* nin
                toadd = nin :- num[i] :>= 0
        }
}
```

The main complication to be tackled is converting a vector all at once. As of Stata 11, Mata lacks an elementwise version of the conditional operator (that used in the example `a > b ? a : b`), so we mimic that for ourselves using a vector `toadd` containing 1s and 0s and its negation, `!toadd`. As the name suggests, `toadd` has values of 1 whenever we can add an element of `rom[]` and 0 otherwise. (Ben Jann's `moremata` package, downloadable from the Statistical Software Components archive, does include a conditional function that works elementwise. Typing `findit moremata` in Stata would point you to more information.)

Once we have worked out that conditional calculation, we can add the Roman numeral. In the line

```
sout = sout :+ toadd :* rom[i]
```

we are multiplying strings as well as adding them. In Mata, adding strings is just concatenation (joining lengthwise), and multiplying strings is just repeated concatenation so that `2 * "Stata"` yields `"StataStata"`. Here both the addition and the multiplication are elementwise, as shown again by the colon prefixes. Thus whenever `toadd` is 1, we add the corresponding element of `1 :* rom[i]`, which is just `rom[i]`; and whenever `toadd` is 0, we add `0 :* rom[i]`, which is always empty.

This is not exactly the code used in `toroman`. It is just a naïve version encoding one subtraction at each step, but there is no reason to limit ourselves to that. We can rewrite the loop to get the number of subtractions needed directly:

```
for (i = 1; i <= rows(rom); i++) {
        sout = sout :+ floor(nin/num[i]) :* rom[i]
        nin = nin :- num[i] :* floor(nin/num[i])
}
```

The more ambitious code is, yet again, much simpler. Let us see how that works with a simple numerical example. The vector given by

```
: floor((2011, 1952, 42)'/1000)
       1
    +---+
  1 |  2|
  2 |  1|
  3 |  0|
    +---+
```

is the number of times we can write `"M"` at the start of the corresponding Roman numerals and the number of times to subtract 1000 for the next step of the loop. I like to use `floor()` as the function here because its name so clearly evokes rounding down (Cox 2003). The Mata function `trunc()`, the equivalent of Stata's `int()` function, would work, too.

The moral is simple but crucial. We can easily underestimate the scope of languages like Stata and Mata to do several things at once if we translate too closely from recipes using one little step at a time.

However, we need to watch out. Suppose that somehow −1 was fed to the code segment above. The Roman numeral that would emerge would be `"CMXCIX"`, which looks crazy but can be explained. It emerges because `floor(-1/1000)` is −1. Subtracting −1000, so adding 1000, yields 999, which is then correctly encoded. There are various ways around this. One is to trap negative values beforehand. We will look at the question of checking shortly. Another is to work with the larger of 0 and `floor(num / rom[i])`, which ensures that negative values are mapped to empty strings and so ignored.

A final thought is this: If we were doing this by hand, then naturally we would stop whenever the job was done. Given 2000, `"MM"` is clearly the solution because repeated subtraction has yielded 0, and we would know not to try anything else. Is it worth building in a check that we are done?

Consider the statistics. Of possible Roman numerals, the frequencies per 1000 finished with each possible element are `"M"`, 1; `"CM"`, 1; `"D"`, 1; `"CD"`, 1; `"C"`, 6; `"XC"`, 10; `"L"`, 10; `"XL"`, 10; `"X"`, 60; `"IX"`, 100; `"V"`, 100; `"IV"`, 100; and `"I"`, 600. Hence, testing to allow leaving a loop early will not in practice help us appreciably. It might well slow us down!

4.3 Checking

As mentioned in the previous subsection, numeric values for this problem should in practice mean positive integers only. A careful approach requires trapping any fractional, zero, and negative numbers given as input. This is simple in Stata using an `if` condition applied before calling the Mata function doing the encoding.

A more subtle issue is the upper limit for this calculation. Possible candidates for conversion include year dates and page numbers, which are both most unlikely to exceed a few thousand and so will be encoded by at most a few characters in a string. Nevertheless, it is worth thinking carefully about the limits to any conversion program.

As of Stata 11, the largest (meaning widest) type of string variable allowed in Stata is `str244`. Thus 245,000, which would convert to a numeral containing 245 Ms, could not be held as a Roman numeral within a Stata variable. In fact, not all smaller numbers can be so held. We can work out the precise limit as follows.

Inspection makes clear that numbers with digits of 8 produce the longest Roman numerals. VIII is the longest for a one-digit decimal number, LXXXVIII and DCCCLXXXVIII are the longest for two- and three-digit numbers, and so on. The smallest problematic number will be the smallest number that needs 245 characters, which is 233,888, represented by 233 Ms followed by the 12 characters DCCCLXXXVIII.

As mentioned in an earlier section, other and much better ways to hold numbers that are several thousand or more characters as Roman numerals were used in history. The point is rather to find the upper limit within Stata to the procedure using the rules in section 2. It seems most unlikely that an upper limit of 233,887 would ever bite in practice, which is good news.

5 Syntax for fromroman and toroman

5.1 fromroman

`fromroman` *romanvar* [*if*] [*in*] , `generate(`*numvar*`)` [`re(`*regex*`)`]

Description

`fromroman` creates a numeric variable *numvar* from a string variable *romanvar* following these rules:

1. Any spaces are ignored.
2. Lowercase letters are treated as if uppercase.
3. Numerals must match the Stata regular expression `^M*(CM|DCCC|DCC|DC|D|CD|CCC|CC|C)?(XC|LXXX|LXX|LX|L|XL|XXX|XX|X)?(IX|VIII|VII|VI|V|IV|III|II|I)?$`. This forbids, for example, CCCC, XXXX, or IIII, but see documentation of the `re()` option below.
4. Single occurrences of CM, CD, XC, XL, IX, and IV are treated as 900, 400, 90, 40, 9, and 4, respectively.
5. M, D, C, L, X, V, and I are treated as 1000, 500, 100, 50, 10, 5, and 1, respectively, as many times as they occur.
6. The results of 4 and 5 are added.

7. Input of any other expression or characters is trapped as an error and results in missing. Examples would be minus signs and decimal points.

There is no explicit upper limit for the integer values created. In practice, the limit is implied by the limits on string variables so that, using these rules, any numbers greater than 244,000 (and some numbers less than that) could not be stored as Roman numerals in a Stata string variable. (The smallest problematic number is 233,888, which would convert to a Roman numeral consisting of 233 Ms followed by DCCCLXXXVIII—that is, a numeral 245 characters long.) See [D] **data types**.

Options

`generate(`*numvar*`)` specifies the name of the new numeric variable to be created. `generate()` is required.

`re(`*regex*`)` specifies a regular expression other than the default for checking input.

5.2 toroman

`toroman` *numvar* [*if*] [*in*], `generate(`*romanvar*`)` [`lower`]

Description

`toroman` creates a string variable *romanvar* containing Roman numerals from a numeric variable *numvar* following these rules:

1. Negative, zero, and fractional numbers are ignored.
2. The conversion uses M, D, C, L, X, V, and I to represent 1000, 500, 100, 50, 10, 5, and 1 as many times as they occur, except that CM, CD, XC, XL, IX, and IV are used to represent 900, 400, 90, 40, 9, 4.
3. No number that is 233,888 or greater is converted. This limit is implied by the limits on string variables so that, using these rules, any number greater than 244,000 (and some numbers less than that) could not be stored as Roman numerals in a Stata string variable. (The smallest problematic number is 233,888, which would convert to a Roman numeral consisting of 233 Ms followed by DCCCLXXXVIII—that is, a numeral 245 characters long.) See [D] **data types**.

Options

`generate(`*romanvar*`)` specifies the name of the new string variable to be created. `generate()` is required.

`lower` specifies that numerals are to be produced as lowercase letters, such as `"mmxi"` rather than `"MMXI"`.

6 Mata functions

Two Mata functions are also included with the media for this column in the file `roman.mata`. These functions could be used either within Mata or within Stata.

Before we look at how to use those, first let us set aside the question of checking using regular expressions in Mata. Mata has a function called `regexm()`. It was not documented in Stata 9 or 10 and is undocumented in Stata 11, meaning that it is documented in a help file but not in a corresponding manual entry. However, `regexm()` in Mata acts exactly as you would expect given knowledge of the function with the same name in Stata.

Given a string input, define first a suitable regular expression

```
: regex =
> "^M*(CM|DCCC|DCC|DC|D|CD|CCC|CC|C)?(XC|LXXX|LXX|LX|L|XL|XXX|XX|X)?
> (IX|VIII|VII|VI|V|IV|III|II|I)?$"
```

and then input that does not pass muster can be shown by something like

```
: test = ("MMXI", "MCMLII", "XLII", "MIX", "foo")´
: select(test, !regexm(test, regex))
  foo
```

Let us see how the Mata functions operate. In both cases, there is mapping from matrix to matrix. So row vector input will work fine as a special case. `toroman()` ignores zeros and negative numbers, as well as any numeric missings, and returns empty strings in such cases. Fractional numbers are not ignored: given x, `toroman()` works with the floor $\lfloor x \rfloor$.

```
: ntest = (2011, 1952, 42, 0, -1, .)
: toroman(ntest)
           1        2        3        4        5        6
  1 |    MMXI   MCMLII     XLII                            |
```

`fromroman()` gives positive integers and flags problematic (nonempty) input.

```
: fromroman(toroman(ntest))
         1      2      3      4      5      6
  1   2011   1952     42      .      .      .

: stest = ("MMXI", "MCMLII", "XLII", "", "foo")
: fromroman(stest)
  Problematic input:
  foo
         1      2      3      4      5
  1   2011   1952     42      .      .
```

These Mata functions could be called from within a Stata session. That way, they would make up for the lack of any Stata functions in this territory. (We have just discussed two new Stata commands, a different matter.) Applications include conversion of Stata local or global macros or scalars to Roman numeral form. Suppose, for example, that you want to work with value labels for Roman numerals and that you know that only small numbers (say, 100 or smaller) will be used. You can define the value labels within a loop like this:

```
forval i = 1/100 {
        mata : st_local("label", toroman(`i´))
        label def roman `i´ "`label´", modify
}
```

That way, the tedious and error-prone business of defining several value labels one by one can be avoided. You could go on to assign such labels to a numeric variable or to use them with the `encode` command (see [D] **encode**).

7 Conclusions

By tradition, teaching discrete probability starts with problems in tossing coins and throwing dice. Outside of sport and gambling, few people care much about such processes, but they are easy to envisage and work well as vehicles for deeper ideas. In the same way, Roman numerals are not themselves of much note in Stata use, but the issues that arise in handling them are of more consequence.

Various simple Stata morals are illustrated by this problem of conversion between special string codes and numeric equivalents.

Write conversion code for both ways. If someone wants one way now, someone will want the other way sooner or later. Writing both is less than twice the work.

Define and check for acceptable input. Regular expressions are one very useful way of doing this. Flag unacceptable input. Consider the limits on conversions, even if they are unlikely to cause a problem in practice.

Combine Mata and Stata. Stata–Mata solutions give you the best of both worlds. Mata functions can be useful outside Mata.

Know your functions. Functions like `subinstr()`, `strpos()`, `strlen()`, and `floor()` give you the low-level manipulations you need.

Even rough solutions can often be improved. Often you have to write down code and mull it over before better code will occur to you.

More ambitious code can often be simpler. Handling special cases individually is sometimes necessary, but it often is a signal that a more general structure is needed.

8 Acknowledgments

Peter A. "Tony" Lachenbruch suggested the problem of handling Roman numerals on Statalist. This column is dedicated to Tony on his retirement as a small token of recognition of his many services to the statistical (and Stata) community.

Sergiy Radyakin's comments on Statalist provoked more error checking. Kerry Kammire suggested an alternative regular expression.

9 References

Abrahams, P. W., and B. R. Larson. 1997. *UNIX for the Impatient*. 2nd ed. Reading, MA: Addison–Wesley.

Aho, A. V., B. W. Kernighan, and P. J. Weinberger. 1988. *The AWK Programming Language*. Reading, MA: Addison–Wesley.

Allen, A. 1999. Review of Mathematics: From the Birth of Numbers, by Jan Gullberg. *American Mathematical Monthly* 106: 77–85.

Cajori, F. 1928. *A History of Mathematical Notations. Volume I: Notation in Elementary Mathematics*. Chicago: Open Court.

Cox, N. J. 2002. Speaking Stata: On numbers and strings. *Stata Journal* 2: 314–329.

———. 2003. Stata tip 2: Building with floors and ceilings. *Stata Journal* 3: 446–447.

Dauben, J. 2002. Review of The Universal History of Numbers and The Universal History of Computing, Parts I and II. *Notices of the American Mathematical Society* 49: 32–38 and 211–216.

Friedl, J. E. F. 2006. *Mastering Regular Expressions*. 3rd ed. Sebastopol, CA: O'Reilly.

Gullberg, J. 1997. *Mathematics: From the Birth of Numbers*. New York: W. W. Norton.

Ifrah, G. 1998. *The Universal History of Numbers: From Prehistory to the Invention of the Computer*. London: Harvill.

Kernighan, B. W., and R. Pike. 1984. *The UNIX Programming Environment*. Englewood Cliffs, NJ: Prentice Hall.

Menninger, K. 1969. *Number Words and Number Symbols: A Cultural History of Numbers*. Cambridge, MA: MIT Press.

Pólya, G. 1957. *How to Solve It: A New Aspect of Mathematical Method*. 2nd ed. Princeton, NJ: Princeton University Press.

Raymond, E. S. 2004. *The Art of UNIX Programming*. Boston, MA: Addison–Wesley.

Turner, K. S. 2005. FAQ: What are regular expressions and how can I use them in Stata? http://www.stata.com/support/faqs/data/regex.html.

Zebrowski, E., Jr. 2001. Review of The Universal History of Numbers: From Prehistory to the Invention of the Computer, by Georges Ifrah. *Isis* 92: 584–585.

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 (2011)
11, Number 1, pp. 143–144

Stata tip 94: Manipulation of prediction parameters for parametric survival regression models

Theresa Boswell
StataCorp
College Station, TX
tboswell@stata.com

Roberto G. Gutierrez
StataCorp
College Station, TX
rgutierrez@stata.com

After fitting a parametric survival regression model using `streg` (see [ST] **streg**), predicting the survival function for the fitted model is available with the `predict` command with the `surv` option. Some users may wish to alter the parameters used by `predict` to compute the survival function for a specific time frame or combination of covariate values.

Manipulation of the prediction parameters can be done directly by altering the variables that `predict` uses to calculate the survival function. However, it is good practice to create a copy of the variables before making any changes so that we can later return variables to their original forms.

This is best illustrated by an example. Using `cancer.dta` included with Stata, we can fit a simple Weibull model with one covariate, `age`:

```
. sysuse cancer
. streg age, dist(weibull)
```

Suppose that we want to obtain the predicted survival function for a specific time range and age value. The time variables used by `predict` to calculate the survival function are stored in variables `_t` and `_t0`, as established by `stset`. Before making any changes, we must first create a copy of these time variables and of our covariate `age`. We can use the `clonevar` command to create a copy. The advantage of using `clonevar` over `generate` is that `clonevar` creates an exact replica of each original variable, including its labels and other properties.

```
. clonevar age_orig = age
. clonevar t_orig = _t
. clonevar t0_orig = _t0
```

Now that we have a copy of the original variables, we are free to manipulate parameters. Let's assume that we want predictions of the survival function for individuals entering the study at age 75 over the time range [0,20]. To alter the time variables, we can use the `range` command to replace `_t` with an evenly spaced grid from 0 to 20:

```
. drop _t
. range _t 0 20
```

The `_t0` variable needs to be set to 0 (for obtaining unconditional survival), and `age` should be set to 75 for all observations:

```
. replace _t0 = 0
. replace age = 75
```


The prediction will now correspond to the survival function for an individual entering the study at age 75 over a time range of 0 to 20. The `predict` command with option `surv` will return the predicted survival function.

```
. predict s, surv
```

To view the predicted values, type

```
. list _t0 _t s
```

or you can graph the survival function by typing

```
. twoway line s _t
```

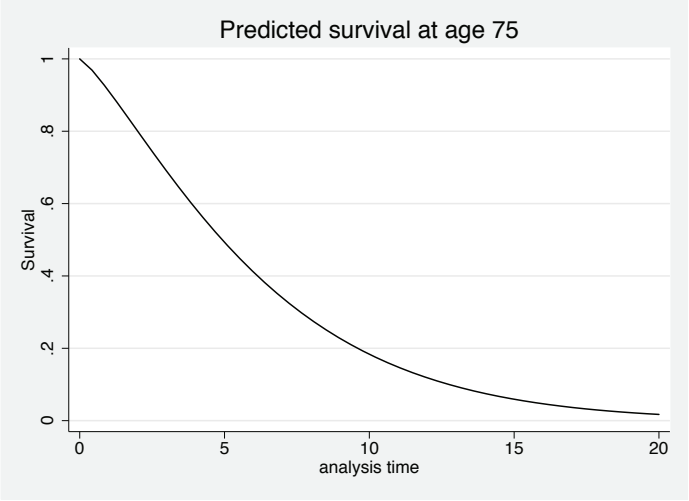


Figure 1. Predicted survival function

Now that we have the predicted values we want, it is prudent to replace all changed variables with their original forms. To do this, we will use the copies we created at the beginning of this example.

```
. replace age = age_orig
. replace _t = t_orig
. replace _t0 = t0_orig
```

There are many cases in which one may wish to manipulate the predicted survival function after `streg` and in which the steps in this tip can be followed to calculate the desired predictions.

The Stata Journal (2011)
11, Number 1, pp. 145–148

Stata tip 95: Estimation of error covariances in a linear model

Nicholas J. Horton
Department of Mathematics and Statistics
Clark Science Center
Smith College
Northampton, MA
nhorton@smith.edu

1 Introduction

A recent review (Horton 2008) of the second edition of *Multilevel and Longitudinal Modeling Using Stata* (Rabe-Hesketh and Skrondal 2008) decried the lack of support in previous versions of Stata for models within the `xtmixed` command that directly estimate the variance–covariance matrix (akin to the REPEATED statement in SAS PROC MIXED). In this tip, I describe how support for these models is now available in Stata 11 (see also `help whatsnew10to11`) and demonstrate its use by replication of an analysis of a longitudinal dental study using an unstructured covariance matrix.

2 Model

I use the notation of Fitzmaurice, Laird, and Ware (2004, chap. 4 and 5) to specify linear models of the form $E(Y_i) = X_i\beta$, where Y_i and X_i denote the vector of responses and the matrix of covariates, respectively, for the ith subject, where $i = 1, \ldots, N$. Assume that each subject has up to n observations on a common set of times. The response vector Y_i is assumed to be multivariate normal with covariance given by $\Sigma_i(\theta)$, where θ is a vector of covariance parameters. If an unstructured covariance matrix is assumed, then there will be $n \times (n+1)/2$ covariance parameters. Restricted maximum-likelihood estimation is used.

3 Example

I consider data from an analysis of a study of dental growth, described on page 184 of Fitzmaurice, Laird, and Ware (2004). Measures of distances (in mm) were obtained on 27 subjects (11 girls and 16 boys) at ages 8, 10, 12, and 14 years.

3.1 Estimation in SAS

Below I give SAS code to fit a model with the mean response unconstrained over time (3 degrees of freedom) and main effect for gender as well as an unstructured working covariance matrix (10 parameters):


```
proc mixed data=one;
        class id time;
        model y = time female / s;
        repeated time / type=un subject=id r;
run;
```

This code generates the following output:

```
The Mixed Procedure
                  Model Information
Data Set                      WORK.ONE
Dependent Variable            y
Covariance Structure          Unstructured
Subject Effect                id
Estimation Method             REML
Residual Variance Method      None
Fixed Effects SE Method       Model-Based
Degrees of Freedom Method     Between-Within

           Dimensions
Covariance Parameters            10
Columns in X                      6
Columns in Z                      0
Subjects                         27
Max Obs Per Subject               4

            Estimated R Matrix for id 1
 Row        Col1        Col2        Col3        Col4
   1      5.3741      2.7887      3.8442      2.6242
   2      2.7887      4.2127      2.8832      3.1717
   3      3.8442      2.8832      6.4284      4.3024
   4      2.6242      3.1717      4.3024      5.3751

                      Solution for Fixed Effects
                                 Standard
Effect       time    Estimate       Error      DF    t Value    Pr > |t|
Intercept             26.9258      0.5376      25      50.08      <.0001
time          8       -3.9074      0.4514      25      -8.66      <.0001
time         10       -2.9259      0.3466      25      -8.44      <.0001
time         12       -1.4444      0.3442      25      -4.20      0.0003
time         14             0           .       .       .         .
female               -2.0452      0.7361      25      -2.78      0.0102
```

3.2 Estimation in Stata

The equivalent model can now be fit in Stata 11:

```
. use http://www.math.smith.edu/labs/denttall
. xtmixed y ib14.time female, || id:, nocons residuals(un, t(time)) var
```

The `xtmixed` command yields the equivalent output:

```
Mixed-effects REML regression                   Number of obs      =       108
Group variable: id                              Number of groups   =        27
                                                Obs per group: min =         4
                                                               avg =       4.0
                                                               max =         4
                                                Wald chi2(4)       =    101.50
Log restricted-likelihood =  -212.4093          Prob > chi2        =    0.0000
```

y	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
time						
8	-3.907407	.4513647	-8.66	0.000	-4.792066	-3.022749
10	-2.925926	.3466401	-8.44	0.000	-3.605328	-2.246524
12	-1.444444	.3441962	-4.20	0.000	-2.119057	-.7698322
female	-2.045172	.736141	-2.78	0.005	-3.487982	-.6023627
_cons	26.92581	.5376092	50.08	0.000	25.87212	27.97951

Random-effects Parameters	Estimate	Std. Err.	[95% Conf.	Interval]
id: (empty)				
Residual: Unstructured				
var(e8)	5.374086	1.510892	3.097379	9.324271
var(e10)	4.21272	1.201038	2.409277	7.366114
var(e12)	6.428418	1.810989	3.700897	11.16609
var(e14)	5.375108	1.608682	2.989761	9.663575
cov(e8,e10)	2.788773	1.112924	.6074823	4.970064
cov(e8,e12)	3.844272	1.392097	1.115811	6.572732
cov(e8,e14)	2.624241	1.207689	.2572134	4.991268
cov(e10,e12)	2.883246	1.183372	.5638802	5.202612
cov(e10,e14)	3.171762	1.153809	.9103389	5.433186
cov(e12,e14)	4.302404	1.499388	1.363657	7.24115

```
LR test vs. linear regression:       chi2(9) =    54.59   Prob > chi2 = 0.0000

Note: The reported degrees of freedom assumes the null hypothesis is not on
      the boundary of the parameter space.  If this is not true, then the
      reported test is conservative.
```

Several points are worth noting:

1. The default output from `xtmixed` provides estimates of variability as well as confidence intervals for the covariance parameter estimates.

2. Considerable flexibility regarding additional covariance structures is provided by the `residuals()` option (including exchangeable, autoregressive, and moving-average structures).

3. Specifying a `by()` variable within the `residuals()` option can allow separate estimation of error covariances by group (for example, in this setting, separate estimation of the structures for men and for women).

4. The `ib14` specification for the time factor variable facilitates changing the reference grouping to match the SAS defaults.

5. Dropping the `var` option will generate correlations (which may be more interpretable if the variances change over time).

For the dental example, we see that the estimated correlation is lowest between the observations that are farthest apart ($r = 0.49$) and generally higher for shorter intervals.

```
corr(e8,e10) |  .5861106   .1306678    .2743855    .7863675
corr(e8,e12) |  .6540481   .1129091    .3761828    .8239756
corr(e8,e14) |  .4882675   .1518479    .1420355    .7280491
corr(e10,e12) |  .5540493   .1370823    .2322075    .7665423
corr(e10,e14) |  .6665393   .1115412    .3894063    .8330066
corr(e12,e14) |  .7319232   .0930009    .4931844     .868134
```

4 Summary

Modeling the associations between observations on the same subject using mixed effects and an unstructured covariance matrix is a flexible and attractive alternative to a random-effects model with cluster–robust standard errors. This is particularly useful when the number of measurement occasions is relatively small, and measurements are taken at a common set of occasions for all subjects. The addition of support for this model within `xtmixed` in Stata 11 is a welcome development.

5 Acknowledgments

Thanks to Kristin MacDonald, Roberto Gutierrez, Garrett Fitzmaurice, and the anonymous reviewers for helpful comments on an earlier draft.

References

Fitzmaurice, G. M., N. M. Laird, and J. H. Ware. 2004. *Applied Longitudinal Analysis*. Hoboken, NJ: Wiley.

Horton, N. J. 2008. Review of Multilevel and Longitudinal Modeling Using Stata, Second Edition, by Sophia Rabe-Hesketh and Anders Skrondal. *Stata Journal* 8: 579–582.

Rabe-Hesketh, S., and A. Skrondal. 2008. *Multilevel and Longitudinal Modeling Using Stata*. 2nd ed. College Station, TX: Stata Press.

The Stata Journal (2011)
11, Number 1, pp. 149–154

Stata tip 96: Cube roots

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
n.j.cox@durham.ac.uk

1 Introduction

Plotting the graph of the cube function $x^3 = y$ underlines that it is single-valued and defined for arguments everywhere on the real line. So also is the inverse or cube root function $x = y^{1/3} = \sqrt[3]{y}$. In Stata, you can see a graph of the cube function by typing, say, `twoway function x^3, range(-5 5)` (figure 1). To see a graph of its inverse, imagine exchanging the axes. Naturally, you may want to supply other arguments to the `range()` option.

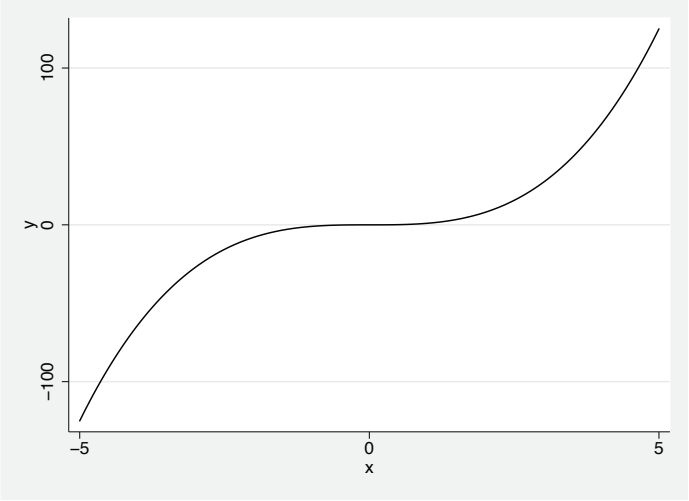


Figure 1. The function x^3 for $-5 \leq x \leq 5$

Otherwise put, for any $a \geq 0$, we can write

$$(-a)(-a)(-a) = -a^3 \qquad (a)(a)(a) = a^3$$

so that cube roots are defined for negative, zero, and positive cubes alike.

This concept might well be described as elementary mathematics. The volume of literature references, say, for your colleagues or students, could be multiplied; I will single

out Gullberg (1997) as friendly but serious and Axler (2009) as serious but friendly. Elementary or not, Stata variously does and does not seem to know about cube roots:

```
. di 8^(1/3)
2
. di -8^(1/3)
-2
. di(-8)^(1/3)
.
. set obs 1
obs was 0, now 1
. gen minus8 = -8
. gen curtminus8 = minus8^(1/3)
(1 missing value generated)
```

This tip covers a bundle of related questions: What is going on here? In particular, why does Stata return missing when there is a perfectly well-defined result? How do we get the correct answer for negative cubes? Why should we ever want to do that? Even if you never do in fact want to do that, the examples above raise details of how Stata works that you should want to understand.

Those who know about complex analysis should note that we confine ourselves to real numbers throughout.

2 Calculation of cube roots

To Stata, cube roots are not special. As is standard with mathematical and statistical software, there is a dedicated square-root function `sqrt()`; but cube roots are just powers, and so they are obtained by using the `^` operator. I always write the power for cube roots as (1/3), which ensures reproducibility of results and ensures that Stata does the best it can to yield an accurate answer. Experimenting with 8 raised to the powers 0.33, 0.333, 0.3333, and so forth will show that you would incur detectable error even with what you might think are excellent approximations. The parentheses around 1/3 are necessary to ensure that the division occurs first, before its result is used as a power.

What you understand from your study of mathematics is not necessarily knowledge shared by Stata. The examples of cube rooting -8 and 8 happen to have simple integer solutions -2 and 2, but even any appearance that Stata can work this out as you would is an illusion:

```
.  di 8^(1/3)
2
.  di %21x 8^(1/3)
+1.fffffffffffffX+000
.  di %21x 2
+1.0000000000000X+001
```

Showing here results in hexadecimal format (Cox 2006) reveals what might be suspected. No part of Stata recognizes that the answer should be an integer. The problem is being treated as one in real (not integer) arithmetic, and the appearance that 2 is the solution is a pleasant side effect of Stata's default numeric display format. Stata's answer is in fact a smidgen less than 2. What is happening underneath? I raised this question with William Gould of StataCorp and draw on his helpful comments here. He expands further on the matter within the Stata blog; see "How Stata calculates powers" at http://blog.stata.com/2011/01/20/how-stata-calculates-powers/.

The main idea here is that Stata is using logarithms to do the powering. This explains why no answer is forthcoming for negative arguments in the `generate` statement: because the logarithm is not defined for such arguments, the calculation fails at the first step and is fated to yield missings.

We still have to explain why `di -8^(1/3)` yields the correct result for the cube root of −8. That is just our good fortune together with convenient formatting. Stata's precedence rules ensure that the negation is carried out last, so this request is equivalent to `-(8^(1/3))`, a calculation that happens to have the same answer. We would not always be so lucky: for the same reason, `di -2^2` returns −4, not 4 as some might expect.

The matter is more vexed yet. Not only does 1/3 have no exact decimal representation, but also, more crucially, it has no exact binary representation. It is easy enough to trap 0 as a special case so that Stata does not fail through trying to calculate ln 0. But Stata cannot be expected to recognize 1/3 as its own true self. The same goes for other odd integer roots (powers of 1/5, 1/7, and so forth) for which the problem also appears.

To get the right result, human intervention is required to spell out what you want. There are various work-arounds. Given a variable `y`, the cube root is in Stata

```
cond(y < 0, -((-y)^(1/3)), y^(1/3))
```

or

```
sign(y) * abs(y)^(1/3)
```

The `cond()` function yields one of two results, depending on whether its first argument is nonzero (true) or zero (false). See Kantor and Cox (2005) for a tutorial if desired. The `sign()` function returns −1, 0, or 1 depending on the sign of its argument. The `abs()` function returns the absolute value or positive square root. The second of the two solutions just given is less prone to silly, small errors and extends more easily to Mata. The code in Mata for a scalar `y` is identical. For a matrix or vector, we need the generalization with elementwise operators,

```
sign(y) :* abs(y):^(1/3)
```

3 Applications of cube roots

Cube roots do not have anything like the utility, indeed the natural roles, of logarithms or square roots in data analysis, but they do have occasional uses. I will single out three reasons why.

First, the cube root of a volume is a length, so if the problem concerns volumes, dimensional analysis immediately suggests cube roots as a simplifying transformation. A case study appears in Cox (2004).

Second, the cube root is also a good transformation yielding approximately normal distributions from gamma or gamma-like distributions. See, for example, McCullagh and Nelder (1989, 288–289). Figure 2 puts normal probability plots for some raw and transformed chi-squared quantiles for 4 degrees of freedom side by side.

```
. set obs 99
. gen chisq4 = invchi2(4, _n/100)
. qnorm chisq4, name(g1)
. gen curt_chisq4 = chisq4^(1/3)
. qnorm curt_chisq4, name(g2)
. graph combine g1 g2
```

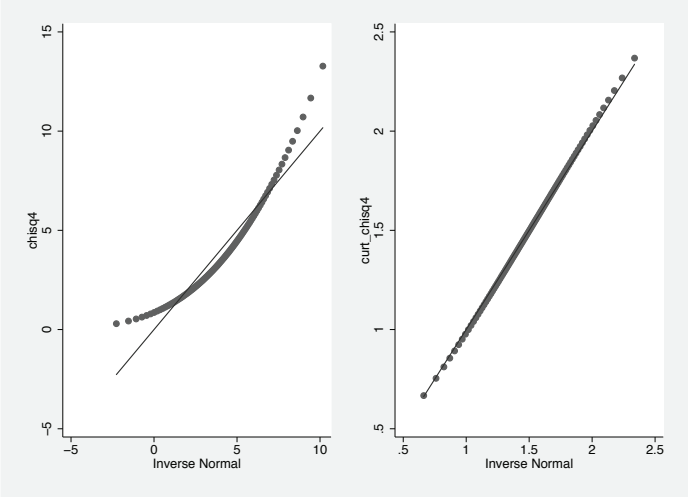


Figure 2. Ninety-nine quantiles from a chi-squared distribution with 4 degrees of freedom are distinctly nonnormal, but their cube roots are very nearly normal

The cube root does an excellent job with a distinctly nonnormal distribution. It has often been applied to precipitation data, which are characteristically right-skewed and sometimes include zeros (Cox 1992).

Third, beyond these specific uses, the cube root deserves wider attention as the simplest transformation that changes distribution shape but is easily applicable to values with varying signs. It is an odd function—an odd function f is one for which

$f(-x) = -f(x)$—but in fact, it treats data evenhandedly by preserving the sign (and in particular, mapping zeros to zeros).

There are many situations in which response variables in particular can be both positive and negative. This is common whenever the response is a balance, change, difference, or derivative. Although such variables are often skew, the most awkward property that may invite transformation is usually heavy (long or fat) tails, high kurtosis in one terminology. Zero usually has a strong substantive meaning, so that we should wish to preserve the distinction between negative, zero, and positive values. (Celsius or Fahrenheit temperatures do not really qualify here, because their zero points are statistically arbitrary, for all the importance of whether water melts or freezes.)

From a different but related point of view, there are frequent discussions in statistical (and Stata) circles of what to do when on other grounds working on logarithmic scales is indicated, but the data contain zeros (or worse, negative values). There is no obvious solution. Working with logarithms of $(x + 1)$ or more generally $(x + k)$, k being large enough to ensure that $x + k$ is always positive, variously appeals and appalls. Even its advocates have to admit to an element of fudge. It certainly does not treat negative, zero, and positive values symmetrically.

Other solutions that do precisely that include $\text{sign}(x)\ \ln(|x| + 1)$ and $\text{asinh}(x)$, although such functions may appear too complicated or esoteric for presentation to some intended audiences. As emphasized earlier, the cube root is another and simpler possibility. It seems unusual in statistical contexts to see its special properties given any mention, but see Ratkowsky (1990, 125) for one oblique exception.

One possible application of cube roots is whenever we wish to plot residuals but also to pull in the tails of large positive and negative residuals compared with the middle of the distribution around zero. See Cox (2008) for pertinent technique. In this and other graphical contexts, the main question is not whether cube roots yield approximately normal distributions, but simply whether they make data easier to visualize and to think about.

These issues extend beyond cube roots. As hinted already, higher odd integer roots (fifth, seventh, and so forth) have essentially similar properties although they seem to arise far less frequently in data analysis. Regardless of that, it is straightforward to define powering of negative and positive numbers alike so long as we treat different signs separately, most conveniently by using `sign()` and `abs()` together with the power operator. That is, the function may be defined as $-(-y)^p$ if $y < 0$, and y^p otherwise.

References

Axler, S. 2009. *Precalculus: A Prelude to Calculus*. Hoboken, NJ: Wiley.

Cox, N. J. 1992. Precipitation statistics for geomorphologists: Variations on a theme by Frank Ahnert. *Catena* 23 (Suppl.): 189–212.

———. 2004. Speaking Stata: Graphing model diagnostics. *Stata Journal* 4: 449–475.

———. 2006. Stata tip 33: Sweet sixteen: Hexadecimal formats and precision problems. *Stata Journal* 6: 282–283.

———. 2008. Stata tip 59: Plotting on any transformed scale. *Stata Journal* 8: 142–145.

Gullberg, J. 1997. *Mathematics: From the Birth of Numbers*. New York: W. W. Norton.

Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the cond() function. *Stata Journal* 5: 413–420.

McCullagh, P., and J. A. Nelder. 1989. *Generalized Linear Models*. 2nd ed. London: Chapman & Hall/CRC.

Ratkowsky, D. A. 1990. *Handbook of Nonlinear Regression Models*. New York: Marcel Dekker.

The Stata Journal (2011)
11, Number 1, p. 155

Software Updates

srd3_1: One-step Welsch bounded-influence estimator. R. Goldstein. *Stata Technical Bulletin* 2: 26. Reprinted in *Stata Technical Bulletin Reprints*, vol. 1, pp. 176.

The program and help file have been updated to Stata 11.1.

srd13_2: Maximum R-squared and pure error lack-of-fit test. R. Goldstein. *Stata Journal* 6: 284. *Stata Technical Bulletin* 9: 24–28. Reprinted in *Stata Technical Bulletin Reprints*, vol. 2, pp. 178–183.

The program has been updated to Stata 11.1. It now supports factor variables and can also be used after `areg` and the user-written command `ivreg2`.

st0213_1: Variable selection in linear regression. C. Lindsey and S. Sheather. *Stata Journal* 10: 650–669.

A bug that prevented the specification of multiple variables in the `fix()` option has been fixed.

