The Stata Journal (2009)
9, Number 3, pp. 466–477

Multiple imputation of missing values: Further update of ice, with an emphasis on categorical variables

Patrick Royston
Hub for Trials Methodology Research
MRC Clinical Trials Unit and University College London
London, UK
pr@ctu.mrc.ac.uk

Abstract. Multiple imputation of missing data continues to be a topic of considerable interest and importance to applied researchers. In this article, the `ice` package for multiple imputation by chained equations (also known as fully conditional specification) is further updated. Special attention is paid to categorical variables. The relationship between `ice` and the new multiple-imputation system in Stata 11 is clarified.

Keywords: st0067_4, multiple imputation, chained equations, categorical variables, negative binomial distribution, ice, uvis, mi

1 Introduction

Royston (2004) introduced `mvis`, an implementation for Stata of multiple imputation by chained equations (MICE), a method of multiple multivariate imputation of missing values under missing-at-random (MAR) assumptions. The name of the main command was changed to `ice` (imputation by chained equations) in Royston (2005a). Two updates of `ice` have followed (Royston 2005b, 2007). This article presents a further update; the focus is on categorical variables. The main features are a considerable simplification of how imputation models for categorical variables are specified and revision of the `conditional()` option.

The `ice` system comprises three ado-files: `ice`, `ice_`, and `uvis`. Previous components `micombine`, `mijoin`, `misplit`, and `ice_reformat` are out of date and have been removed. `micombine` has been superseded by a more comprehensive command, `mim` (Carlin, Galati, and Royston 2008), itself recently updated with new features (Royston, Carlin, and White 2009).

Before describing the new features, it is important to clarify the impact of the Stata 11 multiple-imputation system on `ice`.

2 ice, uvis, and the Stata 11 multiple-imputation system

The `ice` program was written for Stata 9 and above to perform imputation via chained equations (van Buuren, Boshuizen, and Knook 1999). On 27 July 2009, Stata 11 was released, bearing a major new feature: the `mi` system for multiple imputation and estimation of models with multiply imputed data. The system comprises a new database architecture for imputed datasets; utilities for manipulating, checking, and validating such datasets; a sequence of commands for doing imputation, `mi impute`; and a command for combining estimation results using Rubin's rules, `mi estimate`; see the *Stata 11 Multiple-Imputation Reference Manual* (StataCorp 2009) for details. Many (but not all) of the univariate imputation models available in `uvis` are replicated in new commands of the form `mi impute` *XXX*, where *XXX* is a keyword, such as `regress` for linear regression. Multivariate imputation in Stata 11 can be performed using `mi impute monotone` when the missingness pattern is monotone and using `mi impute mvn` when the missingness pattern is arbitrary. `mi impute monotone` implements a noniterative imputation method based on a sequence of independent univariate conditional specifications. It is similar to the implementation of option `monotone` of `ice`. `mi impute mvn` performs multivariate imputation assuming that the data have a multivariate normal distribution. It implements the NORM method of Schafer (1997)—an iterative Markov chain Monte Carlo method (data augmentation) based on multivariate normality. `ice` implements an alternative iterative multivariate imputation method based on a sequence of univariate full conditional specifications, also known as imputation via chained equations. Thus `ice` is not replicated in Stata 11 and is still needed for performing multiple imputation by chained equations for data with arbitrary patterns of missingness.

The `mi import ice` and `mi export ice` commands in Stata 11 make it easy to transport data between the existing `ice` data format and the official `mi` data format introduced in Stata 11. An intermediate step in integrating `ice` more completely into Stata 11 is a program called `mi ice`. The files can be installed from my web page (`net from http://www.homepages.ucl.ac.uk/~ucakjpr/stata/`) under the heading `mi_ice`. The next step is the development of a full-featured new command for Stata 11, likely to be called `mi impute ice`. Because in many people's eyes the flexibility of fully conditional specification embedded in the MICE algorithm offers several advantages over the multivariate normal approach, I expect `ice` and its sequels to continue to be used and useful in Stata 11.

3 Syntax

`ice` *mainvarlist* [*if*] [*in*] [*weight*] [, `boot`[`(`*varlist*`)`] `by(`*varlist*`)` `cc(`*varlist*`)` `clear` `cmd(`*cmdlist*`)` `conditional(`*condlist*`)` `cycles(`#`)` `dropmissing` `dryrun` `eq(`*eqlist*`)` `eqdrop(`*eqdroplist*`)` `genmiss(`*string*`)` `id(`*newvar*`)` `interval(`*intlist*`)` `m(`#`)` `match`[`(`*varlist*`)`] `monotone` `noconstant` `nopp` `noshoweq` `noverbose` `nowarning` `on(`*varlist*`)` `orderasis` `passive(`*passivelist*`)` `persist` `restrict(`[*varname*] [*if*]`)` `saving(`*filename* [`, replace`]`)` `seed(`#`)` `substitute(`*sublist*`)` `trace(`*trace_filename*`)`]

where a typical element of *mainvarlist* is [`i.` | `m.` | `o.`] *varname*.

`uvis` *regression_cmd* {*yvar* | *llvar ulvar*} *xvarlist* [*if*] [*in*] [*weight*] `,`
`gen(`*newvar*`)` [`boot match by(`*varlist*`) noconstant nopp noverbose replace`
`restrict(`[*varname*] [*if*]`) seed(`#`)`]

where *regression_cmd* may be `intreg`, `logistic`, `logit`, `mlogit`, `ologit`, `nbreg`, or `regress`. All weight types supported by *regression_cmd* are allowed. *llvar* and *ulvar* are required with `uvis intreg`. Variables imputed using `nbreg` must be nonnegative but are not restricted to integer values.

4 What is new?

The principal changes to `ice` (version 1.7.3) and `uvis` (version 1.5.5) compared with the December 2007 release (Royston 2007) (versions 1.4.4 and 1.2.7, respectively) are as follows:

1. `ice` and `uvis` now require Stata 10.1. (Strictly speaking, only the new negative binomial regression option requires Stata 10.1; all other features work under Stata 9.2 or higher.)
2. `ice` understands an abbreviated syntax for multilevel categorical variables to be imputed using `ologit` or `mlogit`. This important feature is described in detail below.
3. Negative binomial regression is available via the `cmd(`*varlist*`: nbreg)` option in `ice`, or via `uvis nbreg`, to impute count or count-like data. Typically, such data a) are integer-valued, b) are nonnegative, and c) have variance exceeding the mean. Noninteger variables are allowed, although their missing values are imputed as integers. Negative values are disallowed.
4. The syntax and operation of the `conditional()` option of `ice` have changed substantially (as described below).
5. A `by()` option has been added to `ice` and `uvis` to support imputation in independent subsets of the data (as described below).
6. A `restrict()` option has been added to `ice` and `uvis` to allow one to fit models on a specified subsample but impute missing data for the entire estimation sample (as described below).
7. A `clear` option has been added to `ice` to allow the imputed data to reside in memory without (yet) having been (manually) saved to a file using Stata's `save` command.
8. An `eqdrop()` option has been added to `ice` to delete variables from prediction equations.
9. A `persist` option has been added to `ice` to ignore errors from `uvis` when imputing a "difficult" variable.

5 Options for ice and uvis

Only new or changed options are described.

5.1 Options for ice

`clear` clears the original data from memory and loads the imputed dataset. Unless the `saving()` option is also specified, the data in memory are not permanently saved; this must then be done manually using the `save` or the `saveold` command.

`conditional(`*condlist*`)` invokes conditional imputation. Each item of *condlist* has the form *varlist*`:` `if` *condition*. Items are separated by a backslash (\). Members of *varlist* are imputed only for the subset of observations for which `if` *condition* is true (i.e., *condition* evaluates to a nonzero quantity). Observations on all members of *varlist* for which `if` *condition* is false (i.e., *condition* evaluates to zero) are left unchanged. *condition* must be a Stata expression constructed so that `if` *condition* is meaningful and valid for the current dataset. Note that variables appearing in *condition* may be members of *mainvarlist* or merely variables in the dataset. This is the only situation in which variables that do not appear in *mainvarlist* may be used in an `ice` command. Examples of its use are given in the help file.

`eqdrop(`*eqdroplist*`)` deletes variables from prediction equations. The syntax of *eqdroplist* is *varname1*`:` *varlist1* [`,` *varname2*`:` *varlist2* ...], where each *varname#* (or *varlist#*) is a member (or subset) of *mainvarlist*. One can only remove predictors from equations for variables with missing values (although trying to remove predictors from nonexistent equations is not a fatal error; an information message is issued). Variable names prefixed by `i.` are allowed, provided that the names were prefixed by `i.`, `m.`, or `o.` in *mainvarlist*. They are translated to the corresponding dummy variables created by `xi:`.

`noverbose` suppresses the display of the imputation number (as #) and cycle number within imputations (as `.`), which show the progress of the imputations.

`persist` causes `ice` to ignore errors raised by `uvis` when trying to impute a "difficult" variable or impute with a model that is difficult to fit to the data at hand. Trying to impute a difficult variable by using the `ologit` or `mlogit` command is the most common cause of failure. By default, `ice` stops with an error message. With `persist`, `ice` continues to the next variable to be imputed, not updating the variable that raised an error. Often, by chance, the same variable is successfully updated in a subsequent cycle, and no damage is done to the imputation process.

If the error for a given variable appears in every cycle, you should consider changing the prediction equation for that variable, because its imputed values are unlikely to be appropriate.

`restrict(`[*varname*] [*if*]`)` specifies that imputation models be computed using the subsample identified by *varname* and *if*. The subsample is defined by the observations for which *varname*`!=0` that also meet the *if* conditions. Typically, *varname*`=1` defines the subsample and *varname*=`0` indicates observations not belonging to the subsample. For observations whose subsample status is uncertain, *varname* should be set to a missing value; such observations are dropped from the subsample. By default, `ice` fits imputation models and imputes missing values using the sample of observations identified in the [*if*] and [*in*] expressions. The `restrict()` option identifies a subset of this sample to be used for model estimation. Imputation is restricted to the sample identified in the [*if*] and [*in*] expressions. Thus predictions and their associated imputations are made "out of sample" with respect to the subsample defined by `restrict()`. Examples of its use are given in the help file.

5.2 Options for uvis

`noverbose` suppresses nonerror messages while `uvis` is running.

`restrict(`[*varname*] [*if*]`)`; see the equivalent option for `ice`.

6 Simplified syntax for imputing multilevel categorical variables

6.1 Handling multilevel categorical variables

Experience and common sense suggest that correctly handling a multilevel categorical variable, say, x, in `ice` presents problems for users, for three reasons: 1) the user must decide on and specify an imputation model for predicting x, choosing between `mlogit` for unordered variables (the default) and `ologit` for ordered variables; 2) dummy variables corresponding to the levels of x need to be "passively" imputed (i.e., reconstructed from the imputed values of x) following imputation of x; and 3) the dummy variables need to substitute for x in equations for other variables in which x is a predictor. `ice` comprehensively handles these three aspects through the `cmd()`, `passive()`, and `substitute()` options, respectively. Consider the following example, which uses data from a case–control study in leprosy.

Between 1980–1984, a population of about 112,000 people living in Northern Malawi were screened for leprosy (Fine et al. 1986). New cases of leprosy in initially uninfected people were identified during a follow-up period of five years (Pönnighaus et al. 1992). For illustrative purposes, we use data from a substudy in which controls without leprosy at baseline were selected at random from the screened population. The aim is to assess the effect of BCG vaccination (`bcg`) on the incidence of leprosy. Covariates are `age`, `sex`, `house`, and `school`.

```
. use lep
(1:4 unmatched leprosy and bcg)

. tabulate school, generate(s)

         Schooling |      Freq.     Percent        Cum.
-------------------+-----------------------------------
              none |        282       22.19       22.19
     1-5yr primary |        606       47.68       69.87
     6-8yr primary |        350       27.54       97.40
secondary/tertiary |         33        2.60      100.00
-------------------+-----------------------------------
             Total |      1,271      100.00

. tabulate house, generate(h)

                         Housing |      Freq.     Percent        Cum.
---------------------------------+-----------------------------------
                     burnt brick |        240       19.25       19.25
sun-dried bricks or pounded mud |        295       23.66       42.90
                 wattle and daub |        679       54.45       97.35
               temporary shelter |         33        2.65      100.00
---------------------------------+-----------------------------------
                           Total |      1,247      100.00

. ice d age sex bcg school s2 s3 s4 house h2 h3 h4, clear m(5)
> substitute(school:s2 s3 s4, house:h2 h3 h4) passive(s2:school==1 \
> s3:school==2 \ s4:school==3 \ h2:house==1 \ h3:house==2 \ h4:house==3)
> cmd(school:ologit, house:mlogit)

   #missing |
     values |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |      1,186       86.57       86.57
          4 |        146       10.66       97.23
          8 |         38        2.77      100.00
------------+-----------------------------------
      Total |      1,370      100.00

   Variable | Command | Prediction equation
------------+---------+----------------------------------------------------
          d |         | [No missing data in estimation sample]
        age |         | [No missing data in estimation sample]
        sex |         | [No missing data in estimation sample]
        bcg |         | [No missing data in estimation sample]
     school | ologit  | d age sex bcg h2 h3 h4
         s2 |         | [Passively imputed from school==1]
         s3 |         | [Passively imputed from school==2]
         s4 |         | [Passively imputed from school==3]
      house | mlogit  | d age sex bcg s2 s3 s4
         h2 |         | [Passively imputed from house==1]
         h3 |         | [Passively imputed from house==2]
         h4 |         | [Passively imputed from house==3]
------------------------------------------------------------------------------

Imputing ..........1..........2..........3..........4..........5
[note: imputed dataset now loaded in memory]

Warning: imputed dataset has not (yet) been saved to a file
```

The variables `school` and `house` are categorical, each having four levels. `school` is the number of years of schooling and is ordinal, whereas `house` is the type of dwelling and is unordered. Hence `school` may be modeled by using `ologit`, and `house` may be modeled by using `mlogit`.

The specification of the `ice` command involves creating the dummy variables `s2`, `s3`, and `s4` for `school` and `h2`, `h3`, and `h4` for `house`, and using the `passive()` and `substitute()` options to manage the two sets of three dummy variables. The setup takes some effort and looks quite complicated, even in a simple example such as the present one. With more such variables (and `ice` specifications can have many), the complexity and potential for making errors increase considerably.

6.2 The m. and o. prefixes

Now consider the following, which is an identical setup but uses a new, more streamlined syntax:

```
. use lep, clear
(1:4 unmatched leprosy and bcg)
. ice d age sex bcg o.school m.house, clear m(5)
=> xi: ice d age sex bcg school i.school house i.house, cmd(house:mlogit,
> school:ologit) substitute(school:i.school, house:i.house) clear m(5)
i.school          _Ischool_0-3        (naturally coded; _Ischool_0 omitted)
i.house           _Ihouse_0-3         (naturally coded; _Ihouse_0 omitted)
   #missing |
     values |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |      1,186       86.57       86.57
          1 |        146       10.66       97.23
          2 |         38        2.77      100.00
------------+-----------------------------------
      Total |      1,370      100.00
   Variable | Command | Prediction equation
------------+---------+----------------------------------------------------
          d |         | [No missing data in estimation sample]
        age |         | [No missing data in estimation sample]
        sex |         | [No missing data in estimation sample]
        bcg |         | [No missing data in estimation sample]
     school | ologit  | d age sex bcg _Ihouse_1 _Ihouse_2 _Ihouse_3
 _Ischool_1 |         | [Passively imputed from (school==1)]
 _Ischool_2 |         | [Passively imputed from (school==2)]
 _Ischool_3 |         | [Passively imputed from (school==3)]
      house | mlogit  | d age sex bcg _Ischool_1 _Ischool_2 _Ischool_3
  _Ihouse_1 |         | [Passively imputed from (house==1)]
  _Ihouse_2 |         | [Passively imputed from (house==2)]
  _Ihouse_3 |         | [Passively imputed from (house==3)]
----------------------------------------------------------------------------
Imputing ..........1..........2..........3..........4..........5
[note: imputed dataset now loaded in memory]
Warning: imputed dataset has not (yet) been saved to a file
```

`ice` accepts `o.school` and `m.house` as a sufficient specification of how to model these two variables. The program does the necessary work of setting up the required `passive()` and `substitute()` options, creating dummy variables and defining sensible prediction equations. `ice` invokes Stata's `xi:` command to produce the variables labeled in standard fashion, `_Ischool_1`, etc. The "expanded" `ice` command is displayed before the tables of missing values and prediction equations are presented.

Now suppose that we wanted the prediction equation for `school` to be `d age _Ihouse_1 _Ihouse_2 _Ihouse_3` rather than the default equation, which also includes `sex bcg`. To specify this, we use the `i.` prefix within the `eq()` option for `school` to signify the `_I*` variables for `house`:

```
. ice d age sex bcg o.school m.house, eq(school: d age i.house) dryrun
=> xi: ice d age sex bcg school i.school house i.house, cmd(house:mlogit,
> school:ologit) substitute(school:i.school, house:i.house) eq(school: d age
> i.house) dryrun
  (output omitted)
   Variable   Command   Prediction equation
 ---------------------------------------------------------------------------
          d             [No missing data in estimation sample]
        age             [No missing data in estimation sample]
        sex             [No missing data in estimation sample]
        bcg             [No missing data in estimation sample]
     school   ologit    d age _Ihouse_1 _Ihouse_2 _Ihouse_3
 _Ischool_1             [Passively imputed from (school==1)]
 _Ischool_2             [Passively imputed from (school==2)]
 _Ischool_3             [Passively imputed from (school==3)]
      house   mlogit    d age sex bcg _Ischool_1 _Ischool_2 _Ischool_3
  _Ihouse_1             [Passively imputed from (house==1)]
  _Ihouse_2             [Passively imputed from (house==2)]
  _Ihouse_3             [Passively imputed from (house==3)]
 ---------------------------------------------------------------------------
End of dry run. No imputations were done, no files were created.
```

Finally, an additional advantage of the `m.` and `o.` syntax is that the number of missing values is counted correctly. In the original syntax, missing values are multiply counted due to inclusion of the dummy variables with the "parent" variables in *mainvarlist*.

6.3 The i. prefix

The other new feature is the `i.` prefix, which simply applies `xi:` to the variable in question. (Note: Do not confuse the `i.` prefix in `ice` with the same syntax in Stata 11, which indicates a factor variable. At this point, `ice` does not support factor variables; it may do so in the future.) It is important to emphasize that for the `i.` prefix to work correctly with a multilevel categorical variable, the latter must have *no missing data in the estimation sample*, that is, it must be complete except when all other variables in *mainvarlist* have missing values. This condition is checked by `ice` and an error message is issued if it is violated. The `i.` prefix automatically handles the resulting dummy variables in the correct manner, that is, they become predictors for other variables but are not themselves imputed (because they are assumed to have no missing data). If a categorical variable does have missing data, either the `m.` or the `o.` prefix must be used to invoke the machinery needed to create a valid set of imputation models.

To clarify further, consider the following example with the leprosy data:

```
. use lep, clear
(1:4 unmatched leprosy and bcg)
. ice age sex bcg i.school, dryrun
=> xi: ice age sex bcg i.school, dryrun
i.school          _Ischool_0-3        (naturally coded; _Ischool_0 omitted)
   #missing |
     values |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |      1,271       92.77       92.77
          1 |         99        7.23      100.00
------------+-----------------------------------
      Total |      1,370      100.00
99 missing values of variable school found in the estimation sample
variables with an i. prefix must be complete in the estimation sample
you can use an m. or o. prefix to impute incomplete variables of this type
1 specification error(s) found
r(198);
```

Using `i.school` invokes `xi:` as expected. However, because we have not specified what is to happen to the dummy variables `_Ischool_1`, `_Ischool_2`, and `_Ischool_3`, and because they each have 99 missing values, we find that each of them is independently predicted from the other two dummy variables and `age`, `sex`, and `bcg`. This is clearly incorrect. The correct solution is either `ice age sex bcg m.school` or `ice age sex bcg o.school`, depending on how `school` is to be modeled.

7 Conditional imputation—the conditional() option

Consider an (artificial) dataset including the variables `age`, `female`, and `pregnant`, where `age` is continuous, approximately normally distributed and has missing values; `female` is binary (1 for females, 0 for males) and is complete; and `pregnant` is binary (1 for pregnant, 0 for not pregnant) and also has missing values. Such a dataset is supplied in `pregnant.dta`. Suppose that the probability of being pregnant is related to age. Because males cannot be pregnant, we do not wish to impute pregnancy in males; we should therefore impute missing values of `pregnant` using `age` in females only:

```
. use pregnant, clear
(Artificial dataset on pregnancy and age)
. ice age pregnant female, conditional(pregnant: if female==1) dryrun
   #missing │
     values │      Freq.     Percent        Cum.
────────────┼───────────────────────────────────
          0 │        388       77.60       77.60
          1 │        110       22.00       99.60
          2 │          2        0.40      100.00
────────────┼───────────────────────────────────
      Total │        500      100.00
   Variable │ Command │ Prediction equation
────────────┼─────────┼──────────────────────────────────────────────
        age │ regress │ pregnant female
   pregnant │ logit   │ age if female==1
     female │         │ [No missing data in estimation sample]
──────────────────────────────────────────────────────────────────────

End of dry run. No imputations were done, no files were created.
```

The prediction equation for `age` is `pregnant female`, whereas the equation for `pregnant` is just `age` of the females. It is important here that the values of `pregnant` are correctly defined as 0 for males, even though imputation of `pregnant` is not performed for `female==0`. If `pregnant` were set to, say, -1 for males, then `ice` would see three distinct values of `pregnant`. It would try to impute `pregnant` using `mlogit` rather than `logit`, which would cause an error.

More than one conditional imputation can occur in the same run. Suppose, for example, that the dataset included a variable with missing values called `fertile`, giving the result of a female fertility test and coded 1 = fertile, 0 = infertile. In `ice`, one might specify this case as

```
ice age pregnant female fertile, ///
 conditional(pregnant: if female==1 & fertile==1 \ fertile: if female==1) dryrun
```

to reflect that only fertile females can become pregnant and only females have a fertility test. If males had also had a fertility test, the phrase `fertile: if female==1` would have been omitted.

8 Out-of-sample imputation—the restrict() option

The `restrict()` option is designed for situations in which one wishes to fit imputation equations to a subset of the observations but make imputations across the entire dataset. An example is a "validation study" in which a primary dataset is used to determine and estimate a multivariable model of some kind and a secondary dataset is available to test the accuracy of the model. Both the primary and the secondary datasets may have missing values. The combined (primary + secondary) dataset would typically include a variable, say, `primary`, coded as 1 for the primary and 0 for the secondary dataset. A schematic `ice` run to impute missing values out of sample in the secondary dataset is

```
ice mainvarlist, restrict(primary) <other_stuff> ...
```

The imputation models would be estimated on the subset `primary != 0`, and multiple imputation of the subset consisting of all nonmissing values of `primary` would be done. See the `ice` help file for further comments on this option.

9 Imputation in independent subsets—the by() option

Using `by(`*varlist*`)` performs multiple imputation separately for all combinations of the variables in *varlist*. Observations with missing values for any members of *varlist* are excluded.

An application of `by()` is in randomized trials, where interactions as yet unidentified between treatment and patient characteristics (covariates) may be present. If imputation is not done separately for each treatment group, estimates of interactions with treatment in the analysis model are biased toward zero. The `by()` approach may be useful more generally for coping with interactions with a categorical variable.

Common sense must be applied to the `by()` option. For example, if `by(`*varlist*`)` subdivides the dataset too finely, the imputation models may become unstable and imprecise, compromising the quality of the imputations.

10 Conclusion

Development of `ice` continues as new features are requested by users or considered by the author to be worthwhile. Closer integration with Stata 11 `mi` will follow in due course. The syntax for categorical variables should prove particularly helpful for users and for teachers of multiple imputation in Stata.

11 Acknowledgments

Ian White suggested and partly coded the prefix syntax for categorical variables and the negative binomial regression command for `ice` and `uvis`. I am grateful to Dr Paul Fine for permission to use the leprosy data and to Yulia Marchenko for remarks on the Stata 11 `mi` system.

12 References

Carlin, J. B., J. C. Galati, and P. Royston. 2008. A new framework for managing and analyzing multiply imputed data in Stata. *Stata Journal* 8: 49–67.

Fine, P. E. M., J. M. Pönnighaus, N. Maine, J. A. Clarkson, and L. Bliss. 1986. Protective efficacy of BCG against leprosy in northern Malawi. *Lancet* ii: 499–502.

Pönnighaus, J. M., P. E. M. Fine, J. A. C. Sterne, R. J. Wilson, E. Msosa, P. J. K. Gruer, P. A. Jenkins, S. B. Lucas, G. Liomba, and L. Bliss. 1992. Efficacy of BCG vaccine against leprosy and tuberculosis in Northern Malawi. *Lancet* 339: 636–639.

Royston, P. 2004. Multiple imputation of missing values. *Stata Journal* 4: 227–241.

———. 2005a. Multiple imputation of missing values: Update. *Stata Journal* 5: 188–201.

———. 2005b. Multiple imputation of missing values: Update of ice. *Stata Journal* 5: 527–536.

———. 2007. Multiple imputation of missing values: Further update of ice, with an emphasis on interval censoring. *Stata Journal* 7: 445–464.

Royston, P., J. B. Carlin, and I. R. White. 2009. Multiple imputation of missing values: New features for mim. *Stata Journal* 9: 252–264.

Schafer, J. L. 1997. *Analysis of Incomplete Multivariate Data*. Boca Raton, FL: Chapman & Hall/CRC.

StataCorp. 2009. *Stata 11 Multiple-Imputation Reference Manual*. College Station, TX: Stata Press.

van Buuren, S., H. C. Boshuizen, and D. L. Knook. 1999. Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine* 18: 681–694.

About the author

Patrick Royston is a medical statistician with 30 years of experience who has a strong interest in biostatistical methods and in statistical computing and algorithms. He now works in cancer clinical trials and related research issues. Currently, he is focusing on problems of model building and validation with survival data, including prognostic factor studies; on parametric modeling of survival data; on multiple imputation of missing values; and on novel clinical trial designs.