The Stata Journal (2004)
4, Number 3, pp. 227–241

Multiple imputation of missing values

Patrick Royston
Cancer Division
MRC Clinical Trials Unit
222 Euston Road
London NW1 2DA
UK

Abstract. Following the seminal publications of Rubin about thirty years ago, statisticians have become increasingly aware of the inadequacy of "complete-case" analysis of datasets with missing observations. In medicine, for example, observations may be missing in a sporadic way for different covariates, and a complete-case analysis may omit as many as half of the available cases. Hotdeck imputation was implemented in Stata in 1999 by Mander and Clayton. However, this technique may perform poorly when many rows of data have at least one missing value. This article describes an implementation for Stata of the MICE method of multiple multivariate imputation described by van Buuren, Boshuizen, and Knook (1999). MICE stands for multivariate imputation by chained equations. The basic idea of data analysis with multiple imputation is to create a small number (e.g., 5–10) of copies of the data, each of which has the missing values suitably imputed, and analyze each complete dataset independently. Estimates of parameters of interest are averaged across the copies to give a single estimate. Standard errors are computed according to the "Rubin rules", devised to allow for the between- and within-imputation components of variation in the parameter estimates. This article describes five ado-files. `mvis` creates multiple multivariate imputations. `uvis` imputes missing values for a single variable as a function of several covariates, each with complete data. `micombine` fits a wide variety of regression models to a multiply imputed dataset, combining the estimates using Rubin's rules, and supports survival analysis models (`stcox` and `streg`), categorical data models, generalized linear models, and more. Finally, `misplit` and `mijoin` are utilities to interconvert datasets created by `mvis` and by the `miset` program from John Carlin and colleagues. The use of the routines is illustrated with an example of prognostic modeling in breast cancer.

Keywords: st0067, mvis, uvis, micombine, mijoin, misplit, missing data, missing at random, multiple imputation, multivariate imputation, regression modeling

1 Introduction

Following the seminal work of Rubin (1976, 1987), awareness has grown of the need to go beyond "complete-case" analysis of datasets with missing observations. In medicine, for example, it is common for observations to be missing in a sporadic way for different covariates; a complete-case analysis may omit as many as half of the available cases. Clark and Altman (2003) reported a study of prognosis in ovarian cancer in which missing data on ten covariates reduced the complete-case sample size from 1,189 to 518 patients. Imputation of 2,045 missing values, comprising only 17% of the total of $10 \times 1,189 = 11,890$ slots in the data matrix, more than doubled the available sample size.

Hotdeck imputation was implemented in Stata by Mander and Clayton (1999) but may perform poorly when many rows of data have at least one missing value. This happens quite often and occurred in Clark and Altman's dataset just mentioned.

This article describes an implementation for Stata of the "switching regression" method of multiple multivariate imputation described by van Buuren, Boshuizen, and Knook (1999). See van Buuren's article for full details of the theory behind the method and a discussion of how to build the imputation model.

The basic idea of data analysis with multiple imputation is to create a small number, m, of copies of the data, each of which has the missing values suitably imputed. Traditionally, $m = 3$ or 5. Then, each complete dataset is analyzed independently. Estimates of parameters of interest are averaged across the m copies to give a single estimate. Standard errors are computed according to the "Rubin rules" (Rubin 1987). Recently Carlin, Li, Greenwood, and Coffey (2003) have provided a variety of useful tools for managing such imputed datasets and obtaining combined estimates. However, Carlin's routines assume that the imputed datasets have already been created; no algorithms to create imputations are provided.

Old-fashioned imputation typically replaced missing values with the mean or mode of the nonmissing values for that variable. That approach is now regarded as inadequate. For subsequent statistical inference to be valid, it is essential to inject the correct degree of randomness into the imputations and to incorporate that uncertainty when computing standard errors and confidence intervals for parameters of interest.

Here I present five ado-files. `mvis` creates multiple multivariate imputations. `uvis` imputes missing values for a single variable as a function of several covariates, each of the latter having complete data. `micombine` fits a wide variety of regression models to a multiply imputed dataset, combining the estimates using Rubin's rules. `micombine` goes further than Carlin's `mifit` routine in that it supports survival analysis models (`stcox` and `streg`), categorical data models, generalized linear models, and more. Finally, `misplit` and `mijoin` are utilities to interconvert datasets created by `mvis` and by Carlin et al. (2003)'s `miset` routine. I will give examples of the use of the routines and furthermore propose a novel method for selecting the number m of imputations to give acceptable precision for confidence intervals based on the fitted model.

2 Syntax

`mvis` *mainvarlist* `using` *filename* [`if` *exp*] [`in` *range*] [*weight*], `m(`#`)`
  [<u>`boot`</u>[`(`*varlist*`)`] `cc(`*ccvarlist*`)` <u>`cmd`</u>`(`*cmdlist*`)` <u>`cy`</u>`cles(`#`)` <u>`dr`</u>`aw`[`(`*varlist*`)`]
  <u>`g`</u>`enmiss(`*string*`)` <u>`i`</u>`d(`*string*`)` `on(`*varlist*`)` <u>`nocons`</u>`tant` `replace` <u>`se`</u>`ed(`#`)`]

`uvis` *regression_cmd yvar xvarlist* [`if` *exp*] [`in` *range*] [*weight*], `gen(`*newvar*`)` [`boot draw noconstant replace seed(`#`)`]

where *regression_cmd* may be `logistic`, `logit`, `mlogit`, `ologit`, or `regress`. All weight types supported by *regression_cmd* are allowed.

`micombine` *regression_cmd* [*yvar*] *covarlist* [`if` *exp*] [`in` *range*] [*weight*] [`,` `noconstant detail eform(`*string*`) genxb(`*newvar*`) impid(`*varname*`) lrr` *regression_cmd_options*]

where *regression_cmd* may be `clogit`, `cnreg`, `glm`, `logistic`, `logit`, `poisson`, `probit`, `qreg`, `regress`, `rreg`, `xtgee`, `streg`, `stcox`, `ologit`, `oprobit`, or `mlogit`. All weight types supported by *regression_cmd* are allowed.

`mijoin , clear` [`m(`#`) impid(`*varname*`)`]

`misplit , clear` [`m(`#`) impid(`*varname*`)`]

3 Description

`mvis` imputes missing values in *mainvarlist* m times by using "switching regression", an iterative multivariable regression technique. Missing observations are assumed to be missing at random (MAR) or missing completely at random (MCAR); van Buuren, Boshuizen, and Knook (1999) explain these concepts in a clear way. Imputed variables and all other variables not subject to imputation are stored to a new file called *filename*`.dta` by specifying `using` *filename*. Imputed variables are stored in *filename*`.dta` under their original names, overwriting the original variables. The imputed variables are stored in long format; that is, if the original dataset comprised n observations, *filename*`.dta` will contain $m \times n$ observations.

`uvis` performs univariate imputation of missing values in *yvar* based on multiple regression of *yvar* on *xvarlist* combined with random draws from the conditional distribution of the missing observations, given the observed data and covariates, or by prediction matching (see *Remarks*). `uvis` is called repeatedly by `mvis` in a cyclical fashion to perform multivariate imputation. `uvis` does not create a new dataset but saves the imputed variable in *newvar*, as defined by the `gen(`*newvar*`)` option.

`micombine` estimates the parameters of a regression of *yvar* on *covarlist*. The type of model is determined by *regression_cmd*. Parameter estimates are combined by applying Rubin's rules across several imputations obtained previously by `mvis`.

`mijoin` converts datasets identified by Carlin's `miset` routine to `mvis` format for analysis by `micombine`. The component datasets are stacked (joined vertically). The data must first be `miset`.

`misplit` converts a dataset prepared by `mvis` to `miset` format for analysis by `mifit` and other utilities. The dataset must first be read into memory.

4 Options

4.1 Options for mvis

`m(`*#*`)` sets the number of imputations required (the minimum is 1, with no upper limit).

`boot`[`(`*varlist*`)`] specifies that each member of *varlist* be imputed with the `boot` option of `uvis`. If `(`*varlist*`)` is omitted, all relevant variables are imputed with the `boot` option of `uvis`.

`cc(`*ccvarlist*`)` prevents imputation in members of *mainvarlist* for which any member of *ccvarlist* has a missing value. `cc()` signifies "complete case". This is a convenience feature only and could be achieved (cumbersomely) by using `if` *ccvar1* `!=. &` *ccvar2* `!=. ...`. Note that variables in *ccvarlist* are used for imputation only if they also appear in *mainvarlist*.

`cmd(`*cmdlist*`)` defines the regression commands to be used for each variable in *mainvarlist* when it becomes the dependent variable in the switching regression procedure used by `uvis` (see *Remarks*). The first item in *cmdlist* may be a command, such as `regress`, or may have the syntax *vars*`:`*cmd*, specifying that command *cmd* is relevant to all the variables in *vars*. Subsequent items in *cmdlist* must follow the latter syntax. For unordered categorical variables with at least three levels (e.g., blood group), *cmd* should be specified as `mlogit` to ensure sensible imputations by using multinomial logistic regression. The default *cmd* for a variable is `logit` when there are two distinct values, `mlogit` when there are 3–5 values, and `regress` otherwise.

`cycles(`*#*`)` determines the number of cycles of regression switching to be carried out. The default # is 10.

`draw`[`(`*varlist*`)`] specifies that each member of *varlist* be imputed with the `draw` option of `uvis`. If `(`*varlist*`)` is omitted, all relevant variables are imputed with the `draw` option of `uvis`.

`genmiss(`*string*`)` creates an indicator variable for the missing data in any variable in *varlist* for which at least one value has been imputed. The indicator variable is set to missing for observations excluded by `if`, `in`, etc. The indicator variable for *xvar* is named *stringxvar*.

`id(`*string*`)` creates a variable called *string* containing the original sort order of the data. The default *string* is `_i`.

`on(`*varlist*`)` changes the operation of `mvis` in a major way. With this option, `uvis` imputes each member of *mainvarlist* univariately on *varlist*. This provides a convenient way of producing multiple imputations when imputation for each variable in *mainvarlist* is to be done univariately on a set of complete predictors.

`noconstant` suppresses the regression constant in all regressions.

`replace` permits *filename* to be overwritten with new data.

`seed(`*#*`)` sets the random number seed to *#*. To reproduce a set of imputations, the same random number seed should be used. The default *#* is 0, meaning no seed is set by the program.

4.2 Options for uvis

`gen(`*newvar*`)` creates *newvar* to hold the original (nonmissing) and imputed (originally missing) values of *yvar*.

`boot` invokes a bootstrap method for creating imputed values (see *Remarks*).

`draw` draws imputations at random from the posterior distribution of the missing values of *yvar*, conditional on the observed values and the members of *xvarlist*. The default method of imputation is prediction matching (see *Remarks*).

`noconstant` suppresses the regression constant in all regressions.

`replace` permits *newvar* (see `gen(`*newvar*`)`) to be overwritten with new data.

`seed(`*#*`)` sets the random number seed to *#*. See *Remarks* for comments on how to ensure reproducible imputations using the `seed()` option. The default *#* is 0, meaning that no seed is set by the program.

4.3 Options for micombine

`noconstant` suppresses the regression constant in all regressions.

`detail` gives details of the regression model for each imputation.

`eform(`*string*`)` specifies that the exponentiated form of the coefficients be output and that the constant not be reported; *string* is used to label the exponentiated coefficients.

`genxb(`*newvar*`)` creates *newvar* to hold the linear predictor from each regression model, averaged over all the imputations.

`impid(`*varname*`)` specifies that *varname* is the variable identifying the imputations. The number of imputations is determined as the number of unique values of *varname*. The default *varname* is `_j`.

`lrr` specifies that the Li–Raghunathan–Rubin (LRR) robust estimate of the variance–covariance matrix of the regression coefficients be used.

regression_cmd_options may be any of the options appropriate to *regression_cmd*.

4.4 Options for mijoin and misplit

`clear` is not optional and confirms that you are willing to replace the data in memory.

`impid(`*varname*`)` specifies that *varname* is the variable identifying the imputations. The number of imputations is determined as the number of unique values of *varname*. The default *varname* is `_j`.

`m(`*#*`)` sets the number of imputed datasets to *#*.

5 Remarks

5.1 Univariate imputation

`uvis` (univariate imputation sampling) imputes missing values of the variable *yvar* from complete cases in *xvarlist*. The algorithm used is exactly as described by van Buuren, Boshuizen, and Knook (1999, section 3.2). The use of prediction matching ensures that values are imputed only within the observed distribution of *yvar*.

With the `boot` option of `uvis`, the parameter $\widehat{\beta}_*$ is estimated by regression of *yvar* on *xvarlist* within a bootstrap sample. The bootstrap method has the advantage of robustness since it is not necessary to assume that $\widehat{\beta}$ is normally distributed.

Some comments are required about prediction matching when *yvar* is a categoric variable. When *yvar* is binary and logistic regression is used, no issue arises. Prediction matching may be done on the logistic or the probability scale, almost always with identical results (`uvis` uses the logistic scale). With multicategory predictors, either ordinal (`ologit`) or multinomial (`mlogit`) logistic regression may be used to model *yvar*, as appropriate. In such cases, for a given missing value of *yvar*, one finds the observed value of *yvar* that minimizes the mean absolute difference between the logits of the predicted class probabilities. Specifically, let $\widehat{\pi}_{\mathrm{obs};i;j}$ and $\widehat{\pi}_{\mathrm{mis};j}$ denote the predicted probability that the ith nonmissing observation and the target missing observation of *yvar*, respectively, will fall into class j ($j = 1, \ldots, K$). These predictions are obtained from an `ologit` or `mlogit` fit of *yvar* on *xvarlist*. The imputation for the target observation is observation number

$$\arg\min_i \sum_{j=1}^{K} \left|\mathrm{logit}\left(\widehat{\pi}_{\mathrm{obs};i;j}\right) - \mathrm{logit}\left(\widehat{\pi}_{\mathrm{mis};j}\right)\right|$$

among the nonmissing values of *yvar*.

5.2 Multivariate imputation

`mvis` carries out multivariate imputation on *mainvarlist* by using "regression switching" (van Buuren, Boshuizen, and Knook [1999]). The algorithm is a type of Gibbs sampler in which the distribution of missing values of a covariate is sampled conditional on the distribution of the remaining covariates. Each variable in *mainvarlist* becomes in turn the response variable; `uvis` is used to impute its missing values univariately on the remaining variables. Let the variables in *mainvarlist* be $x_1, x_2, \ldots, x_k$. The procedure is as follows:

1. Ignore observations for which every member of $x_1, x_2, \ldots, x_k$ or any member of *ccvarlist* (if there are any) has a missing value.
2. For each variable with any missing data in $x_1, x_2, \ldots, x_k$, randomly order that variable and replicate its observed values across the missing cases. This step initializes the iterative procedure by filling in missing data at random.
3. For each of $x_1, x_2, \ldots, x_k$, in turn, impute missing values by applying `uvis` with the remaining variables as covariates.
4. Repeat step 3 # times, where # is specified by the `cycles(#)` option. At each cycle, replace previous imputations with updated ones obtained from the latest application of `uvis`. This creates a single imputation sample.
5. To obtain m imputations, repeat the procedure m times independently. The number of imputations is controlled by the `m(#)` option of `mvis`.

At step 4, van Buuren, Boshuizen, and Knook (1999) recommend 20 cycles but go on to say that 10 or even 5 are probably sufficient. I chose to use a default of 10.

6 Example

We will again work with the breast cancer dataset which was analyzed in detail by Sauerbrei and Royston (1999). The data are provided in `brcancer.dta` and relate to a set of 686 patients with node-positive breast cancer. The outcome of interest is the recurrence-free survival time (RFS), that is, the duration in years from entry into the study (typically, the time of diagnosis of primary breast cancer) until either death or disease recurrence, whichever occurred first. There were 299 events for this outcome and the median follow-up time was about 5 years.

Sauerbrei and Royston (1999) derived a Cox proportional-hazards model for RFS that included five covariates: age (`x1`) with a fractional polynomial transformation with powers -2 and -0.5, tumor grade 2/3 (`x4a`), number of positive lymph nodes (`x5`) with the exponential transformation `x5e` $= \exp(-0.12 *$ `x5`$)$, progesterone receptors (`x6`) with a fractional polynomial transformation with power 0.5, and hormonal therapy with tamoxifen (`hormon`). This model may be fit in Stata as follows:

(*Continued on next page*)

```
. webuse brcancer
(German breast cancer data)
. stset rectime, fail(censrec)
     failure event:  censrec != 0 & censrec < .
obs. time interval:  (0, rectime]
 exit on or before:  failure
------------------------------------------------------------------------------
      686  total obs.
        0  exclusions
------------------------------------------------------------------------------
      686  obs. remaining, representing
      299  failures in single record/single failure data
   771400  total analysis time at risk, at risk from t =         0
                             earliest observed entry t =         0
                                  last observed exit t =      2659
. fracgen x1 -2 -0.5
-> gen double x1_1 = X^-2
-> gen double x1_2 = X^-0.5
   (where: X = x1/10)
. fracgen x6 0.5
-> gen double x6_1 = X^0.5
   (where: X = (x6+1)/1000)
. stcox x1_1 x1_2 x4a x5e x6_1 hormon, nohr
         failure _d:  censrec
   analysis time _t:  rectime
Iteration 0:   log likelihood = -1788.1731
(Iteration log omitted)
Cox regression -- Breslow method for ties
No. of subjects =          686                     Number of obs   =       686
No. of failures =          299
Time at risk    =       771400
                                                   LR chi2(6)      =    153.11
Log likelihood  =   -1711.6186                     Prob > chi2     =    0.0000
------------------------------------------------------------------------------
          _t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        x1_1 |   43.55382   8.253433     5.28   0.000     27.37738    59.73025
        x1_2 |  -17.48136   3.911882    -4.47   0.000    -25.14851   -9.814212
         x4a |   .5174351   .2493739     2.07   0.038     .0286713    1.006199
         x5e |  -1.981213   .2268903    -8.73   0.000    -2.425909   -1.536516
        x6_1 |   -1.84008   .3508432    -5.24   0.000     -2.52772    -1.15244
      hormon |  -.3944998    .128097    -3.08   0.002    -.6455654   -.1434342
------------------------------------------------------------------------------
```

Note the use of `fracgen` to transform the covariates and `stcox` to fit the Cox model. (In fact, this model could have been fitted in one step by `fracpoly`, but such usage does not generalize to multiple imputation with `micombine`.)

To illustrate multiple imputation, I created from `brcancer.dta` a new dataset `brcaex.dta` in which approximately 20% of the observations on the five covariates `x1`, `x4a`, `x5e`, `x6`, and `hormon` were replaced completely at random with missing values, creating new covariates called `mx1`, `mx4a`, `mx5e`, `mx6`, and `mhormon`. Five imputations of the missing values were created as follows:

```
. use brcaex, clear
(German breast cancer data)
. mvis mx1 mx4a mx5e mx6 mhormon lnt _d using brcaeximp, m(5) genmiss(m_) seed(
> 101)
imputing 1..2..3..4..5..file brcaeximp.dta saved
```

The `mvis` command saves the imputations to a new file, `brcaeximp.dta`, with the original variable names `mx1`, `mx4a`, `mx5e`, `mx6`, and `mhormon` intact and with all the missing values replaced by imputations. This format is suitable for use by `micombine`. As recommended by van Buuren, Boshuizen, and Knook (1999), the (log) survival time `lnt` and the censoring indicator `_d` are included in the imputation model.

Here we use `micombine` to fit Sauerbrei and Royston's model and obtain appropriate parameter estimates and standard errors. First, the new dataset is loaded and the requisite fractional polynomial transformations are applied to `mx1` and `mx6`:

```
. use brcaeximp, clear
(German breast cancer data)
. fracgen mx1 -2 -0.5
-> gen double mx1_1 = X^-2
-> gen double mx1_2 = X^-0.5
   (where: X = mx1/10)
. fracgen mx6 0.5
-> gen double mx6_1 = X^0.5
   (where: X = (mx6+1)/1000)
```

Finally, the model is fit on each imputation, and combined estimates are obtained:

```
. micombine stcox mx1_1 mx1_2 mx4a mx5e mx6_1 mhormon
Multiple imputation parameter estimates (5 imputations)
------------------------------------------------------------------------------
          _t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       mx1_1 |   42.35447   14.14086     3.00   0.003     14.63889    70.07004
       mx1_2 |  -16.83557   6.331493    -2.66   0.008    -29.24507   -4.426075
        mx4a |    .695139   .2944367     2.36   0.018     .1180537    1.272224
        mx5e |  -1.862887   .2759056    -6.75   0.000    -2.403652   -1.322122
       mx6_1 |   -1.81862   .4307429    -4.22   0.000    -2.662861   -.9743798
     mhormon |  -.3781116   .1681751    -2.25   0.025    -.7077288   -.0484944
------------------------------------------------------------------------------
686 observations.
```

The results may be compared with those from the original data presented above. The parameter estimates are similar, but since some information has been lost, the standard errors are larger with the imputed data.

6.1 A note on prediction

For a given observation, the multiple-imputation estimate of the linear predictor or index (`xb`) is the average of the estimated linear predictors over the m imputations. This may be computed by specifying the `genxb(`*newvar*`)` option of `micombine`. Out-of-sample predictions from the combined model in each imputation may be found by using the `predict` command immediately after using `micombine`. Predictions from fitting the model separately in each imputation require the use of the `forvalues` command with `if` filtration by the values of the imputation indicator variable (`_j`). The model is fitted in each imputation and `predict` is then used for prediction.

6.2 A note on the imputation model

Inspection of the imputations for the breast cancer example shows that the imputed observations are only weakly correlated with the original ones. Consider the strongest factor, the number of positive lymph nodes (`x5`). The Spearman rank correlations for the five imputations between the original and imputed values of `x5`, computed at the 20% of randomly deleted values are only 0.16, 0.15, 0.09, 0.05, and 0.15. As an experiment, five additional imputations of each prognostic factor were made independently by applying `uvis` with single, random, uniformly distributed variables as predictors. The deviance, or minus twice the maximized partial log likelihood from each dataset, averaged over the five imputations, was 3470.8, compared with 3437.5 for the imputations based on the prognostic factors and survival time, and 3423.2 for the model fitted to the data before deletions. To compute the deviances, the linear predictor from the combined model was calculated in each dataset, and the partial likelihood was calculated for a Cox model with no covariates and with the linear predictor offset. The loss of predictive accuracy (measured by the deviance) was 14.3 for the prognostic factor-based imputations, compared with 47.6 for the random imputations. This simple experiment demonstrates the importance of preserving the multivariate structure of the original predictors in the imputations, even when the predictive power of the imputation model is not large. Use of completely random imputations is likely to give suboptimal results.

7 Choice of m

There is little discussion in the literature as to how large the number m of imputations should be. Quoting Rubin (1987), van Buuren, Boshuizen, and Knook (1999) say that "Simulation studies have shown that ... m can be as low as three for data with 20 percent of missing entries. In the following I use $m = 5$, which is a conservative choice". The basis of this statement seems to concern the precision of the vector $\widehat{\beta}$ of regression coefficients. However, the variance–covariance matrix of $\widehat{\beta}$ is also important; in particular, sufficiently accurate confidence intervals for $\widehat{\beta}$ are desirable. Based on such considerations, I shall suggest an approach to determining m. This should be seen as an initial proposal, which I hope will stimulate further research on the topic.

7.1 Rubin's rules

First, I need to state Rubin's rules for estimating a scalar quantity Q based on values from m complete-data imputations. For example, Q could be one of the components of β. Let Q_j and W_j denote the point estimate and variance respectively from the jth ($j = 1, \ldots, m$) complete dataset. The multiple-imputation point estimate Q^* of Q is the arithmetic mean of the m complete-data estimates. The estimated variance T of Q is obtained by a components-of-variance argument, leading to the following formulas

$$T = W + \left(1 + \frac{1}{m}\right) B$$

where

$$W = \frac{1}{m} \sum_{j=1}^{m} W_j$$

$$B = \frac{1}{m-1} \sum_{j=1}^{m} (Q_j - Q^*)^2$$

are the within- and between-imputation components of variance, respectively. For confidence intervals, Rubin (1987) gives the approximation

$$Q^* \pm t_\nu \sqrt{T}$$

where the degrees of freedom ν are estimated by

$$\nu = (m-1) \left\{ 1 + \frac{W}{\left(1 + \frac{1}{m}\right) B} \right\}^2$$

and where t_ν is the appropriate fractile of the central t distribution on ν degree of freedom. Note that both ν and T are estimated from the data and that both depend on the quantity B. Note also that ν depends on $(W/B)^2$, a quantity that may have a large variance and a highly skew distribution. B itself is an estimated variance with $m - 1$ degrees of freedom.

7.2 Unreliable confidence intervals?

I explored the potential instability of the confidence coefficient $t_\nu \sqrt{T}$ within the breast cancer data. The confidence level chosen was 95%, so for example, for $\nu = 15.8$ (note that ν may be fractional here), $t_\nu = 2.122$. Using `mvis` as described above, I created a large dataset comprising $M = 450$ complete-data imputations from the artificially censored dataset `brcaex.dta`. For each of $m = 3$, 5, 10, 20, and 50, I computed the mean and 95% confidence interval for $t_\nu \sqrt{T}$. This was done by breaking the M imputed datasets into blocks of M/m datasets, applying `micombine` to each block of m imputations, and finally, summarizing the resulting replicated values of $t_\nu \sqrt{T}$. For example, with $m = 3$, there were $450/3 = 150$ such blocks; therefore, the summary statistics are based on 150 observations. Figure 1 shows for each covariate, numbered from 1 to 6, how $t_\nu \sqrt{T}$, shown as a 95% empirical confidence interval, depends on m.

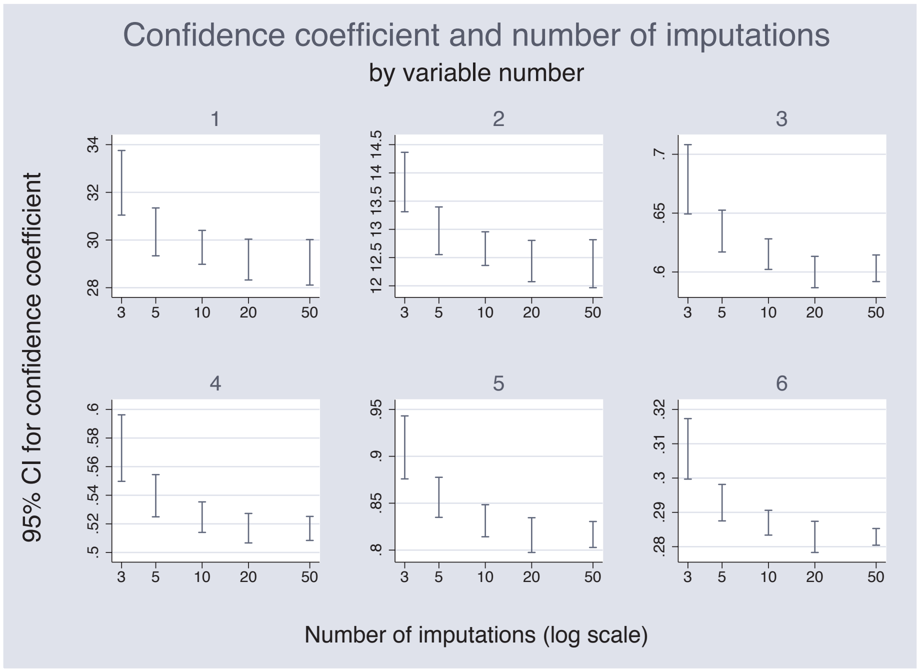

Figure 1: Breast cancer data: dependence of the 95% confidence interval for the confidence coefficient, $t_\nu\sqrt{T}$ on the number of imputations, m.

For each of the six covariates, there is a similar pattern of dependence on m. The gradual reduction of the confidence coefficient with increasing m occurs because both T and t_ν are inversely related to m. Confidence intervals for Q are correspondingly wider for lower m.

Figure 2 shows the coefficient of variation (CV = 100× standard deviation divided by the mean) of the confidence coefficient, in a format similar to figure 1.

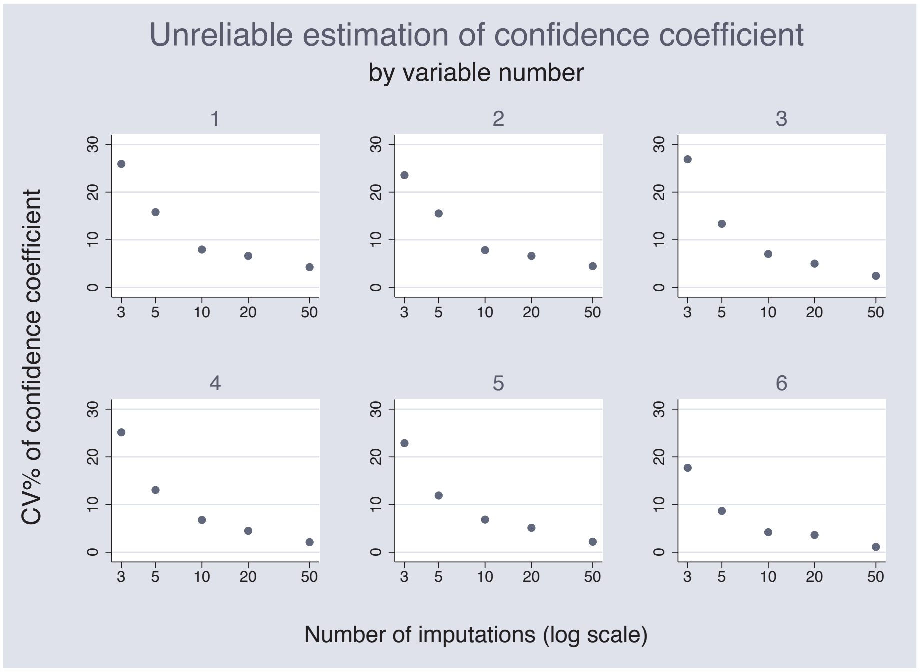

Figure 2: Breast cancer data: Coefficient of variation (100 × S.D. divided by mean) of the confidence coefficient.

The graph illustrates that the estimated confidence coefficient may be very variable for low m, and this translates into unreliable confidence intervals for Q. For example, with $m = 5$, a favorite literature choice, the CV is in the region of 13% in this example. The CV may be doubled to indicate roughly the range of uncertainty in confidence intervals for Q, in this example giving the unacceptably large figure of ±26%.

7.3 A rule of thumb for selecting m

Based on the foregoing analysis, I propose the following rule of thumb: choose m to be large enough such that the CV of the confidence coefficient for the worst-case parameter is $< 5\%$. An approximately equivalent criterion would be to require the standard deviation of ln(confidence coefficient) to be < 0.05. This is somewhat more convenient to compute. The worst-case parameter will typically attach to the variable with the greatest proportion of missing data. The rule implies that the range of uncertainty in confidence intervals for Q will be roughly $< 10\%$. In the example discussed above, the rule would require m to be at least 20 and possibly more (see figure 2).

To apply the rule, the CV should be evaluated for all parameters for which accurate estimates and confidence intervals are of central interest. This might exclude, for example, confounders (adjustment variables) in the analysis of epidemiological data.

Parameters for confounders are not normally of interest. The CV may be evaluated in the way I have indicated, by creating replicate sets of multiple imputations, using `micombine` on each and summarizing the results by standard methods. Pointers to a high CV for a given variable are a large fraction of missing data or a large value (say, > 1.2) of the between-imputation inflation factor or BIF, $\sqrt{T/W}$. The latter quantity reflects the inflation in the standard error of a parameter due to variation between imputations. The quantities B, W, T, ν, and BIF are returned by `micombine` in the matrices `e(B)`, `e(W)`, `e(V)`, `e(nu)`, and `e(BIF)`, respectively. Specifically, `e(B)`, `e(W)`, and `e(V)` are covariance matrices for the full parameter vector β, whereas `e(nu)` and `e(BIF)` are column vectors with an entry for each element of β.

An ado-file called `postmi.ado` to facilitate these calculations is under development and will be published separately. A beta version can be obtained from the author (patrick.royston@ctu.mrc.ac.uk).

8 Further comments

Sometimes it is necessary to investigate possible models, for example, prognostic models, in which selection of influential variables is required and there are missing data. See Clark and Altman (2003) for an interesting example. For example, the stability of the final model across the imputation samples is of interest.

In survival analysis, it is recommended that the log of the survival time and the censoring indicator be used as predictors in the imputation model. van Buuren, Boshuizen, and Knook (1999) give a detailed discussion of the different types of covariate that can be included in the imputation model and suggest an approach to dealing with variables that are missing not at random (MNAR).

In the present implementation of multivariate imputation sampling in `mvis`, all the variables in *varlist* are used for imputation of all the others. This restriction could in principle be lifted, but it is not clear whether the additional complexity would improve the results much.

See also van Buuren's web site *http://www.multiple-imputation.com* for further information and other software sources.

Finally, a note of caution. If the MAR assumption is valid, the methods implemented here will give correct results—but the assumption is hard to check. Common sense suggests that, as the proportion of missing observations on a given variable increases, the reliability of estimates (e.g., regression coefficients) relating to that variable will diminish. It is expected that the variance of estimates will increase, but if the MAR assumption breaks down, their (unknown) bias will also increase. Although no reliable rule of thumb is available, in my practice I am not comfortable with imputing missing values for a variable in which the proportion of missing observations exceeds approximately 50%.

9 Acknowledgment

I thank Ian White for helpful comments on the manuscript and software design and for assistance with some aspects of the programming.

10 References

Carlin, J. B., N. Li, P. Greenwood, and C. Coffey. 2003. Tools for analyzing multiple imputed datasets. *Stata Journal* 3(3): 226–244.

Clark, T. G. and D. G. Altman. 2003. Developing a prognostic model in the presence of missing data: an ovarian cancer case study. *Journal of Clinical Epidemiology* 56: 28–37.

Mander, A. and D. Clayton. 1999. sg116: Hotdeck imputation. *Stata Technical Bulletin* 51: 32–34. In *Stata Technical Bulletin Reprints*, vol. 9, 196–199. College Station, TX: Stata Press.

Rubin, D. B. 1976. Inference and missing data (with discussion). *Biometrika* 63: 581–592.

—. 1987. *Multiple imputation for non-response in surveys*. New York: John Wiley & Sons.

Sauerbrei, W. and P. Royston. 1999. Building multivariable prognostic and diagnostic models: transformation of the predictors using fractional polynomials. *Journal of the Royal Statistical Society,* Series A 162: 71–94.

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 25 years of experience, with a strong interest in biostatistical methodology and in statistical computing and algorithms. At present, he works in clinical trials and related research issues in cancer. Currently, he is focusing on problems of model building and validation with survival data, including prognostic factors studies; on parametric modeling of survival data; and on novel trial designs.

The Stata Journal (2005)
5, Number 2, pp. 188–201

Multiple imputation of missing values: update

Patrick Royston
Cancer Division
MRC Clinical Trials Unit
222 Euston Road
London NW1 2DA UK

Abstract. This article describes a substantial update to `mvis`, which brings it more closely in line with the feature set of S. van Buuren and C. G. M. Oudshoorn's implementation of the MICE system in R and S-PLUS (for details, see *http://www.multiple-imputation.com*). To make a clear distinction from `mvis`, the principal program of the new Stata release is called `ice`. I will give details of how to use the new features and a practical illustrative example using real data. All the facilities of `mvis` are retained by `ice`. Some improvements to `micombine` for computing estimates from multiply imputed datasets are also described.

Keywords: st0067_1, ice, mvis, uvis, micombine, mijoin, misplit, missing data, missing at random, multiple imputation, multivariate imputation, regression modeling

1 Introduction

Royston (2004) introduced `mvis`, an implementation for Stata of a method of multiple multivariate imputation of missing values under missing-at-random (MAR) assumptions. The method is known as MICE, an acronym for multiple imputation by chained equations (van Buuren et al. 1999). See van Buuren's article for details of the theory behind MICE and his interesting web site *http://www.multiple-imputation.com* for abundant literature references, reports, information, and software links. The material will not be repeated here.

Royston (2004) presented five ado-files: `mvis` to create multiple multivariate imputations; `uvis` to impute missing values of a single variable as a function of several covariates, each with complete data; `micombine` to fit a wide variety of regression models to a multiply imputed dataset, combining the estimates using Rubin's rules (1987); and `misplit` and `mijoin`, utilities to inter-convert datasets created by `mvis` and by Carlin et al.'s `miset` routine (2003).

In this article, I will describe a substantial update to `mvis`, which brings it more closely in line with the feature set of S. van Buuren and C. G. M. Oudshoorn's implementation of MICE in R and S-PLUS (for details, see *http://www.multiple-imputation.com*). To avoid confusion with `mvis`, the principal program of the new Stata release is called `ice`. I will give details of how to use the new features and a practical illustrative example using real data. All the facilities of `mvis` are retained by `ice`. Some improvements to `micombine` are also described.

2 Syntax

`ice` *mainvarlist* `using` *filename*[`.dta`] [`if` *exp*] [`in` *range*] [*weight*] [`,`
`boot`[`(`*varlist*`)`] `cc(`*ccvarlist*`)` `cmd(`*cmdlist*`)` `cycles(`#`)` `draw`[`(`*varlist*`)`]
`dryrun` `eq(`*eqlist*`)` `genmiss(`*string*`)` `id(`*string*`)` `m(`#`)` `on(`*varlist*`)` `noconstant`
`noshoweq` `passive(`*passivelist*`)` `substitute(`*sublist*`)` `replace` `seed(`#`)`]

`uvis` *regression_cmd* *yvar* *xvarlist* [`if` *exp*] [`in` *range*] [*weight*]`,`
`gen(`*newvarname*`)` [`noconstant` `boot` `draw` `replace` `seed(`#`)`]

where *regression_cmd* may be `logistic`, `logit`, `mlogit`, `ologit`, or `regress`. All weight types supported by *regression_cmd* are allowed.

`micombine` *regression_cmd* [*yvar*] [*covarlist*] [`if` *exp*] [`in` *range*] [*weight*] [`,`
`br` `noconstant` `detail` `eform(`*string*`)` `genxb(`*newvarname*`)` `impid(`*varname*`)`
`lrr` `obsid(`*varname*`)` *regression_cmd_options*]

where *regression_cmd* may be `clogit`, `cnreg`, `glm`, `logistic`, `logit`, `poisson`, `probit`, `qreg`, `regress`, `rreg`, `xtgee`, `streg`, `stcox`, `ologit`, `oprobit`, or `mlogit`. All weight types supported by *regression_cmd* are allowed.

`mijoin` [*filestubname*]`, clear` [`m(`#`)` `impid(`*varname*`)`]

`misplit, clear` [`m(`#`)` `impid(`*varname*`)`]

The options are described in the help files. Features new to `mvis` and now in `ice` are discussed in some detail in the next section. New features of `micombine` are discussed in the section *Changes to micombine*.

3 Options

I shall give details here only of options that are new or modified from the previous release of the suite of programs.

3.1 New options for ice

`dryrun` does a "dry run"; that is, `ice` reports the prediction equations it has constructed from the various inputs. No imputation is done, and no files are created. It is not mandatory to specify an output file with `using` for a dry run. The prediction equation setup needs to be carefully checked before running what may be a lengthy imputation process.

`eq(`*eqlist*`)` allows customization of prediction equations for any subset of variables in *mainvarlist*. The `eq()` option, particularly when used with `passive()`, allows great flexibility in the possible imputation schemes. The syntax of *eqlist* is *varname1*`:` *varlist1* [`,` *varname2*`:` *varlist2* ...] where each *varname#* (or *varlist#*) is a member or subset of *mainvarlist*. It is your responsibility to ensure that each equation is sensible. `ice` places no restrictions except to check that all variables mentioned are indeed in *mainvarlist*, and that an equation is not defined for a variable specified to be passively imputed (see the `passive()` option). Note that `eq()` takes precedence over all default definitions and assumptions about the way a given variable in *mainvarlist* will be imputed. The default, if the `passive()` and `substitute()` options are not invoked, is that each variable in *mainvarlist* with any missing data is imputed from all the other variables in *mainvarlist*.

`noshoweq` suppresses the presentation of the prediction equations.

`passive(`*passivelist*`)` allows the use of "passive" imputation of variables that depend on other variables, some of which are imputed. The syntax of *passivelist* is *varname*`:` *exp* [\varname`:` *exp* ...]. Here *exp* denotes a valid Stata expression (see `help exp` in Stata). Notice the requirement to use '\' as a separator between items in *passivelist*, rather than the usual comma; the reason is that a comma may be a valid part of an expression. The option is most easily explained by example. Suppose that `x1` is a categorical variable with 3 levels, and that two dummy variables `x1a` and `x1b` have been created by the commands

```
. generate byte x1a=(x1==2)
. generate byte x1b=(x1==3)
```

Now suppose that `x1` is to be imputed by the `mlogit` command and is to be treated as the two dummy variables `x1a` and `x1b` when predicting other variables. Use of `mlogit` is achieved by the option `cmd(x1:mlogit)`. When `x1` is imputed, we want `x1a` and `x1b` to be updated with new values that depend on the imputed values of `x1`. This may be achieved by specifying `passive(x1a:x1==2\x1b:x1==3)`. It is necessary also to remove `x1` from the list of predictors when variables other than `x1` are being imputed, and this is done by using the `substitute()` option; in the present example, you would specify `substitute(x1:x1a x1b)`. In this example, the `generate` statements given above will make `x1a` take the (possibly unintended) value of 0 when `x1` is missing. However, `ice` is careful to ensure that `x1a` and `x1b` inherit the missingness of `x1` and are passively imputed following active imputation of missing values of `x1`. If this were not done, incorrect results could occur. The responsibility of the user is to create `x1a` and `x1b` before running `ice` such that their missing values are identical to those of `x1`. The values they assume initially are irrelevant since `ice` recalculates them anyway.

A second example is multiplicative interactions between variables, say, between `x1` and `x2` (e.g., `x12=x1*x2`); this could be implemented through `passive(x12:x1*x2)`.

It would cause the interaction term `x12` to be omitted when either `x1` or `x2` was being imputed since it would make no sense to impute `x1` from its interaction with `x2`. The `substitute()` option is not needed here. It should be stressed that variables to be imputed passively must be included in *mainvarlist*; otherwise, they will not be recognized.

`substitute(`*sublist*`)` is typically used with the `passive()` option to represent multilevel categorical variables as dummy variables in models for predicting other variables. See `passive()` for more details. The syntax of *sublist* is *varname*`:`*dummyvarlist* [`,`*varname*`:`*dummyvarlist* ...], where *varname* is the name of a variable to be substituted and *dummyvarlist* is the list of dummy variables representing it.

3.2 Options for uvis, mijoin, misplit, micombine

None of the options for `uvis`, `mijoin`, `misplit` have been changed. Details of some changes to `micombine` are given in the section *Changes to micombine*.

4 What is new?

The changes to `mvis` implemented in `ice` mainly affect two areas:

1. The flexibility of the prediction equations used in the system of chained equations and
2. The way the program handles categorical variables, interactions, and transformations.

4.1 Flexible prediction equations

The earlier program `mvis` is restricted to imputing missing observations for each variable in *mainvarlist* from all the other members of *mainvarlist*. While this is sensible in many applications, there are many other cases in which greater flexibility is needed. I will give a relatively simple example in the section *Example: Fetal size*. With the `eq()` option of `ice`, an equation (i.e., a list of right-hand-side variables, comprising part of a model) may be specified in order to impute any variable in *mainvarlist* from any subset of *mainvarlist*. The `eq()` option overrides all automatic behavior invoked by the `passive()` and `substitute()` options (described in context below), giving the user ultimate control over the prediction equations.

4.2 Categorical variables

`mvis` does not handle categorical variables in an ideal fashion. For example, suppose that `x1` is continuous and `x2` is a three-level unordered categorical variable taking the values 0, 1, 2, both variables having missing data. It would be preferable to impute `x1` by linear regression on dummy variables `x21` and `x22` indicating values of 1 and 2, respectively, of `x2`. A natural way to impute `x2` is by multinomial logistic regression (`mlogit` command) of `x2` on `x1`. A possible way to achieve this with `mvis` may be thought to be

```
. generate byte x21=(x2==1) if x2<.
. generate byte x22=(x2==2) if x2<.
. mvis x1 x2 using temp, m(5) cmd(x2:mlogit)
```

However, although appropriate for predicting `x2`, this approach predicts `x1` from `x2` but not from the dummy variables `x21` and `x22`, which are not used. An alternative scheme with dummy variables for `x2`,

```
. mvis x1 x21 x22 using temp, m(5) cmd(x21 x22:logit)
```

does not allow constrained imputation of `x2` (as would be achieved by using `mlogit`). The result could be inconsistent estimates of the missing values of `x2` when the latter is reconstucted from the original and imputed values of `x21` and `x22`. For example, suppose that `x2` = 2 in a particular observation. Then `x21` = 0 and `x22` = 1. Suppose that `x2` were missing for this observation and that `x21` and `x22` were imputed independently, as in the command above. It could happen by chance that `x21` = 1 and `x22` = 1, which do not encode a possible value of `x2`. An appropriate solution with `ice` is

```
. ice x1 x2 x12 x22 using temp, m(5) cmd(x2:mlogit)
    passive(x21:x2==1\x22:x2==2) substitute(x2:x21 x22)
```

The logic here is worth spelling out. *mainvarlist* must include all variables that are involved in the imputation; hence `x2` and both its dummy variables are listed. However, we do not wish to predict `x1` from `x2 x12 x22`, but only from `x12 x22`. This is achieved by the option `substitute(x2:x21 x22)`, which replaces `x2` with `x21 x22` whenever `x2` is a predictor and removes redundant mentions of variables already included. Finally, it is necessary to construct imputed values of `x21` and `x22` from imputed values of `x2`, the latter being obtained internally via the model `mlogit x2 x1`. This is done by the option `passive(x21:x2==1\x22:x2==2)`, which recalculates the dummy variables from the newly imputed values of `x2`.

Although it may at first sight appear complicated, the approach is consistent. The options `passive()`, `substitute()`, and `eq()` together cover all practical possibilities with categorical variables in a unified syntax. More generally, passive imputation (a term coined, I believe, by Steff van Buuren) is an important feature of the MICE system for dealing flexibly with categorical covariates, interactions, transformations and the like. The `substitute()` option is mainly a convenience feature, since substitution could be achieved explicitly by use of the `eq()` option, for example, `eq(x1:x21 x22)` instead of `substitute(x2:x21 x22)`.

4.3 Interactions

Correct imputation involving multiplicative interactions requires the `passive()` option. Suppose that we plan to impute continuous variables `x1` and `x2` and a binary variable

z. Suppose also that there are good reasons for imputing x2 from the main effects and interaction of x1 with z, that is from x1, z, and a new variable x1z = x1*z. How do we impute x1 and z in this situation? If we were to specify

```
. generate x1z=x1*z
. ice x1 x2 z x1z using temp, m(5)
```

we would get x2 imputed from x1, z, and x1z (as required); x1 from x2, z, and x1z; and z from x1, x2, and x1z. However, it makes no sense to impute a variable from its interaction with another variable. We wish to eliminate the interaction term x1z from the prediction equations for x1 and z. Even if the problem were tackled by using the eq() option, because of the danger of inconsistency we do not want the imputation of x1z to be treated independently of the imputation of its components, x1 and z. The way to solve the problem is to use the passive() option. Note also the dryrun option that allows us to see and check the prediction equations but not yet to run the imputations:

```
. ice x1 x2 z x1z using temp, m(5) passive(x1z:x1*z) dryrun
    Variable | Command      | Prediction equation
-------------+--------------+------------------------------------------------
          x1 |              | [No missing data in estimation sample]
          x2 | regress      | x1 z x1z
           z | mlogit       | x1 x2
         x1z |              | [Passively imputed from x1*z]
End of dry run. No imputations were done, no files were created.
```

As seen above, ice automatically removes the composite passive variable x1z from the relevant prediction equations. When it discerns that x1z is computed from x1 and z, it *knows* to exclude x1z from the prediction equations for imputing x1 and z. This would happen with any variable that is a combination of others.

4.4 Transformations

Simple or complex transformations may be handled by combining the passive() and eq() options. Here is a fairly complicated but realistic example, which prefigures the example presented in detail in the section *Example: Fetal size*. Suppose that we have three continuous variables y1, y2, and x, all with missing values. Preliminary analysis indicates that the following prediction equations are appropriate:

$$E(y_1) = \beta_1 x^{-2} + \beta_2 x^{-1}$$
$$E(y_2) = \gamma_1 x + \gamma_2 x^2$$
$$E(x) = \delta_1 y_1 + \delta_2 y_2$$

In other words, in expectation, y1 is a fractional polynomial function of x with powers $(-2, -1)$, y2 is a quadratic function of x, and x is a linear function of y1 and y2. You may imagine y1 and y2 to be two response variables linked to a common covariate, x. With this imputation scheme, missing values may be imputed in ice as follows:

```
. generate xa=x^-2
. generate xb=x^-1
. generate xc=x^2
. ice y1 y2 x xa xb xc using temp, m(5) passive(xa:x^-2\xb:x^-1\xc:x^2)
> eq(y1:xa xb, y2:x xc, x:y1 y2)
    Variable | Command       | Prediction equation
------------+---------------+------------------------------------------------
          y1 | regress       | xa xb
          y2 | regress       | x xc
           x | regress       | y1 y2
          xa |               | [Passively imputed from x^-2]
          xb |               | [Passively imputed from x^-1]
          xc |               | [Passively imputed from x^2]
```

Here the equation for each variable must be defined explicitly, and this is easily done by using the `eq()` option.

4.5 More on passively imputed variables

Passive (passively imputed) variables in general are computed directly as a function of one or more active (actively imputed) variables. They inherit their missing values from their component active variables. An active variable is a member of *mainvarlist* that is not defined as passive and that has at least one missing value in the estimation sample. Finally, there is a third class of fully observed variables that have no missing values within the estimation sample defined by `if`, `in`, and weights.

It is sometimes convenient to create passive variables as a function of other passive (and perhaps also of active and fully observed) variables. This will work fine provided that the chain of computation of passive variables is correctly reflected in the order of the variables presented in *mainvarlist*. For example, the command

```
. ice y x z xa xaz using temp, m(5) passive(xa:x^-2\xaz:xa*z)
    eq(y:xa z xaz, x:y z, z:y xa)
```

will compute `xa=x^-2` followed by `xaz=xa*z`, which is appropriate since `x` and `z` precede `xa` and `xa` precedes `xaz` in *mainvarlist*. On the other hand, on reversing `xa` and `xaz` in *mainvarlist*, namely

```
. ice y x z xaz xa using temp, m(5) passive(xa:x^-2\xaz:xa*z)
    eq(y:xa z xaz, x:y z, z:y xa)
```

will cause `ice` to recalculate `xaz` before `xa` has been updated from `x`. This will give incorrect results. The rule is simple: structure *mainvarlist* such that primary passive variables (that is, variables that are derived only from active variables) precede secondary passive variables (which depend on primary passive variables) and the latter precede tertiary passive variables (which involve secondary passive variables), and so on. The order of definition of passive variables in the `passive()` option is immaterial.

Note that secondary and higher-order passive variables are not traced back to their original active variables and are therefore not automatically removed from the prediction equations of the relevant active variables. Consider the example

```
. ice x1 x2 z z1 z2 x2z1 x2z2 using temp, replace m(5) substitute(z:z1 z2)
> passive(z1:z==1\z2:z==2\x2z1:x2*z1\x2z2:x2*z2)
   Variable | Command      | Prediction equation
------------+--------------+-----------------------------------------------
         x1 | regress      | x2 z1 z2 x2z1 x2z2
         x2 | regress      | x1 z1 z2
          z | mlogit       | x1 x2 x2z1 x2z2
         z1 |              | [Passively imputed from z==1]
         z2 |              | [Passively imputed from z==2]
       x2z1 |              | [Passively imputed from x2*z1]
       x2z2 |              | [Passively imputed from x2*z2]
```

We see that `z` is inappropriately going to be predicted from `x1`, `x2`, and `z`'s interaction with `x2`, expressed indirectly through the interaction between `x2` and `z1`, `z2`. Here, `x2z1` and `x2z2` are defined as secondary passive variables. To correct the problem, you could either specify the equation for `z` directly, e.g., `eq(z:x1 x2)` or define only primary passive variables, as in `passive(z1:z==1\z2:z==2\x2z1:x2*(z==1)\x2z2:x2*(z==2))`. The second solution is perhaps cleaner, though more laborious.

5 Example: Fetal size

5.1 Data

I will present an example that is rather different from the usual sort of imputation problem encountered in clinical biostatistics and epidemiology. Some 15 years ago, Altman and Chitty (1993) designed and performed an influential study of the growth of the fetus. The aim was to establish gestational age-specific reference intervals ("normal ranges") for each of a large number of in-utero "parameters" (anthropometric measurements) of fetal size. In each of 649 singleton pregnancies, ultrasound scanning of the abdomen was carried out once on each mother according to a predefined study schedule. The intention was adequately to cover the important gestational period between 12 and 42 weeks, and this was largely achieved. The dataset is therefore cross-sectional in character. A longitudinal study of growth was also performed, but I will not consider that here.

In this example, I will consider a subset of the study dataset comprising just 4 variables: gestational age (`ga`) in weeks (measured to the nearest day), and three fetal-size measurements: abdominal circumference (`ac`), head circumference (`hc`), and mandible (jawbone) length (`ml`). The pairwise rank correlations and relative sample sizes are shown in table 1.

The data for `ac` and `hc` are $> 90\%$ complete, whereas only about a quarter of fetuses had mandible measurements. The clinical reason is that it is difficult to make accurate measurements of this structure. Values taken after 28 weeks' gestation are regarded by ultrasonographers as unreliable and have been discarded. Note the high rank correlations among all four variables.

Table 1: Rank correlations among variables in the fetal-size dataset ($n = 649$), based on available pairs of nonmissing observations. Values in parentheses are the percentage of complete observations for the variable or pair of variables in question. Only `ga` had complete data

Variable	`ac`	`hc`	`ml`	`ga`
`ac`	1.000 (94%)			
`hc`	0.991 (86%)	1.000 (92%)		
`ml`	0.948 (25%)	0.951 (25%)	1.000 (26%)	
`ga`	0.988 (94%)	0.989 (92%)	0.947 (26%)	1.000 (100%)

5.2 Developing the imputation model

I will use the data to illustrate certain aspects of multiple imputation and the use of `ice`. Imagine that we wished to get some idea of what mandible length looks like after 28 weeks. Such a result would clearly be an extrapolation beyond observed data and should not be regarded as reliable. However, the strong relationships among the body-structure measurements indicated by table 1 makes it appealing to investigate what (if anything) can be done with `ml`.

All three fetal-size variables show variance increasing with `ga`, the trend being largely removed by log transformation of fetal size. I will therefore work with the logarithmic transformations `lnac`, `lnhc`, and `lnml`. Investigation of the relationships between each log fetal-size variable and the others and `ga` was done with Stata's `mfp` command for multivariable fractional polynomial (MFP) modeling. To avoid instability, the log transformed variables were not further transformed in the model-building phase, whereas `ga` was allowed up to a second degree FP transformation. Further, due to the large proportion (74%) of missing data in `ml`, it seemed sensible to exclude `lnml` from the prediction of `lnac` and `lnhc`. Model selection was performed at the $\alpha = 0.05$ significance level. For `lnac`, for example, the command was

```
. mfp lnac lhhc ga, select(0.05) df(1, ga:4)
```

FP2 functions of `ga` were selected, with powers of $(-2, 3)$ for `lnac` and $(0, 2)$ for `lnhc`. For predicting `lnml`, only `lnac` and `lnhc` were needed, `ga` being eliminated as not significant at the 0.05 level, conditional on `lnac` and `lnhc`. Variance explained by the model was 99% for `lnac` and `lnhc` and 94% for `lnml`. Gestational age was complete and therefore required no imputation model.

5.3 Imputation

The prediction-equation matrix for `lnac`, `lnhc`, and `lnml` obtained by MFP modeling is shown in table 2.

Table 2: Prediction equations for the fetal-size dataset

Variable	**Predictor**			
	lnac	lnhc	lnml	ga
lnac	–	Linear	–	FP2$(-2,3)$
lnhc	Linear	–	–	FP2$(0,2)$
lnml	Linear	Linear	–	–
ga	–	–	–	–

The transformations FP2$(-2,3)$ and FP2$(0,2)$ were applied to $x =$ ga denoting, respectively, the models $\beta_0 + \beta_1 x^{-2} + \beta_2 x^3$ and $\beta_0 + \beta_1 \ln x + \beta_2 x^2$. The implementation in `ice` is as follows: first, the FP transformations are computed and then the imputation itself. For present purposes, we will create just one imputation and store the data in a new file called `fetalimp.dta`. The following commands are required:

```
. generate ga_1=ga^-2

. generate ga_2=ga^3

. generate ga_3=ln(ga)

. generate ga_4=ga^2

. ice lnac lnhc lnml ga_1 ga_2 ga_3 ga_4 using fetalimp, m(1) genmiss(m_)
> eq(lnac:lnhc ga_1 ga_2, lnhc:lnac ga_3 ga_4, lnml:lnac lnhc)
    Variable | Command      | Prediction equation
-------------+--------------+-------------------------------------------------
             |              |
        lnac | regress      | lnhc ga_1 ga_2
        lnhc | regress      | lnac ga_3 ga_4
        lnml | regress      | lnac lnhc
        ga_1 |              | [No missing data in estimation sample]
        ga_2 |              | [No missing data in estimation sample]
        ga_3 |              | [No missing data in estimation sample]
        ga_4 |              | [No missing data in estimation sample]

Imputing 1..file fetalimp.dta saved
```

The results for `lnml` are shown as a graph of `lnml` against `ga` in figure 1.

(*Continued on next page*)

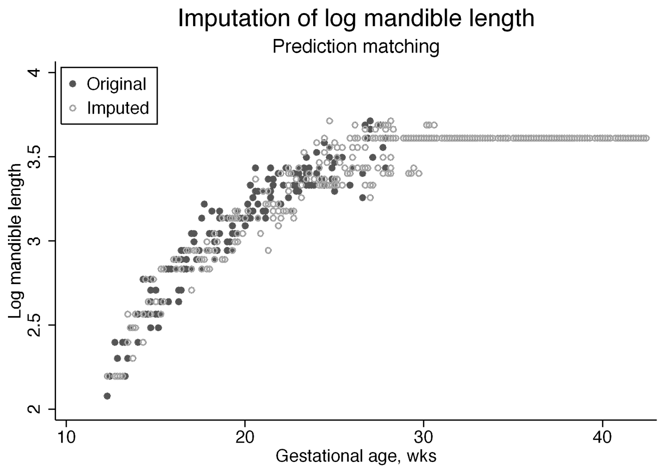

Figure 1: Original and imputed values of log mandible length using prediction matching

Although all missing values of `lnml` have been replaced, the results are unsatisfactory since a constant has been imputed for `ga` > 30 weeks. The reason is that, by default, `ice` has used prediction matching to obtain imputed values. Since the distribution of `lnml` is not represented in the data beyond 28 weeks' gestation, `ice` has been forced to match according to the predictions at the highest available `ga` values for which `lnml` is not missing.

To make the imputation more realistic and useful, prediction matching may be replaced by random draws from the posterior distribution of the log fetal-size variables. This is achieved by adding the option `draw(lnac lnhc lnml)` to the `ice` command. The plot equivalent to figure 1 is shown in figure 2.

(*Continued on next page*)

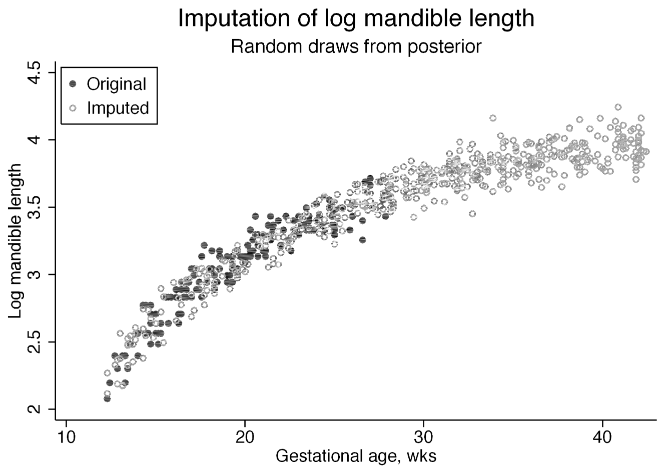

Figure 2: Original and imputed values of log mandible length using random draws from the posterior distribution of the three log fetal-size variables

The results are visually much more satisfactory. Mean log mandible length continues to increase smoothly right to the end of pregnancy, flattening off as the birth size of the fetus is approached. It should of course be stressed once again that the behavior seen in figure 2 is an extrapolation based on the assumption that the relationship between the four variables is similar in the extrapolation region to that in the region where data are observed. This is a strong assumption and, in the present example, is essentially uncheckable. However, the example does show the ability of `ice` to produce what seem to be reasonable estimates of unobserved quantities, with a plausible amount of random variation injected.

If `ga` had missing values, the `ice` command would need to include the option `passive(ga_1:ga^-2\ga_2:ga^3\ga_3:ln(ga)\ga_4:ga^2)`. The revised command would create the prediction equation `lnac lnhc lnml` for `ga`.

6 Changes to micombine

6.1 New options

There are two new options for `micombine`: `impid(`*varname*`)` and `br`. Both have been added in response to user requests. The first of these options allows data created outside of `ice` to be analyzed by using `micombine`. *varname* is the name of a variable identifying the imputations. The number of imputations is determined as the number of unique values of *varname*. The default *varname* is `_j`, the name used by `ice` when creating a dataset of imputations. The second new option, `br`, defines more precise degrees of freedom (d.f.) and confidence interval for each estimated regression coefficient.

The method is due to Barnard and Rubin (1999), hence the acronym `br`. Note that `br` is implemented for all models fit by `micombine`, not just linear regression, even though the methodology was developed in a linear regression context. Whether this is an improvement over the default inference and confidence intervals in the nonlinear-regression case remains to be investigated.

6.2 Postestimation aspects

Stata's `ereturn local`, `ereturn scalar`, and `ereturn matrix` commands set `e()` macros, scalars, and matrices returned by estimation commands. `micombine` respects this convention. It stores the sample size for one imputation (i.e., of the original data); mean regression coefficients; and estimated variance–covariance matrix, calculated according to the Rubin rules for multiple imputation, in `e(N)`, `e(b)`, and `e(V)`, respectively. The `test` and `testparm` commands work as expected following use of `micombine`, giving Wald tests of the desired parameters. Furthermore, `micombine` stores in `e(m_df)` the model d.f. and in `e(ll)` and `e(chi2)` the mean log likelihood and the mean model χ^2 statistic, each averaged over the m imputations.

The `predict` command evaluates predictions for all observations in the estimation sample using the parameters stored in `e(b)` and `e(V)`. Since the data vary across imputations, the predictions will vary correspondingly. The same principle applies to other postestimation commands that work at the level of the individual observations.

7 Conclusion

The methodology originally developed by Donald Rubin (1976) and others in the late 1970s and brilliantly implemented as the MICE system for more general use by Stef van Buuren and colleagues in the late 1990s is finally coming of age for the practitioner. I believe that this latest development will prove useful to Stata users. I hope also that it will provide a platform on which further numerical experiments in multiple imputation may be carried out. There are many outstanding questions to be tackled in the practical use of multiple imputation, including, for example, how to select an appropriate model and how to do regression diagnostics.

8 Acknowledgments

I thank numerous email correspondents and, in particular, Jennifer Hill for suggesting improvements and new features for `mvis` and `micombine`, and equally important, for pointing out unwanted "features" of the software. Without such helpful feedback, it is that much more difficult to develop flexible, reliable, and useful software.

9 References

Altman, D. G. and L. S. Chitty. 1993. Design and analysis of studies to derive charts of fetal size. *Ultrasound in Obstetrics and Gynecology* 3: 378–384.

Barnard, J. and D. B. Rubin. 1999. Small-sample degrees of freedom with multiple imputation. *Biometrika* 86: 948–955.

Carlin, J. B., N. Li, P. Greenwood, and C. Coffey. 2003. Tools for analyzing multiple imputed datasets. *Stata Journal* 3(3): 226–244.

Royston, P. 2004. Multiple imputation of missing values. *Stata Journal* 4(3): 227–241.

Rubin, D. B. 1976. Inference and missing data (with discussion). *Biometrika* 63: 581–592.

—. 1987. *Multiple Imputation for Nonresponse in Surveys*. New York: Wiley.

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 25 years of experience, with a strong interest in biostatistical methodology and in statistical computing and algorithms. At present, he works in clinical trials and related research issues in kidney cancer and other cancers. Currently, he is focusing on problems of model building and validation with survival data, including prognostic factors studies; on complex sample size problems in clinical trials with a survival-time endpoint; on issues in the design and analysis of microarray studies; and on novel trial designs.

The Stata Journal (2005)
5, Number 4, pp. 527–536

Multiple imputation of missing values: Update of ice

Patrick Royston
Cancer Group
MRC Clinical Trials Unit
222 Euston Road
London NW1 2DA
UK

1 Introduction

Royston (2004) introduced `mvis`, an implementation for Stata of MICE, a method of multiple multivariate imputation of missing values under missing-at-random (MAR) assumptions. In a second article, Royston (2005) described `ice`, an upgrade incorporating various improvements and changes to the software based on personal experience, discussion with colleagues, and user requests. This article describes an update to `ice`. The changes are less substantial but nevertheless important enough to warrant a brief explanation. The major modification is that the default method of imputing missing values in `ice` is now by sampling from the posterior predictive distribution rather than by predicted mean matching.

The `ice` system comprises five ado-files: `ice`, `micombine`, `mijoin`, `misplit`, and `uvis`. The last three programs have not been changed and are included in the present release for the sake of completeness.

2 Syntax

`ice` *mainvarlist* `using` *filename* [*if*] [*in*] [*weight*] [`,` `boot`[`(`*varlist*`)`]
`cc(`*ccvarlist*`)` `cmd(`*cmdlist*`)` `cycles(`*#*`)` `dryrun` `eq(`*eqlist*`)` `genmiss(`*string*`)`
`id(`*string*`)` `m(`*#*`)` `match`[`(`*varlist*`)`] `on(`*varlist*`)` `noconstant` `noshoweq`
`passive(`*passivelist*`)` `replace` `seed(`*#*`)` `substitute(`*sublist*`)` `trace(`*filename*`)`]

`uvis` *regression_cmd* *yvar* *xvarlist* [*if*] [*in*] [*weight*]`,` `gen(`*newvarname*`)`
[`noconstant` `boot` `match` `replace` `seed(`*#*`)`]

where *regression_cmd* may be `logistic`, `logit`, `mlogit`, `ologit`, or `regress`. All weight types supported by *regression_cmd* are allowed.

`micombine` *regression_cmd* [*yvar*] [*covarlist*] [*if*] [*in*] [*weight*] [`, br noconstant detail eform(`*string*`) genxb(`*newvarname*`) impid(`*varname*`) lrr obsid(`*varname*`)` *regression_cmd_options*]

where *regression_cmd* may be `clogit`, `cnreg`, `glm`, `logistic`, `logit`, `mlogit`, `nbreg`, `ologit`, `oprobit`, `poisson`, `probit`, `qreg`, `regress`, `rreg`, `stcox`, `streg`, or `xtgee`. All weight types supported by *regression_cmd* are allowed.

`mijoin, clear` [`m(`#`) impid(`*varname*`)`]

`misplit, clear` [`m(`#`) impid(`*varname*`)`]

3 Options

Only the changes to options are described.

3.1 Options for ice

`draw`[`(`*varlist*`)`] has been replaced with `match`[`(`*varlist*`)`]. `match`[`(`*varlist*`)`] instructs that each member of *varlist* be imputed with the `match` option of `uvis`. This option provides prediction matching for each member of *varlist*. If (*varlist*) is omitted, all relevant variables are imputed with the `match` option of `uvis`. The default, if `match()` is not specified, is to draw from the posterior predictive distribution of each variable requiring imputation.

`trace(`*filename*`)` allows one to monitor the convergence of the MICE algorithm. For each original variable with missing values, the mean of the imputed values is stored as a variable in *filename*, together with the cycle number at which that mean was calculated. The results are stored only for the final imputation. For diagnostic purposes, it is sensible to run `trace()` with `m(1)` and many cycles, such as `cycles(100)`. When the run is complete, it is helpful to load *filename* into memory and plot the mean for each imputed variable against the cycle number. If necessary, smoothing may be applied to clarify any apparent pattern. Convergence is judged to have occurred when the pattern of the imputed means is random. The number of cycles needed for convergence is usually obvious from the appearance of the plot.

3.2 Options for uvis

`draw` has been replaced with `match`. `match` creates imputations by prediction matching. The default is to draw imputations at random from the posterior distribution of the missing values of *yvar*, conditional on the observed values and the members of *xvarlist*.

4 What is new?

The principal changes to `ice` are as follows:

1. The default method of imputation involves drawing from the posterior predictive distribution.
2. With prediction matching in `uvis`, imputation is made at random among candidate values of *yvar* if more than one observation satisfies the matching criterion. Previously, it was likely that just one value of *yvar* would be selected in this situation, giving inappropriately restricted imputations.
3. When arranging the system of chained equations that is the heart of the MICE algorithm, variables are imputed in order of increasing missingness. The variable with the least missingness is imputed first, followed by that with the second, lowest amount, and so on. This approach may speed up convergence to the conditional distribution for each variable. Previously the order was arbitrary (it was base on the order of variables in *mainvarlist*).
4. `ice` reports the number of observations containing 0, 1, 2, ... missing values before proceeding with the imputation. Use of the `dryrun` option also gives this report.

5 Example

I will compare in a simple example with artificial data the use of `match` with drawing from the posterior. The dataset `test2.dta` contains $n = 120$ observations on two variables, `x` and `y`, related by the equation

$$y_i = 6x_i + e_i$$

where the errors e_i are normally distributed with mean 0 and variance 1. `y` and `x` are strongly correlated (Pearson $r = .985$). Forty values each of `x` and `y` are deleted completely at random, leaving a dataset in which 40 pairs of values of `x` and `y` are observed and 80 pairs have a missing value of either `x` or `y`.

Figure 1 shows the relationship between `y` and `x` in one imputation using prediction matching with 1, 2, 3, 5, 7, or 10 cycles of the MICE algorithm.

(*Continued on next page*)

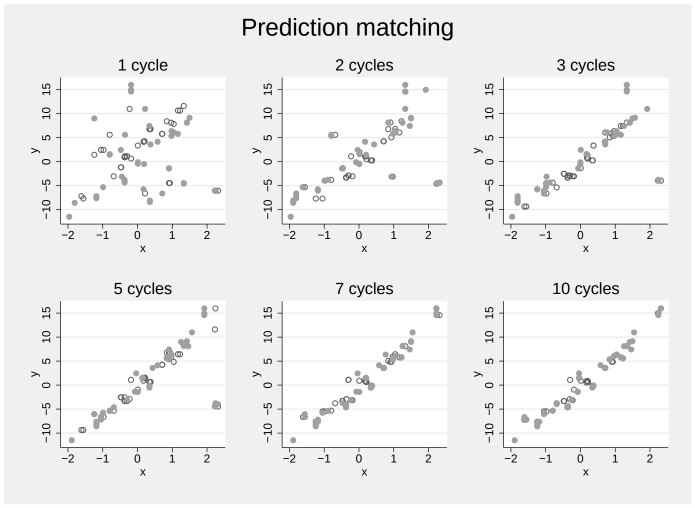

Figure 1: Artificial data. Imputation of `x` and `y` using prediction matching after different numbers of cycles of the MICE algorithm. Open circles, `y` missing; filled circles, `x` missing.

The Stata command used was

```
. ice x y using filename1, match(x y) seed(11) trace(filename2) cycles(100) m(1)
```

The open circles show values for which `y` has been imputed, and the filled circles, values for which `x` has been imputed. The 40 pairs in which both `x` and `y` were observed are omitted. The algorithm appears to converge after about 10 cycles. Note the occurrence of "wild" points for small numbers of cycles that are far away from the line $y = 6x$.

Figure 2 shows a trace of the means of `x` and `y` for the first 100 cycles of the MICE algorithm, stored in *filename2*.

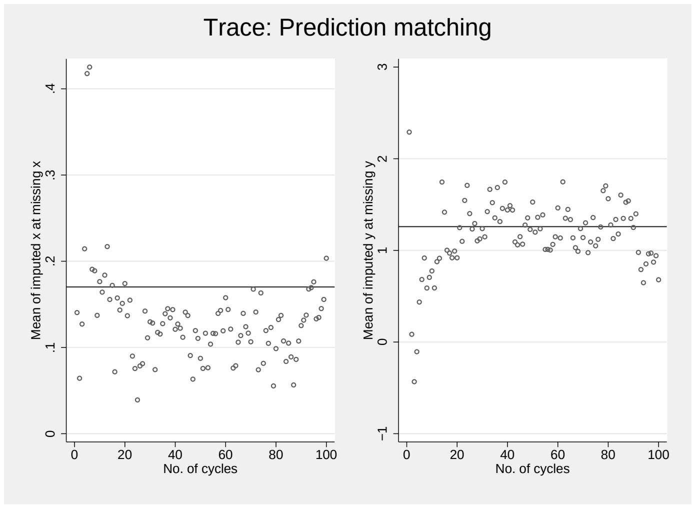

Figure 2: Artificial data. Trace of the first 100 cycles of the MICE algorithm using prediction matching. Horizontal lines, mean of original observations before being set to missing.

The means are initially wild and appear to stabilize after about 20 cycles. The horizontal lines show the mean of the original observations before being set to missing. Most of the imputed means of `x` are below or substantially below the correct value, suggesting the possibility of bias in the imputation of `x` in this example. Furthermore, there may be a tendency for the means to form a pattern of oscillation rather than the completely random appearance we would wish for.

Figure 3 repeats figure 1 but using draws from the posterior predictive distributions of `x` and `y` instead of prediction matching.

(*Continued on next page*)

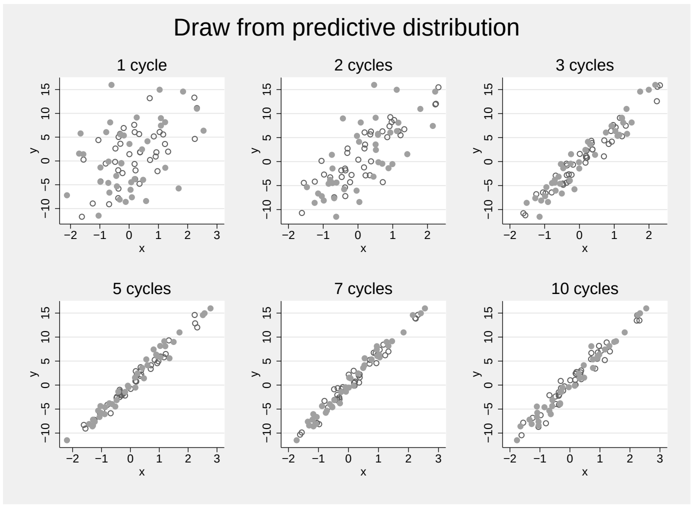

Figure 3: Artificial data. Imputation of `x` and `y` using draws from the posterior predictive distribution after different numbers of cycles of the MICE algorithm. Open circles, `y` missing; filled circles, `x` missing.

The Stata command is the same as before, except that `match(x y)` has been omitted to activate the default drawing algorithm. Two features are apparent. First, the algorithm settles down rapidly and smoothly, with no wild values appearing; the scatter about the line $y = 6x$ is progressively reduced as the number of cycles increases. About five cycles seems enough for convergence. Second, richer sets of imputations are created, since the algorithm is no longer restricted to imputing only observed values of `x` and `y`.

Finally, figure 4 repeats figure 2 for the draw method.

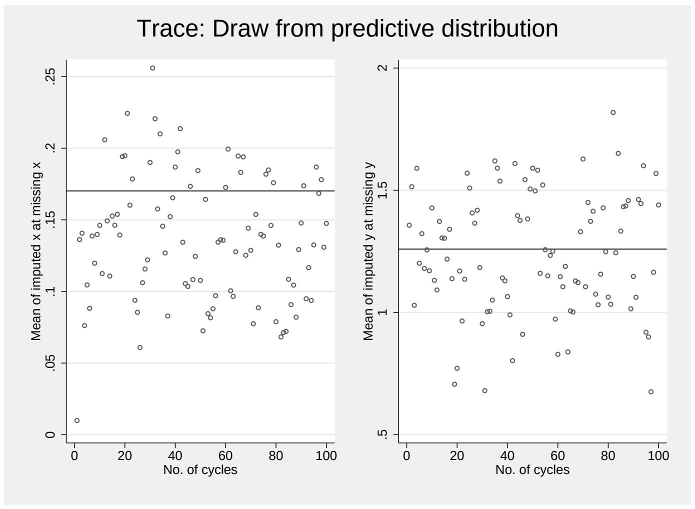

Figure 4: Artificial data. Trace of the first 100 cycles of the MICE algorithm using draws from the posterior predictive distributions.

The rapid convergence is clear. The imputed means of `x` now include the correct value. There is no tendency to oscillation.

The example clearly points to the advantages of the draw method when the normality assumptions for continuous variables are fulfilled, as here. The draw method is many times faster than prediction matching.

I now consider in general terms what may be done using data from imputing continuous variables when the normality assumptions fail.

6 Imputing continuous variables

When a continuous variable X has missing values, there are essentially four options for imputing it with `ice`:

1. Assume normality for X and draw from the posterior predictive distribution (the default). Example of `ice` command with `test2.dta`:

```
. ice x y using filename, m(20)
```

2. Transform X toward (approximate) normality and draw from the posterior predictive distribution. Retransform back to the original scale. For example,

```
. gen logx=log(x)
. gen logy=log(y)
. ice logx logy using filename, m(20)
. use filename, clear
. replace x=exp(logx)
. replace y=exp(logy)
```

3. Use prediction matching. For example,

```
. ice x y using filename, m(20) match(x y)
```

4. Use ordinal logistic regression (`ologit`). For example,

```
. ice x y using filename, m(20) cmd(ologit)
```

Option 1 is optimal if the normality assumption is (reasonably) appropriate, as in the above example. However, both normality and a continuous distribution for X are assumed. An observed distribution that is heavily grouped or rounded may not give sensible imputed values, since imputations will fall between the observed values. Furthermore, because of the effect of grouping the standard deviation may be incorrect. A possibility is to round the imputed values to resemble the pattern in the observed distribution.

Option 2 should be considered for positively skewed variables; the distribution may often resemble a lognormal. Again, if the original data are grouped, rounding may be considered after transformation back to the original scale. A related possibility is to use the more general Box–Cox transformation to normality (Stata's `boxcox` command).

Option 3 is a reasonable general choice, though concerns exist that prediction matching may give biased imputations, convergence may be slow, and computation may be lengthy (compounded by the need for more MICE cycles).

Option 4 is particularly useful with ordinal variables that either are intrinsically categorical or take a restricted set of values because rounding has been applied. In Stata, the `ologit` command is restricted to response variables with 50 or fewer categories, so variables with more than 50 distinct values will need to be grouped or rounded before imputation is performed.

6.1 Example

As an example of the problems of option 1, imputation with an inappropriate assumption of normality, figure 5 shows the distribution of the variable `x5` (number of positive lymph nodes) in the breast cancer dataset `brcaex.dta` analyzed by Royston (2004).

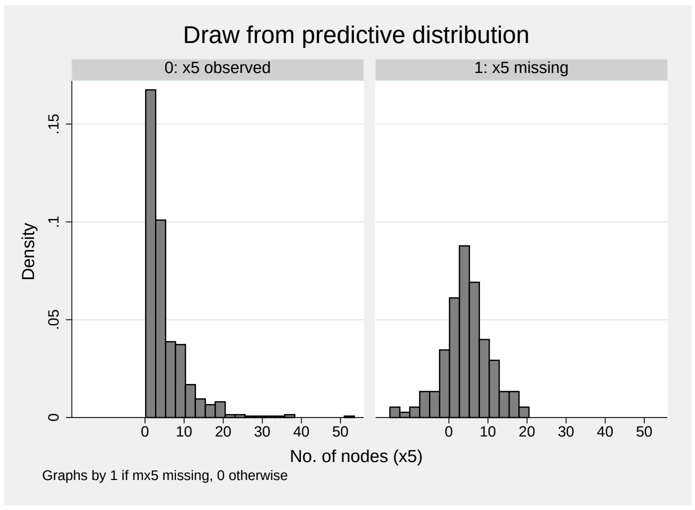

Figure 5: Breast cancer data. Imputation of `x5` by drawing from the posterior predictive distribution, assuming normality. Left panel: distribution of observed `x5`. Right panel: distribution of imputed `x5`.

The distribution of `x5` takes the integers 1, 2, ..., and is highly positively skewed, with more than 25% of the values being 1. The imputed values are symmetrically distributed about the mean of 5, and many are negative. As an alternative (option 4), figure 6 shows the results of using drawing and `ologit`.

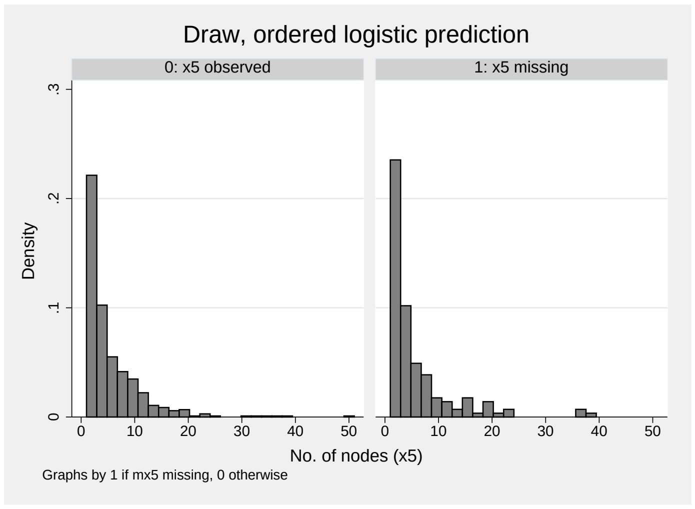

Figure 6: Breast cancer data. Imputation of `x5` by drawing from the posterior predictive distribution, using ordinal logistic regression (`ologit` command). Left panel: distribution of observed `x5`. Right panel: distribution of imputed `x5`.

The distribution of imputed values of `x5` is similar to that of the nonmissing observations—as it should be, given that the missing values were assigned completely at random. The results from prediction matching are much the same as this. The log transformation performs rather less well, although much better than not transforming; the distributional shape does not come out quite right.

7 Conclusion

This paper further develops the MICE software for Stata. It should be seen as work in progress. As experience and knowledge increase, I expect to issue further updates of `ice`.

8 Acknowledgment

I am grateful to Gillian Raab for pointing out issues in the earlier release of `ice` and for providing examples that illustrate some of the problems.

9 References

Royston, P. 2004. Multiple imputation of missing values. *Stata Journal* 4(3): 227–241.

———. 2005. Multiple imputation of missing values: update. *Stata Journal* 5(2): 188–201.

About the Author

Patrick Royston is a medical statistician with 25 years of experience, with a strong interest in biostatistical methodology and in statistical computing and algorithms. He works in clinical trials and related research issues in kidney cancer and other cancers. Currently, he is focusing on problems of model building and validation with survival data, including prognostic factors studies; on complex sample size problems in clinical trials with a survival-time endpoint; on writing a book on multivariable regression modeling; and on new trial designs.