Title

stata.com

mi impute chained — Impute missing values using chained equations

Description

`mi impute chained` fills in missing values in multiple variables iteratively by using chained equations, a sequence of univariate imputation methods with fully conditional specification (FCS) of prediction equations. It accommodates arbitrary missing-value patterns. You can perform separate imputations on different subsets of the data by specifying the `by()` option. You can also account for frequency, analytic (with continuous variables only), importance, and sampling weights.

Menu

Statistics > Multiple imputation

Syntax

Default specification of prediction equations, basic syntax

`mi impute chained (`*uvmethod*`)` *ivars* [`=` *indepvars*] [*if*] [*weight*] [`,` *impute_options options*]

Default specification of prediction equations, full syntax

`mi impute chained` *lhs* [`=` *indepvars*] [*if*] [*weight*] [`,` *impute_options options*]

Custom specification of prediction equations

`mi impute chained` *lhsc* [`=` *indepvars*] [*if*] [*weight*] [`,` *impute_options options*]

where *lhs* is *lhs_spec* [*lhs_spec* [...]] and *lhs_spec* is

(*uvmethod* [*if*] [`,` *uvspec_options*]) *ivars*

lhsc is *lhsc_spec* [*lhsc_spec* [...]] and *lhsc_spec* is

(*uvmethod* [*if*] [`, include(`*xspec*`) omit(`*varlist*`) noimputed` *uvspec_options*]) *ivars*

ivars (or *newivar* if *uvmethod* is `intreg`) are the names of the imputation variables.

uvspec_options are `ascontinuous`, `noisily`, and the method-specific *options* as described in the manual entry for each univariate imputation method.

The `include()`, `omit()`, and `noimputed` options allow you to customize the default prediction equations.

uvmethod	Description
regress	linear regression for a continuous variable; [MI] **mi impute regress**
pmm	predictive mean matching for a continuous variable; [MI] **mi impute pmm**
truncreg	truncated regression for a continuous variable with a restricted range; [MI] **mi impute truncreg**
intreg	interval regression for a continuous partially observed (censored) variable; [MI] **mi impute intreg**
logit	logistic regression for a binary variable; [MI] **mi impute logit**
ologit	ordered logistic regression for an ordinal variable; [MI] **mi impute ologit**
mlogit	multinomial logistic regression for a nominal variable; [MI] **mi impute mlogit**
poisson	Poisson regression for a count variable; [MI] **mi impute poisson**
nbreg	negative binomial regression for an overdispersed count variable; [MI] **mi impute nbreg**

impute_options	Description
Main	
* add(#)	specify number of imputations to add; required when no imputations exist
* replace	replace imputed values in existing imputations
rseed(#)	specify random-number seed
double	store imputed values in double precision; the default is to store them as float
by(*varlist* [, *byopts*])	impute separately on each group formed by *varlist*
Reporting	
dots	display dots as imputations are performed
noisily	display intermediate output
nolegend	suppress all table legends
Advanced	
force	proceed with imputation, even when missing imputed values are encountered
noupdate	do not perform mi update; see [MI] **noupdate option**

*add(#) is required when no imputations exist; add(#) or replace is required if imputations exist.
noupdate does not appear in the dialog box.

options	Description
MICE options	
burnin(#)	specify number of iterations for the burn-in period; default is burnin(10)
chainonly	perform chained iterations for the length of the burn-in period without creating imputations in the data
augment	perform augmented regression in the presence of perfect prediction for all categorical imputation variables
noimputed	do not include imputation variables in any prediction equation
bootstrap	estimate model parameters using sampling with replacement
savetrace(...)	save summaries of imputed values from each iteration in *filename*.dta
Reporting	
dryrun	show conditional specifications without imputing data
report	show report about each conditional specification
chaindots	display dots as chained iterations are performed
showevery(#)	display intermediate results from every #th iteration
showiter(*numlist*)	display intermediate results from every iteration in *numlist*
Advanced	
orderasis	impute variables in the specified order
nomonotone	impute using chained equations even when variables follow a monotone-missing pattern; default is to use monotone method
nomonotonechk	do not check whether variables follow a monotone-missing pattern

You must mi set your data before using mi impute chained; see [MI] **mi set**.

You must mi register *ivars* as imputed before using mi impute chained; see [MI] **mi set**.

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

collect is allowed; see [U] **11.1.10 Prefix commands**.

fweights, aweights (regress, pmm, truncreg, and intreg only), iweights, and pweights are allowed; see [U] **11.1.6 weight**.

Options

Main

add(), replace, rseed(), double, by(); see [MI] **mi impute**.

The following options appear on a Specification dialog that appears when you click on the **Create ...** button on the **Main** tab. The include(), omit(), and noimputed options allow you to customize the default prediction equations.

include(*xspec*) specifies that *xspec* be included in prediction equations of all imputation variables corresponding to the current left-hand-side specification *lhsc_spec*. *xspec* includes complete variables and expressions of imputation variables bound in parentheses. If the noimputed option is specified within *lhsc_spec* or with mi impute chained, then *xspec* may also include imputation variables. *xspec* may contain factor variables; see [U] **11.4.3 Factor variables**.

omit(*varlist*) specifies that *varlist* be omitted from the prediction equations of all imputation variables corresponding to the current left-hand-side specification *lhsc_spec*. *varlist* may include complete variables or imputation variables. *varlist* may contain factor variables; see [U] **11.4.3 Factor**

variables. In `omit()`, you should list variables to be omitted exactly as they appear in the prediction equation (abbreviations are allowed). For example, if variable `x1` is listed as a factor variable, use `omit(i.x1)` to omit it from the prediction equation.

`noimputed` specifies that no imputation variables automatically be included in prediction equations of imputation variables corresponding to the current *uvmethod*.

uvspec_options are options specified within each univariate imputation method, *uvmethod*. *uvspec_options* include `ascontinuous`, `noisily`, and the method-specific *options* as described in the manual entry for each univariate imputation method.

`ascontinuous` specifies that categorical imputation variables corresponding to the current *uvmethod* be included as continuous in all prediction equations. This option is only allowed when *uvmethod* is `logit`, `ologit`, or `mlogit`.

`noisily` specifies that the output from the current univariate model fit to the observed data be displayed. This option is useful in combination with the `showevery(#)` or `showiter(`*numlist*`)` option to display results from a particular univariate imputation model for specific iterations.

MICE options

`burnin(#)` specifies the number of iterations for the burn-in period for each chain (one chain per imputation). The default is `burnin(10)`. This option specifies the number of iterations necessary for a chain to reach approximate stationarity or, equivalently, to converge to a stationary distribution. The required length of the burn-in period will depend on the starting values used and the missing-data patterns observed in the data. It is important to examine the chain for convergence to determine an adequate length of the burn-in period prior to obtaining imputations; see *Convergence of MICE*. The provided default is what current literature recommends. However, you are responsible for determining that sufficient iterations are performed.

`chainonly` specifies that `mi impute chained` perform chained iterations for the length of the burn-in period and then stop. This option is useful in combination with `savetrace()` to examine the convergence of the method prior to imputation. No imputations are created when `chainonly` is specified, so `add()` or `replace` is not required with `mi impute chained, chainonly` and they are ignored if specified.

`augment` specifies that augmented regression be performed if perfect prediction is detected. By default, an error is issued when perfect prediction is detected. The idea behind the augmented-regression approach is to add a few observations with small weights to the data during estimation to avoid perfect prediction. See *The issue of perfect prediction during imputation of categorical data* under *Remarks and examples* in [MI] **mi impute** for more information. `augment` is not allowed with importance weights. This option is equivalent to specifying `augment` within univariate specifications of all categorical imputation methods: `logit`, `ologit`, and `mlogit`.

`noimputed` specifies that no imputation variables automatically be included in any of the prediction equations. This option is seldom used. This option is convenient if you wish to use different sets of imputation variables in all prediction equations. It is equivalent to specifying `noimputed` within all univariate specifications.

`bootstrap` specifies that posterior estimates of model parameters be obtained using sampling with replacement; that is, posterior estimates are estimated from a bootstrap sample. The default is to sample the estimates from the posterior distribution of model parameters or from the large-sample normal approximation of the posterior distribution. This option is useful when asymptotic normality of parameter estimates is suspect. This option is equivalent to specifying `bootstrap` within all univariate specifications.

`savetrace(`*filename*`[ , `*traceopts*`])` specifies to save the means and standard deviations of imputed values from each iteration to a Stata dataset called *filename*`.dta`. If the file already exists, the `replace` suboption specifies to overwrite the existing file. `savetrace()` is useful for monitoring convergence of the chained algorithm. This option cannot be combined with `by()`.

traceopts are `replace`, `double`, and `detail`.

`replace` indicates that *filename*`.dta` be overwritten if it exists.

`double` specifies that the variables be stored as `doubles`, meaning 8-byte reals. By default, they are stored as `floats`, meaning 4-byte reals. See [D] **Data types**.

`detail` specifies that additional summaries of imputed values including the smallest and the largest values and the 25th, 50th, and 75th percentiles are saved in *filename*`.dta`.

Reporting

`dots`, `noisily`, `nolegend`; see [MI] **mi impute**. `noisily` specifies that the output from all univariate conditional models fit to the observed data be displayed. `nolegend` suppresses all imputation table legends that include a legend with the titles of the univariate imputation methods used, a legend about conditional imputation when `conditional()` is used within univariate specifications, and group legends when `by()` is specified.

`dryrun` specifies to show the conditional specifications that would be used to impute each variable without actually imputing data. This option is recommended for checking specifications of conditional models prior to imputation.

`report` specifies to show a report about each univariate conditional specification. This option, in combination with `dryrun`, is recommended for checking specifications of conditional models prior to imputation.

`chaindots` specifies that all chained iterations be displayed as dots. An `x` is displayed for every failed iteration.

`showevery(#)` specifies that intermediate regression output be displayed for every #th iteration. This option requires `noisily`. If `noisily` is specified with `mi impute chained`, then the output from the specified iterations is displayed for all univariate conditional models. If `noisily` is used within a univariate specification, then the output from the corresponding univariate model from the specified iterations is displayed.

`showiter(`*numlist*`)` specifies that intermediate regression output be displayed for each iteration in *numlist*. This option requires `noisily`. If `noisily` is specified with `mi impute chained`, then the output from the specified iterations is displayed for all univariate conditional models. If `noisily` is used within a univariate specification, then the output from the corresponding univariate model from the specified iterations is displayed.

Advanced

`force`; see [MI] **mi impute**.

`orderasis` requests that the variables be imputed in the specified order. By default, variables are imputed in order from the most observed to the least observed.

`nomonotone`, a rarely used option, specifies not to use monotone imputation and to proceed with chained iterations even when imputation variables follow a monotone-missing pattern. `mi impute chained` checks whether imputation variables have a monotone missing-data pattern and, if they do, imputes them using the monotone method (without iteration). If `nomonotone` is used, `mi impute chained` imputes variables iteratively even if variables are monotone-missing.

`nomonotonechk` specifies not to check whether imputation variables follow a monotone-missing pattern. By default, `mi impute chained` checks whether imputation variables have a monotone missing-data pattern and, if they do, imputes them using the monotone method (without iteration). If `nomonotonechk` is used, `mi impute chained` does not check the missing-data pattern and imputes variables iteratively even if variables are monotone-missing. Once imputation variables are established to have an arbitrary missing-data pattern, this option may be used to avoid potentially time-consuming checks; the monotonicity check may be time consuming when a large number of variables is being imputed.

The following option is available with `mi impute` but is not shown in the dialog box:

`noupdate`; see [MI] **noupdate option**.

Remarks and examples

stata.com

Remarks are presented under the following headings:

See [MI] **mi impute** for a general description and details about options common to all imputation methods, *impute_options*. Also see [MI] **Workflow** for general advice on working with `mi`.

Multivariate imputation using chained equations

When a missing-data structure is monotone distinct, multiple variables can be imputed sequentially without iteration by using univariate conditional models (see [MI] **mi impute monotone**). Such monotone imputation is impossible with arbitrary missing-data patterns, and simultaneous imputation of multiple variables in such cases requires iteration. We described the impact of an arbitrary missing-data pattern on multivariate imputation and two common imputation approaches used in such cases, the multivariate normal method and multivariate imputation using chained equations (MICE), in *Multivariate imputation* in [MI] **mi impute**. In this entry, we describe MICE, also known as imputation using FCS (van Buuren, Boshuizen, and Knook 1999) or sequential regression multivariate imputation (SRMI; Raghunathan et al. 2001), in more detail. We use the terms MICE, FCS, and SRMI interchangeably throughout the documentation.

MICE is similar to monotone imputation in the sense that it is also based on a series of univariate imputation models. Unlike monotone imputation, MICE uses FCSs of prediction equations (chained equations) and requires iteration. Iteration is needed to account for possible dependence of the estimated model parameters on the imputed data when a missing-data structure is not monotone distinct.

The general idea behind MICE is to impute multiple variables iteratively via a sequence of univariate imputation models, one for each imputation variable, with fully conditional specifications of prediction equations: all variables except the one being imputed are included in a prediction equation. Formally, for imputation variables $X_1, X_2, \ldots, X_p$ and complete predictors (independent variables) $\mathbf{Z}$, this procedure can be described as follows. Imputed values are drawn from

$$
\begin{aligned}
X_1^{(t+1)} &\sim g_1(X_1|X_2^{(t)},\ldots,X_p^{(t)},\mathbf{Z},\boldsymbol{\phi}_1)\\
X_2^{(t+1)} &\sim g_2(X_2|X_1^{(t+1)},X_3^{(t)},\ldots,X_p^{(t)},\mathbf{Z},\boldsymbol{\phi}_2)\\
&\ldots\\
X_p^{(t+1)} &\sim g_p(X_p|X_1^{(t+1)},X_2^{(t+1)},\ldots,X_{p-1}^{(t+1)},\mathbf{Z},\boldsymbol{\phi}_p)
\end{aligned}
\tag{1}
$$

for iterations $t = 0, 1, \ldots, T$ until convergence at $t = T$, where $\boldsymbol{\phi}_j$ are the corresponding model parameters with a uniform prior. The univariate imputation models, $g_j(\cdot)$, can each be of a different type (normal, logistic, etc.), as is appropriate for imputing X_j.

Fully conditional specifications (1) are similar to the Gibbs sampling algorithm (Geman and Geman 1984; Gelfand and Smith 1990), one of the MCMC methods for simulating from complicated multivariate distributions. In fact, in certain cases these specifications do correspond to a genuine Gibbs sampler. For example, when all X_j's are continuous and all $g_j(\cdot)$'s are normal linear regressions with constant variances, then (1) corresponds to a Gibbs sampler based on a multivariate normal distribution with a uniform prior for model parameters. Such correspondence does not hold in general because unlike the Gibbs sampler, the conditional densities $\{g_j(\cdot),\ j = 1, 2, \ldots, p\}$ may not correspond to any multivariate joint conditional distribution of $X_1, X_2, \ldots, X_p$ given $\mathbf{Z}$ (Arnold, Castillo, and Sarabia 2001). This issue is known as incompatibility of conditionals (for example, Arnold, Castillo, and Sarabia [1999]). When conditionals are not compatible, the MICE procedure may not converge to any stationary distribution, which can raise concerns about its validity as a principled statistical method; see *Compatibility of conditionals* and *Convergence of MICE* for more details.

Despite the lack of a general theoretical justification, MICE is very popular in practice. Its popularity is mainly due to the tremendous flexibility it offers for imputing various types of data arising in observational studies. Similarly to monotone imputation, the variable-by-variable specification of MICE allows practitioners to simultaneously impute variables of different types by choosing from several univariate imputation methods appropriate for each variable. Being able to specify a separate model for each variable provides an imputer with great flexibility in incorporating certain characteristics specific to each variable. For example, we can use predictive mean matching ([MI] **mi impute pmm**) or truncated regression ([MI] **mi impute truncreg**) to impute a variable with a restricted range. We can impute variables defined on a subsample using only observations in that subsample while using the entire sample to impute other variables; see *Conditional imputation* in [MI] **mi impute** for details. For more information about multivariate imputation using chained equations, see van Buuren, Boshuizen, and Knook (1999); Raghunathan et al. (2001); van Buuren et al. (2006); van Buuren (2007); White, Royston, and Wood (2011); and Royston (2004, 2005a, 2005b, 2007, 2009), among others.

The specification of a conditional imputation model $g_j(\cdot)$ includes an imputation method and a prediction equation relating an imputation variable to other explanatory variables. In what follows, we distinguish between the default specification (of prediction equations) in which the identities of the complete explanatory variables are the same across all prediction equations, and the custom specification in which the identities are allowed to differ.

Under the default specification, prediction equations of each imputation variable include all complete independent variables and all imputation variables except the one being imputed. Under the custom specification, each prediction equation may include a subset of the predictors that would be used under the default specification. The custom specification also allows expressions of imputation variables in prediction equations.

Model (1) corresponds to the default specification. For example, consider imputation variables X_1, X_2, and X_3 and complete predictors Z_1 and Z_2. Under the default specification, the individual prediction equations are determined as follows. The most observed variable—say, X_1—is predicted from X_2, X_3, Z_1, and Z_2. The next most observed variable—say, X_2—is predicted from X_3, Z_1,

Z_2, and previously imputed X_1. The least observed variable, X_3, is predicted from Z_1, Z_2, and previously imputed X_1 and X_2. (A constant is included in all prediction equations, by default.) We use the following notation to refer to the above sequence of prediction equations (imputation sequence): $X_1|\mathbf{X}_{-1}, Z_1, Z_2 \rightarrow X_2|\mathbf{X}_{-2}, Z_1, Z_2 \rightarrow X_3|\mathbf{X}_{-3}, Z_1, Z_2$, where $\mathbf{X}_{-j}$ denotes all imputed or to-be-imputed variables except X_j.

A sequence such as $X_1|\mathbf{X}_{-1}, Z_1 \rightarrow X_2|\mathbf{X}_{-2}, Z_1, Z_2 \rightarrow X_3|\mathbf{X}_{-3}, Z_2$ would correspond to a custom specification. Here X_1 is assumed to be conditionally independent of Z_2 given $\mathbf{X}_{-1}$ and Z_1, and X_3 is assumed to be conditionally independent of Z_1 given $\mathbf{X}_{-3}$ and Z_2.

Compatibility of conditionals

A concern with MICE is its lack of a formal theoretical justification. Its theoretical weakness is possible incompatibility of fully conditional specifications (1). As we briefly mentioned earlier, it is possible to specify a set of full conditionals with MICE for which no multivariate distribution exists (for example, van Buuren et al. [2006] and van Buuren [2007]). In such a case, the validity of MICE as a statistical procedure is questionable.

The impact of incompatibility of conditional specifications in practice is still under investigation. For example, van Buuren et al. (2006) performed several simulations to investigate the consequences of strongly incompatible specifications on multiple-imputation (MI) results in a simple setting and found very little impact of it on estimated parameters. The effect of incompatible conditionals on the quality of imputations and final MI inference in general is not yet known. Of course, if a joint model is of main scientific interest, then incompatibility of conditionals poses a problem. In the discussion of Arnold, Castillo, and Sarabia (2001), Andrew Gelman and Trivellore Raghunathan mention that the existence of an underlying joint distribution may be less important within the imputation context than the ability to incorporate the unique features of the data.

For more information about the compatibility of conditional specifications, see Arnold, Castillo, and Sarabia (2001); van Buuren (2007); and Arnold, Castillo, and Sarabia (1999) and references therein.

Convergence of MICE

MICE is an iterative method and is similar in spirit to the Gibbs sampler, an MCMC method. Similarly to MCMC methods, MICE builds a sequence of draws $\{\mathbf{X}_m^{(t)} : t = 1, 2, \ldots\}$, a chain, and iterates until this chain reaches a stationary distribution. So as with any MCMC method, monitoring convergence is important with MICE.

MICE performs simulation by running multiple independent chains (see *Convergence of iterative methods* in [MI] **mi impute**). To assess convergence of multiple chains, we need to examine the stationarity of each chain by the end of the specified burn-in period b. In practice, convergence of MICE is often examined visually. Trace plots—plots of summaries of the distribution (means, standard deviations, quantiles, etc.) of imputed values against iteration numbers—are used to examine stationarity of the chain. Long-term trends in trace plots are indicative of slow convergence to stationarity. A suitable value for the burn-in period b can be inferred from a trace plot as the earliest iteration after which each chain does not exhibit a visible trend and the fluctuations in values become more regular. When the initial values are close to the mode of the target posterior distribution (when one exists), b will generally be small. When the initial values are far off in the tails of the posterior distribution, the initial number of iterations b will generally be larger.

The number of iterations necessary for MICE to converge depends on, among other things, the fractions of missing information and initial values. The higher the fractions of missing information and the farther the initial values are from the mode of the posterior predictive distribution of missing data, the slower the convergence, and thus the larger the number of iterations required. Current literature suggests that in many practical applications a low number of burn-in iterations, somewhere between 5 and 20 iterations, is usually sufficient for convergence (for example, van Buuren [2007]). In any case, examination of the data and missing-data patterns is highly recommended when investigating convergence of MICE.

The convergence of MICE may not be achieved when specified conditional models are incompatible, as described in *Compatibility of conditionals*. The simulation draws will depend on the order in which variables are imputed and on the chosen length of the burn-in period. It is important to evaluate the quality of imputations (see *Imputation diagnostics* in [MI] **mi impute**) to determine the impact of incompatibility on MI analysis.

First use

Before we describe various uses of `mi impute chained`, let's look at a simple example first.

Consider the heart attack data example examining the relationship between heart attacks and smoking from *Multivariate imputation* of [MI] **mi impute**, where the `age` and `bmi` variables contain missing values. In another version of the dataset, `bmi` and `age` have a nonmonotone missing-data pattern, and thus monotone imputation is not possible:

```
. use https://www.stata-press.com/data/r17/mheart8s0
(Fictional heart attack data; arbitrary pattern)
. mi misstable patterns, frequency
  Missing-value patterns
    (1 means complete)

             |  Pattern
  Frequency  |  1  2
-------------+------
         118 |  1  1
          24 |  1  0
           8 |  0  1
           4 |  0  0
-------------+------
         154 |
Variables are  (1) age  (2) bmi
```

`mi impute chained` does not require missing data to be monotone, so we can use it to impute missing values of `age` and `bmi` in this dataset. We use the same model specification as before:

```
. mi impute chained (regress) bmi age = attack smokes hsgrad female, add(10)
Conditional models:
                age: regress age bmi attack smokes hsgrad female
                bmi: regress bmi age attack smokes hsgrad female
Performing chained iterations ...
Multivariate imputation                     Imputations =       10
Chained equations                                 added =       10
Imputed: m=1 through m=10                       updated =        0
Initialization: monotone                     Iterations =      100
                                                burn-in =       10
                bmi: linear regression
                age: linear regression
```

Variable	Observations per m			
	Complete	Incomplete	Imputed	Total
bmi	126	28	28	154
age	142	12	12	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

As before, 10 imputations are created (the `add(10)` option). The linear regression imputation method (`regress`) is used to impute both continuous variables. The `attack`, `smokes`, `hsgrad`, and `female` variables are used as complete predictors (independent variables).

`mi impute chained` reports the conditional specifications used to impute each variable and the order in which they were imputed. By default, `mi impute chained` imputes variables in order from the most observed to the least observed. In our example, `age` has the least number of missing values and so is imputed first, even though we listed `bmi` before `age` in the command specification.

With the default specification, `mi impute chained` builds appropriate FCSs automatically using the supplied imputation variables and complete predictors, specified as right-hand-side variables. The default prediction equation for `age` includes `bmi` and all the complete predictors, and the default prediction equation for `bmi` includes `age` and all the complete predictors.

The main header and table output were described in detail in [MI] **mi impute**. The information specific to `mi impute chained` includes the type of initialization, the burn-in period, and the number of iterations. By default, `mi impute chained` uses 10 burn-in iterations (also referred to as cycles in the literature) before drawing imputed values. The total number of iterations performed by `mi impute chained` to obtain 10 imputations is 100. Also, similarly to `mi impute monotone`, the additional information above the table includes the legend describing what univariate imputation method was used to impute each variable. (If desired, this legend may be suppressed by specifying the `nolegend` option.)

Using mi impute chained

Below we summarize general capabilities of `mi impute chained`.

1. `mi impute chained` offers two main syntaxes—one using the default prediction equations and the other allowing customization of prediction equations. We will refer to the two syntaxes as default and custom, respectively. We describe the two syntaxes in detail in the next two sections.

2. `mi impute chained` allows specification of a global (outer) `if` condition,

   ```
   . mi impute chained ... if exp ...
   ```

   and equation-specific (inner) `if` conditions,

   ```
   . mi impute chained ... (... if exp ...) ...
   ```

   A global `if` is applied to all equations. You may combine global and equation-specific `if` conditions:

   ```
   . mi impute chained ... (... if exp ...) ... if exp ...
   ```

3. `mi impute chained` allows specification of global weights, which are applied to all equations:

   ```
   . mi impute chained ... [weight] ...
   ```

4. `mi impute chained` uses fully specified prediction equations by default. Customize prediction equations by including or omitting desired terms:

   ```
   . mi imp chain (... , include(z3) ...) (..., omit(z1) ...) ...
   ```

5. `mi impute chained` automatically includes appropriate imputation variables in prediction equations. Use a global `noimputed` option to prevent inclusion of imputation variables in all prediction equations:

   ```
   . mi impute chained ..., noimputed ...
   ```

   Or use an equation-specific `noimputed` option to prevent inclusion of imputation variables in only some prediction equations:

   ```
   . mi impute chained ... (..., noimputed ...) ...
   ```

As we mentioned earlier, `mi impute chained` is an iterative imputation method. By default, it performs 10 burn-in iterations for each imputation before drawing the final set of imputed values. The number of iterations is determined by the length of the burn-in period after which a random sequence (chain) is assumed to converge to its stationary distribution. The provided default may not be applicable to all situations, so you can use the `burnin()` option to modify it.

Use the `chainonly` and `savetrace()` options to determine the appropriate burn-in period. For example,

```
. mi impute chained ..., burnin(100) chainonly savetrace(impstats) ...
```

saves summaries of imputed values from 100 iterations for each of the imputation variables to `impstats.dta` without proceeding to impute data. You can apply techniques from *Convergence of MICE* to the data in `impstats.dta` to determine an adequate burn-in period.

Use a combination of the `dryrun` and `report` options to check the specification of each univariate imputation model prior to imputing data.

In the next two sections, we describe the use of `mi impute chained` first using hypothetical situations and then using real examples.

Default prediction equations

We showed in *First use* an example of `mi impute chained` with default prediction equations using the heart attack data. Here we provide more details about this default specification.

By default, `mi impute chained` imputes missing values by using the default prediction equations. It builds the corresponding univariate imputation models based on the supplied information: *uvmethod*, the imputation method; *ivars*, the imputation variables; and *indepvars*, the complete predictors or independent variables.

Suppose that continuous variables `x1`, `x2`, and `x3` contain missing values and are ordered from the most observed to the least observed. We want to impute these variables, and we decide to use the same univariate imputation method, say, linear regression, for all. We can do this by typing

```
. mi impute chained (regress) x1 x2 x3 ...
```

The above command corresponds to the first syntax diagram of `mi impute chained`: *uvmethod* is `regress` and *ivars* is `x1 x2 x3`. Relating the above to the model notation used in (1), g_1, g_2, g_3 represent linear regression imputation models and the prediction sequence is $X_1|X_2, X_3 \rightarrow X_2|X_1, X_3 \rightarrow X_3|X_1, X_2$.

By default, `mi impute chained` imputes variables in order from the most observed to the least observed, regardless of the order in which variables were specified. For example, we can list imputation variables in the reverse order,

```
. mi impute chained (regress) x3 x2 x1 ...
```

and `mi impute chained` will still impute `x1` first, `x2` second, and `x3` last. You can use the `orderasis` option to instruct `mi impute chained` to perform imputation of variables in the specified order.

If we have additional covariates containing no missing values (say, `z1` and `z2`) that we want to include in the imputation model, we can do so by typing

```
. mi impute chained (regress) x1 x2 x3 = z1 z2 ...
```

Now *indepvars* is `z1 z2` and the prediction sequence is $X_1|X_2, X_3, Z_1, Z_2 \rightarrow X_2|X_1, X_3, Z_1, Z_2 \rightarrow X_3|X_1, X_2, Z_1, Z_2$. Independent variables are included in the prediction equations of all univariate models.

Suppose that we want to use a different imputation method for one of the variables—we want to impute `x3` using predictive mean matching. We can do this by typing

```
. mi impute chained (regress) x1 x2 (pmm, knn(5)) x3  = z1 z2 ...
```

The above corresponds to the second syntax diagram of `mi impute chained`, a generalization of the first that accommodates differing imputation methods. The right-hand side of the equation is unchanged. `z1` and `z2` are included in all three prediction equations. The left-hand side now has two specifications: `(regress) x1 x2` and `(pmm, knn(5)) x3`. In previous examples, we had only one left-hand-side specification, *lhs_spec*—`(regress) x1 x2 x3`. (The number of left-hand-side specifications does not necessarily correspond to the number of univariate models; the latter is determined by the number of imputation variables.) In this example, `x1` and `x2` are imputed using linear regression, and `x3` is imputed using predictive mean matching with five nearest neighbors specified in `pmm`'s option `knn()`. All method-specific options must be specified within the parentheses surrounding the method.

Suppose now we want to restrict the imputation sample for `x2` to observations where `z1` is one; also see *Imputing on subsamples* of [MI] **mi impute**. The corresponding syntax is

```
. mi impute chained (regress) x1 (regress if z1==1) x2 (pmm, knn(5))
> x3  = z1 z2 ...
```

If, in addition to the above, we want to impute all variables using an overall subsample where `z3` is one, we can specify the global `if z3==1` condition:

```
. mi impute chained (regress) x1 (regress if z1==1) x2 (pmm, knn(5))
> x3  = z1 z2 if z3==1 ...
```

In the above, restrictions included only complete variables. When restrictions include imputation variables, you should use the `conditional()` option instead of an `if` condition; see *Conditional imputation* in [MI] **mi impute**. Suppose that we need to impute `x2` using only observations for which `x1` is positive, provided that missing values of `x1` are nested within missing values of `x2`. We can do this by typing

```
. mi impute chained (regress) x1 (regress, cond(if x1>0)) x2 (pmm, knn(5)) x3
> = z1 z2 ...
```

When any imputation variable is imputed using a categorical method (`logit`, `ologit`, or `mlogit`), `mi impute chained` automatically includes it as a factor variable in the prediction equations of other imputation variables. Suppose that `x1` is a categorical variable and is imputed using the multinomial logistic method:

```
. mi impute chained (mlogit) x1 (regress) x2 x3 ...
```

The above will result in the prediction sequence $X_1|X_2, X_3 \rightarrow X_2|\texttt{i.}X_1, X_3 \rightarrow X_3|\texttt{i.}X_1, X_2$ where $\texttt{i.}X_1$ denotes the factors of X_1.

If you wish to include a factor variable as continuous in prediction equations, you can use the `ascontinuous` option within the specification of the univariate imputation method for that variable:

```
. mi impute chained (mlogit, ascontinuous) x1 (regress) x2 x3 ...
```

As we discussed in *The issue of perfect prediction during imputation of categorical data* of [MI] **mi impute**, perfect prediction often occurs during imputation of categorical variables. One way of dealing with it is to use the augmented-regression approach (White, Daniel, and Royston 2010), available through the `augment` option. For example, if perfect prediction occurs during imputation of `x1` in the above, you can specify `augment` within the method specification of `x1` to perform augmented regression:

```
. mi impute chained (mlogit, augment) x1 (regress) x2 x3 ...
```

Alternatively, you can use the `augment` option with `mi impute chained` to perform augmented regression for all categorical variables for which the issue of perfect prediction arises:

```
. mi impute chained (mlogit) x1 (logit) x2 (regress) x3 ..., augment ...
```

The above is equivalent to specifying `augment` within each specification of a univariate categorical imputation method:

```
. mi impute chained (mlogit, augment) x1 (logit, augment) x2 (regress) x3 ...
```

Custom prediction equations

In the previous section, we considered various uses of `mi impute chained` with default prediction equations. Often, however, you may want to use different prediction equations for some or even all imputation variables. We can easily modify the above specifications to accommodate this.

Let's consider situations in which we want to use different sets of complete variables for some imputation variables first. Recall our following hypothetical example:

```
. mi impute chained (regress) x1 x2 x3 = z1 z2 ...                    (M1)
```

Suppose that we want to omit `z2` from the prediction equation for `x3`. To accommodate this, we need to include two separate specifications: one for `x1` and `x2` and one for `x3`:

```
. mi impute chained (regress) x1 x2 (regress, omit(z2)) x3  = z1 z2 ...
```

The above corresponds to the custom specification, the third syntax diagram, of `mi impute chained`. As before, we list all the complete variables *indepvars* to be included in all prediction equations to the right of the `=` sign. So, *indepvars* is still `z1 z2`. The prediction equation for `x3`, however, omits variable `z2`, specified within the `omit()` option. The prediction sequence for the above specification is $X_1|X_2, X_3, Z_1, Z_2 \rightarrow X_2|X_1, X_3, Z_1, Z_2 \rightarrow X_3|X_1, X_2, Z_1$.

Alternatively, we could have achieved the above by including variable `z1` in all prediction equations, as a right-hand-side specification *indepvars*, and using the `include()` option to add variable `z2` to the prediction equations of `x1` and `x2`:

```
. mi impute chained (regress, include(z2)) x1 x2 (regress) x3  = z1 ...
```

You may also want to modify the sets of imputation variables to be included in prediction equations. By default, `mi impute chained` automatically includes the appropriate fully conditional specifications of imputation variables in all prediction equations.

Suppose that in addition to different sets of complete predictors, we assume that X_1 and X_2 are conditionally independent given X_3, which implies that prediction equations for `x1` and `x2` include only `x3` and not each other. We can accommodate this with the command

```
. mi impute chained (regress, include(x3 z2) noimputed) x1 x2 (regress)    ///
            x3  = z1 ...
```

which corresponds to the prediction sequence $X_1|X_3, Z_1, Z_2 \rightarrow X_2|X_3, Z_1, Z_2 \rightarrow X_3|X_1, X_2, Z_1$.

The above is also equivalent to the command

```
. mi impute chained (regress, omit(x1 x2)) x1 x2 (regress, omit(z2))      ///
            x3  = z1 z2 ...
```

There are other equivalent ways of achieving the above custom specifications by using various combinations of `include()`, `omit()`, and `noimputed`. The most convenient specification will depend on your particular structure of the prediction equations. You can also combine these options within the same univariate specification.

It is important to realize that equivalent syntaxes may produce different (yet equivalent with stable imputation models) sequences of imputed values when they have different ordering of variables in prediction equations. `mi impute chained` builds prediction equations as follows. Appropriate imputation variables are included first, unless the `noimputed` option is specified. By default, imputation variables are included in order from the most observed to the least observed. If the `orderasis` option is used, the variables are included in the specified order. Next, terms specified in the `include()` option are included in the listed order. Then right-hand-side variables (*indepvars*) are included in the listed order. Finally, variables listed in the `omit()` option are removed from the prediction equation. When you specify `omit()`, it is important to specify variables as they are included in the prediction equation; if `x1` is included as a factor variable, `omit(i.x1)` should be used.

You can also include functions of imputation variables in prediction equations with the custom specification of `mi impute chained`. As we discussed in *Model building* in [MI] **mi impute**, there are two ways to do that. You can include functions of imputation variables as separate imputation variables directly in your imputation model or you can impute them passively using `mi impute chained`.

For example, using model (M1), suppose that we would like to include the interaction between x1 and x2 in the conditional model for x3:

```
. mi impute chained (regress) x1 x2                          ///
                    (regress, include((x1*x2))) x3           ///
                                                = z1 z2 ...
```

The expression x1*x2, specified in the include() option, is enclosed in parentheses.

We also could have typed

```
. mi impute chained (regress, include((x1*x2))) x1 x2 x3 = z1 z2 ...
```

and mi impute chained would appropriately include the interaction term X_1X_2 only in the prediction equation of X_3.

You can include any other expressions of imputation variables in include() within any of the left-hand-side specifications. Just remember to enclose such expressions in parentheses.

All the examples we considered in *Default prediction equations* are also applicable to mi impute chained with custom prediction equations. For example, to restrict imputation of x2 to observations where z1==1 in one of our earlier examples, we can type

```
. mi impute chained (reg) x1 (reg if z1==1) x2 (reg, omit(z2)) x3  = z1 z2 ...
```

Link between mi impute chained and mi impute monotone

Similarly to mi impute monotone (see [MI] **mi impute monotone**), mi impute chained uses a sequence of univariate imputation models to impute variables. So the use of mi impute chained is very similar to that of mi impute monotone except:

1. mi impute chained does not require that the specified imputation variables follow a monotone-missing pattern.
2. mi impute chained requires iteration to accommodate arbitrary missing-data patterns.
3. mi impute chained, by default, uses FCSs of the prediction equations where all specified complete variables and all imputation variables except the one being imputed are included in prediction equations.
4. mi impute chained provides an alternative way of specifying custom prediction equations to accommodate FCS of imputation variables.

When a missing-value pattern is monotone, mi impute chained defaults to the monotone method (unless nomonotone is specified) and produces the same results as mi impute monotone. However, using mi impute monotone in this case is faster because it performs the estimation step only once, on the original data, whereas mi impute chained performs estimation on every chained iteration.

The best approach to follow is

1. Check the missing-data pattern using misstable nested (or mi misstable nested if the data are already mi set; see [R] **misstable** or [MI] **mi misstable**) first.
2. If the missing-data pattern is monotone, use mi impute monotone to impute variables. If the missing-data pattern is not monotone, use mi impute chained to impute variables.

It is worth mentioning the difference between the documented custom syntaxes of mi impute chained and mi impute monotone.

With monotone imputation, variables are imputed in a particular, monotone-missing order and prediction equations are built in a particular way: previously imputed variables are added sequentially to the prediction equations of other imputation variables. So when building custom prediction equations, it is easier to construct one equation at a time in the order of the monotone missing pattern. As such, the custom syntax of `mi impute monotone`, as documented in [MI] **mi impute monotone**, requires full specification of a separate conditional model for each imputation variable in the monotone-missing order.

Imputation using chained equations does not require specific ordering in which variables must be imputed, although imputing variables in order from the most observed to the least observed usually leads to faster convergence. Also, because all imputation variables except the one being imputed are included in prediction equations, it does not matter in what order prediction equations are specified. The custom syntax of `mi impute chained` reflects this.

Examples

For the purpose of illustration, we use five imputations in our examples.

▷ Example 1: Different imputation methods

Recall the heart attack example from *First use*. If we wanted to impute `bmi` using predictive mean matching with, say, three nearest neighbors instead of linear regression, we could type

```
. use https://www.stata-press.com/data/r17/mheart8s0, clear
(Fictional heart attack data; arbitrary pattern)
. mi impute chained (pmm, knn(3)) bmi (reg) age = attack smokes hsgrad female,
> add(5)
Conditional models:
               age: regress age bmi attack smokes hsgrad female
               bmi: pmm bmi age attack smokes hsgrad female, knn(3)
Performing chained iterations ...
Multivariate imputation                     Imputations =        5
Chained equations                                 added =        5
Imputed: m=1 through m=5                        updated =        0
Initialization: monotone                     Iterations =       50
                                                burn-in =       10
               bmi: predictive mean matching
               age: linear regression
```

	Observations per *m*			
Variable	Complete	Incomplete	Imputed	Total
bmi	126	28	28	154
age	142	12	12	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

As shown previously, `mi impute chained` imputed `age` first and `bmi` second, because `age` is the variable with the fewest missing values.

◁

Example 2: Convergence of MICE

In *Convergence of MICE*, we described ways to assess convergence of the MICE algorithm. Continuing our previous example, let's investigate the trends in the summaries of imputed values of `age` and `bmi` over iterations.

Following the recommendation from *Using mi impute chained*, we use a combination of `chainonly` and `savetrace()` to perform chained iterations without creating imputations in the data and save summaries of imputed values to the new dataset `impstats.dta`. We perform 100 iterations and specify a random-number seed for reproducibility:

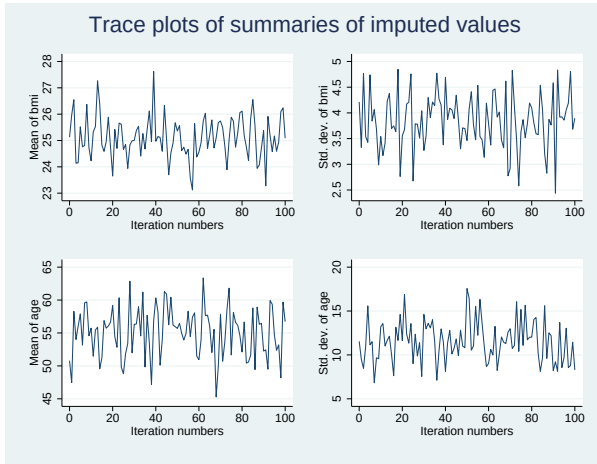
```
. use https://www.stata-press.com/data/r17/mheart8s0, clear
(Fictional heart attack data; arbitrary pattern)
. mi impute chained (pmm, knn(3)) bmi (reg) age = attack smokes hsgrad female,
> chainonly burnin(100) savetrace(impstats) rseed(1359)
Conditional models:
               age: regress age bmi attack smokes hsgrad female
               bmi: pmm bmi age attack smokes hsgrad female, knn(3)
Performing chained iterations ...
Note: No imputation performed.
```

By default, means and standard deviations of imputed values for each imputation variable are saved along with iteration and imputation numbers (imputation number is always 0 when `chainonly` is used):

```
. use impstats
(Summaries of imputed values from -mi impute chained-)
. describe
Contains data from impstats.dta
 Observations:           101                  Summaries of imputed values
                                                from -mi impute chained-
    Variables:             6                  30 Apr 2021 22:49
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
iter            byte    %12.0g                Iteration numbers
m               byte    %12.0g                Imputation numbers
age_mean        float   %9.0g                 Mean of age
age_sd          float   %9.0g                 Std. dev. of age
bmi_mean        float   %9.0g                 Mean of bmi
bmi_sd          float   %9.0g                 Std. dev. of bmi
-------------------------------------------------------------------------------
Sorted by:
```

We use the time-series command `tsline` (see [TS] **tsline**) to plot summaries of imputed values with respect to the iteration number. We first use `tsset` to set `iter` as the "time" variable and then use `tsline` to obtain trace plots. We create trace plots for all variables and combine them in one graph using `graph combine`:

```
. tsset iter
Time variable: iter, 0 to 100
        Delta: 1 unit
. tsline bmi_mean, name(gr1) nodraw
. tsline bmi_sd, name(gr2) nodraw
. tsline age_mean, name(gr3) nodraw
. tsline age_sd, name(gr4) nodraw
. graph combine gr1 gr2 gr3 gr4, title(Trace plots of summaries of imputed values)
> rows(2)
```



The trace plots show no apparent trends in the summaries of the imputed values, so the default number of burn-in iterations, 10, seems adequate. Although a low number of burn-in iterations may be sufficient in some applications, there are situations when larger numbers are required (for example, van Buuren [2007]).

It is also useful to look at several chains, each obtained using a different set of initial values, to check convergence and stability of the algorithm.

Let's look at three separate chains. The easiest way to do this is to use the `add()` option instead of `chainonly` to create three imputations. Remember that `mi impute chained` starts a new chain for each imputation, so a different set of initial values is used for each imputation. When `savetrace()` is specified, `mi impute chained` saves summaries of imputed values for each imputation.

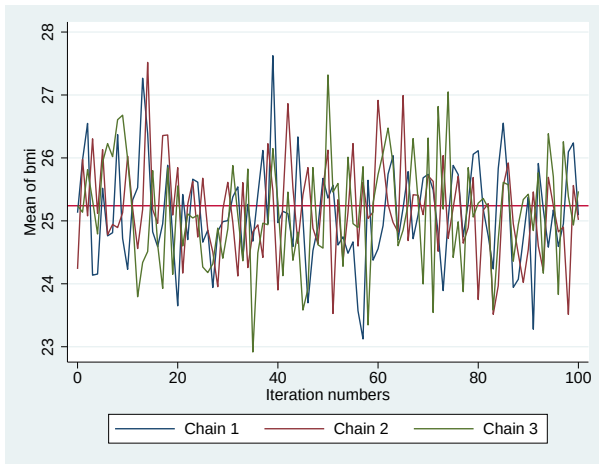
```
. use https://www.stata-press.com/data/r17/mheart8s0
(Fictional heart attack data; arbitrary pattern)
. quietly mi impute chained (pmm, knn(3)) bmi (reg) age = attack smokes hsgrad
> female, add(3) burnin(100) savetrace(impstats, replace) rseed(1359)
```

The results are saved in a long form. If we want to overlay separate chains in one graph, we need to convert our data to a wide form first—one variable per chain. We use the `reshape` command for this (see [D] **reshape**):

```
. use impstats, clear
(Summaries of imputed values from -mi impute chained-)
. reshape wide *mean *sd, i(iter) j(m)
(j = 1 2 3)
Data                                Long   ->   Wide
-----------------------------------------------------------------------------
Number of observations               303   ->   101
Number of variables                    6   ->   13
j variable (3 values)                  m   ->   (dropped)
xij variables:
                                age_mean   ->   age_mean1 age_mean2 age_mean3
                                bmi_mean   ->   bmi_mean1 bmi_mean2 bmi_mean3
                                  age_sd   ->   age_sd1 age_sd2 age_sd3
                                  bmi_sd   ->   bmi_sd1 bmi_sd2 bmi_sd3
-----------------------------------------------------------------------------
```

We can now plot the three chains for, say, the mean of `bmi` using `tsline`:

```
. tsset iter
Time variable: iter, 0 to 100
        Delta: 1 unit
. tsline bmi_mean1 bmi_mean2 bmi_mean3, ytitle(Mean of bmi) yline(25.24)
> legend(rows(1) label(1 "Chain 1") label(2 "Chain 2") label(3 "Chain 3"))
```



There are no apparent trends in any of the chains. All three chains seem to oscillate around the observed mean estimate of `bmi` of 25.24, providing some evidence of convergence of the algorithm.

Example 3: Custom prediction equations

Continuing example 1, we believe that there is no association between bmi and hsgrad conditional on other predictors, so we want to use hsgrad to model only age and omit it from the model for bmi:

```
. use https://www.stata-press.com/data/r17/mheart8s0
(Fictional heart attack data; arbitrary pattern)
. mi impute chained
>   (pmm, knn(3) omit(hsgrad)) bmi
>   (regress)                  age
>                                  = attack smokes hsgrad female, add(5)
Conditional models:
               age: regress age bmi attack smokes hsgrad female
               bmi: pmm bmi age attack smokes female, knn(3)
Performing chained iterations ...
Multivariate imputation                      Imputations =        5
Chained equations                                  added =        5
Imputed: m=1 through m=5                         updated =        0
Initialization: monotone                      Iterations =       50
                                                 burn-in =       10
               bmi: predictive mean matching
               age: linear regression
```

Variable	Observations per *m*			
	Complete	Incomplete	Imputed	Total
bmi	126	28	28	154
age	142	12	12	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

All right-hand-side complete predictors (attack, smokes, and female) are used in both prediction equations. The prediction equation for age additionally includes the hsgrad variable.

Example 4: Imputing variables of different types

We now consider an `mi set` version of the heart attack data containing an indicator for smoking high-tar cigarettes (variable `hightar`):

```
. use https://www.stata-press.com/data/r17/mheart9s0, clear
(Fictional heart attack data; bmi, age, and hightar missing; arbitrary pattern)
. mi describe
Style: mlong
       last mi update 22dec2020 10:41:23, 51 days ago
Observations:
   Complete            98
   Incomplete          56  (M = 0 imputations)
   ----------------------
   Total              154
Variables:
   Imputed: 3; bmi(24) age(30) hightar(12)
   Passive: 0
   Regular: 4; attack smokes female hsgrad
   System:  3; _mi_m _mi_id _mi_miss
   (there are no unregistered variables)
. mi misstable nested
     1.  hightar(12)
     2.  bmi(24)
     3.  age(30)
```

According to `mi describe`, there are no imputations, three registered imputed variables (`age`, `bmi`, and `hightar`), and four registered regular variables. `mi misstable nested` reports that missing values of the three imputation variables are not nested.

The `hightar` variable is a binary variable, so we choose the logistic method to impute its values (see [MI] **mi impute logit**). Because `hightar` records whether a subject smokes high-tar cigarettes, we use only those who smoke to impute its missing values. As such, including `smokes` as a predictor of `hightar` is redundant, so we omit this variable from the prediction equation for `hightar`:

```
. mi impute chained
>   (pmm, knn(3) omit(hsgrad))           bmi
>   (regress)                            age
>   (logit if smokes==1, omit(smokes)) hightar
>                                          = attack smokes hsgrad female, add(5)
Conditional models:
           hightar: logit hightar bmi age attack hsgrad female if smokes==1
               bmi: pmm bmi i.hightar age attack smokes female, knn(3)
               age: regress age i.hightar bmi attack smokes hsgrad female
Performing chained iterations ...
Multivariate imputation                     Imputations =        5
Chained equations                                 added =        5
Imputed: m=1 through m=5                        updated =        0
Initialization: monotone                     Iterations =       50
                                                burn-in =       10
               bmi: predictive mean matching
               age: linear regression
           hightar: logistic regression
```

	Observations per *m*			
Variable	Complete	Incomplete	Imputed	Total
bmi	130	24	24	154
age	124	30	30	154
hightar	52	12	12	64

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

From the output, we see that all incomplete values of each of the variables are imputed in all imputations. Because we restricted the imputation sample of `hightar` to smokers, the total number of observations reported for `hightar` is 64 and not 154. `mi impute chained` also automatically included the binary variable `hightar` as a factor variable in prediction equations for `age` and `bmi` because we used `logit` to impute it.

As we described in *Conditional imputation*, you should be careful when using an `if` statement for imputing variables conditionally on other variables. It was safe to use `if` here, because `smokes` did not contain missing values and there were no missing values of `hightar` for the subjects who do not smoke.

◁

Example 5: Conditional imputation

Continuing example 4, suppose now that the smokes variable also contains missing values:

```
. use https://www.stata-press.com/data/r17/mheart10s0, clear
(Fict. heart attack data; bmi, age, hightar, & smokes missing; arbitrary pattern)
. mi describe
Style: mlong
       last mi update 22dec2020 10:41:23, 51 days ago
Observations:
   Complete            92
   Incomplete          62  (M = 0 imputations)
   ----------------------
   Total              154
Variables:
   Imputed: 4; bmi(24) age(30) hightar(19) smokes(14)
   Passive: 0
   Regular: 3; attack female hsgrad
   System:  3; _mi_m _mi_id _mi_miss
   (there are no unregistered variables)
. mi misstable nested
     1.  smokes(14) -> hightar(19)
     2.  bmi(24)
     3.  age(30)
```

The smokes variable is now registered as imputed and the three regular variables are now attack, female, and hsgrad. mi misstable nested reports that although the missing-data pattern with respect to all four imputation variables is not monotone, the missing-data pattern with respect to smokes and hightar is monotone. Recall from *Conditional imputation* that one of the requirements of conditional imputation is that missing values of all conditioning variables (smokes) are nested within missing values of the conditional variable (hightar). So this requirement is satisfied in our data.

Because smokes contains missing values, we cannot use an if condition to restrict the imputation sample of hightar to those who smoke. We must use the conditional() option. We use the logistic method (see [MI] **mi impute logit**) to fill in missing values of smokes.

```
. mi impute chained
>   (pmm, knn(3) omit(hsgrad))                bmi
>   (regress)                                 age
>   (logit, cond(if smokes==1) omit(i.smokes)) hightar
>   (logit)                                   smokes
>                                            = attack hsgrad female, add(5)
Conditional models:
            smokes: logit smokes bmi age attack hsgrad female
           hightar: logit hightar bmi age attack hsgrad female,
                      cond(if smokes==1)
               bmi: pmm bmi i.smokes i.hightar age attack female, knn(3)
               age: regress age i.smokes i.hightar bmi attack hsgrad female
Performing chained iterations ...
Multivariate imputation                     Imputations =        5
Chained equations                                 added =        5
Imputed: m=1 through m=5                        updated =        0
Initialization: monotone                     Iterations =       50
                                                burn-in =       10
Conditional imputation:
  hightar: incomplete out-of-sample obs replaced with value 0
               bmi: predictive mean matching
               age: linear regression
           hightar: logistic regression
            smokes: logistic regression
```

	Observations per *m*			
Variable	Complete	Incomplete	Imputed	Total
bmi	130	24	24	154
age	124	30	30	154
hightar	135	19	19	154
smokes	140	14	14	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

With conditional imputation, a legend appears before the imputation table, reporting the conditional constant, the value that was used to replace all incomplete values of an imputation variable outside the conditional sample. The missing values of `hightar` in that sample were replaced with 0.

The `smokes` variable is imputed using `logit` and thus is included in prediction equations as a factor variable, `i.smokes`. As such, we specified `omit(i.smokes)` to omit `smokes` from the prediction equation for `hightar`.

Also notice that compared with imputation on a restricted subsample using an `if` condition, the reported total number of observations in the imputation sample for `hightar` is still 154. All incomplete observations, within and outside the conditional sample, are included in the imputation sample during conditional imputation. So the reported numbers of complete, incomplete, and imputed observations correspond with observations within and outside the conditional sample.

◁

▷ Example 6: Including expressions of imputation variables

In *Model building* of [MI] **mi impute**, we described two ways of accommodating functional relationships during imputation. Here we demonstrate a passive imputation approach that includes expressions of imputation variables directly into the imputation model.

Continuing example 5, suppose we assume that the conditional distribution of bmi exhibits some curvature with respect to age. We want to include age^2 in the prediction equation for bmi. If the relationship between bmi and age is indeed curvilinear, it would be unreasonable to assume that the conditional distribution of age given bmi is linear. One possibility is to determine what the relationship is between age and bmi given other predictors in the observed data (see, for example, [R] **mfp**) and include the appropriate functional terms of bmi in the prediction equation for age. Following White, Royston, and Wood (2011) to relax the linearity assumption, we use predictive mean matching with, say, five nearest neighbors instead of linear regression to impute age:

```
. mi impute chained
>   (pmm, knn(3) omit(hsgrad) incl((age^2)))   bmi
>   (pmm, knn(5))                              age
>   (logit, cond(if smokes==1) omit(i.smokes)) hightar
>   (logit)                                    smokes
>                                        = attack hsgrad female, replace
Conditional models:
            smokes: logit smokes bmi age attack hsgrad female
           hightar: logit hightar bmi age attack hsgrad female,
                      cond(if smokes==1)
               bmi: pmm bmi i.smokes i.hightar age (age^2)
                      attack female, knn(3)
               age: pmm age i.smokes i.hightar bmi attack hsgrad
                      female, knn(5)
Performing chained iterations ...
Multivariate imputation                     Imputations =        5
Chained equations                                 added =        0
Imputed: m=1 through m=5                        updated =        5
Initialization: monotone                     Iterations =       50
                                                burn-in =       10
Conditional imputation:
  hightar: incomplete out-of-sample obs replaced with value 0
               bmi: predictive mean matching
               age: predictive mean matching
           hightar: logistic regression
            smokes: logistic regression
```

	Observations per *m*			
Variable	Complete	Incomplete	Imputed	Total
bmi	130	24	24	154
age	124	30	30	154
hightar	135	19	19	154
smokes	140	14	14	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

We included the expression term in parentheses in the include() option in the prediction equation for bmi.

◁

▷ Example 7: Imputing on subsamples

Suppose that in our primary logistic analysis of heart attacks, we are planning to investigate various interaction effects with respect to gender. The female variable is complete, so the best way to accommodate such interactions is to use the by() option to perform imputation separately for females and males.

We continue example 3. Before imputing missing values, let's review our conditional specifications for each group. We can use the `dryrun` option to see univariate conditional models that will be used during imputation without actually imputing data:

```
. use https://www.stata-press.com/data/r17/mheart8s0, clear
(Fictional heart attack data; arbitrary pattern)

. mi impute chained
>   (pmm, knn(3) omit(hsgrad)) bmi
>   (regress)                  age
>                                  = attack smokes hsgrad, by(female) dryrun

Performing setup for each by() group:

-> female = 0
Conditional models:
               age: regress age bmi attack smokes hsgrad
               bmi: pmm bmi age attack smokes, knn(3)

-> female = 1
Conditional models:
               age: regress age bmi attack smokes hsgrad
               bmi: pmm bmi age attack smokes, knn(3)
```

Conditional specifications are as we expected, so we can proceed to imputation.

```
. mi impute chained
>   (pmm, knn(3) omit(hsgrad)) bmi
>   (regress)                  age
>                                  = attack smokes hsgrad
>                                              , add(5) by(female, noreport) dots

-> female = 0
Performing chained iterations:
  imputing m=1 through m=5 ..... done

-> female = 1
Performing chained iterations:
  imputing m=1 through m=5 ..... done

Multivariate imputation                      Imputations =        5
Chained equations                                  added =        5
Imputed: m=1 through m=5                         updated =        0

Initialization: monotone                      Iterations =       50
                                                 burn-in =       10

               bmi: predictive mean matching
               age: linear regression
```

by() Variable	Observations per *m*			
	Complete	Incomplete	Imputed	Total
female = 0				
bmi	95	21	21	116
age	106	10	10	116
female = 1				
bmi	31	7	7	38
age	36	2	2	38
Overall				
bmi	126	28	28	154
age	142	12	12	154

(Complete + Incomplete = Total; Imputed is the minimum across *m* of the number of filled-in observations.)

To avoid longer output, we specified the `noreport` option within `by()` to suppress information about the setup and imputation steps that otherwise would have been reported for each group.

◁

Stored results

`mi impute chained` stores the following in `r()`:

Scalars

`r(M)`	total number of imputations
`r(M_add)`	number of added imputations
`r(M_update)`	number of updated imputations
`r(k_ivars)`	number of imputed variables
`r(burnin)`	number of burn-in iterations
`r(N_g)`	number of imputed groups (`1` if `by()` is not specified)

Macros

`r(method)`	name of imputation method (`chained`)
`r(ivars)`	names of imputation variables
`r(uvmethods)`	names of univariate imputation methods
`r(init)`	type of initialization
`r(rngstate)`	random-number state used
`r(by)`	names of variables specified within `by()`

Matrices

`r(N)`	number of observations in imputation sample in each group (per variable)
`r(N_complete)`	number of complete observations in imputation sample in each group (per variable)
`r(N_incomplete)`	number of incomplete observations in imputation sample in each group (per variable)
`r(N_imputed)`	number of imputed observations in imputation sample in each group (per variable)

Methods and formulas

Let $X_1, X_2, \ldots, X_p$ denote imputation variables ordered from the most observed to the least observed and let $\mathbf{Z}$ denote the set of complete independent variables. (If $X_1, X_2, \ldots, X_p$ are monotone-missing and neither `nomonotone` nor `nomonotonechk` is used, then `mi impute chained` uses monotone imputation; see *Methods and formulas* of [MI] **mi impute monotone** for details.)

With the default specification of prediction equations, the chained-equation algorithm proceeds as follows. First, at iteration $t = 0$, missing values are initialized using monotone imputation. That is, missing values of $X_i^{(0)}$, $i = 1, \ldots, p$, are simulated from conditional densities of the form

$$f_i(X_i|X_1^{(0)}, X_2^{(0)}, \ldots, X_{i-1}^{(0)}, \mathbf{Z}, \boldsymbol{\theta}_i) \tag{2}$$

where the conditional density $f_i(\cdot)$ is determined according to the chosen univariate imputation method and $\boldsymbol{\theta}_i$ is its corresponding set of parameters with uniform prior; see *Methods and formulas* of chosen univariate imputation methods for details.

At iteration t, missing values of X_i for all $i = 1, \ldots, p$ are simulated from full conditionals, conditional densities of the form:

$$g_i(X_i|X_1^{(t)}, X_2^{(t)}, \ldots, X_{i-1}^{(t)}, X_{i+1}^{(t-1)}, \ldots, X_p^{(t-1)}, \mathbf{Z}, \boldsymbol{\phi}_i) \tag{3}$$

where again the conditional density $g_i(\cdot)$ is determined according to the chosen univariate imputation method and $\boldsymbol{\phi}_i$ is its corresponding set of parameters with uniform prior.

The algorithm iterates for a prespecified number of iterations b, $t = 1, \ldots, b$, and a final set of imputed values is obtained from the last iteration. At each iteration, the imputation process consists of steps 1–3 described in *Methods and formulas* of each respective univariate imputation method's manual entry.

Each imputation is obtained independently by repeating (2) and (3).

Conditional specifications in (2) and (3) correspond to the default specification of prediction equations. With the custom specification, the sets of complete predictors $\mathbf{Z} = \mathbf{Z}_i$ and imputation variables may differ across univariate specifications, and prediction equations may additionally include functions of imputation variables.

In summary, `mi impute chained` follows the steps below to fill in missing values in $X_1, \ldots, X_p$:

1. `mi impute chained` first builds appropriate univariate imputation models using the supplied information about imputation methods, imputation variables $\mathbf{X}$, and complete predictors $\mathbf{Z}$. By default, fully conditional specification of prediction equations is used. The order in which imputation variables are listed is ignored unless the `orderasis` option is used. By default, `mi impute chained` imputes variables in order from the most observed to the least observed.
2. Initialize missing values at $t = 0$ using monotone imputation (2).
3. Perform the iterative procedure (3) for $t = 1, \ldots, b$, for the length of the burn-in period, to obtain imputed values. At each iteration t,
    3.1. Fit a univariate model for X_i to the observed data to obtain the estimates of ϕ_i. See step 1 in *Methods and formulas* of each respective univariate imputation method's manual entry for details.
    3.2. Fill in missing values of X_i according to the specified imputation model. See step 2 and step 3 in *Methods and formulas* of each respective univariate imputation method's manual entry for details.
    3.3. Repeat steps 3.1 and 3.2 for each imputation variable X_i, $i = 1, \ldots, p$.
4. Repeat steps 2 and 3 to obtain M multiple imputations.

The iterative procedure (3) may not always correspond to a genuine simulation of imputed values from their predictive distribution $f(\mathbf{X}_m|\mathbf{X}_o, \mathbf{Z})$ because the set of full conditionals $\{g_i : i = 1, 2, \ldots, p\}$ may not correspond to this distribution or, in fact, to any proper multivariate distribution. The extent to which this is a problem in practical applications is still an open research problem. Some limited simulation studies reported only minimal effect of such incompatibility on final MI estimates (for example, van Buuren et al. [2006]).

Acknowledgments

The `mi impute chained` command was inspired by the community-contributed command `ice` by Patrick Royston of the MRC Clinical Trials Unit, London, and coauthor of the Stata Press book *Flexible Parametric Survival Analysis Using Stata: Beyond the Cox Model*; and Ian White of the MRC Biostatistics Unit, London. We are indebted to them for their extensive work in the multiple-imputation area in Stata. We are also grateful to them for their comments and advice on `mi impute chained`.

References

Andersen, A., and A. Rieckmann. 2016. Using mi impute chained to fit ANCOVA models in randomized trials with censored dependent and independent variables. *Stata Journal* 16: 650–661.

Arnold, B. C., E. Castillo, and J. M. Sarabia. 1999. *Conditional Specification of Statistical Models*. New York: Springer.

——. 2001. Conditionally specified distributions: An introduction. *Statistical Science* 16: 249–274. https://doi.org/10.1214/ss/1009213728.

Gelfand, A. E., and A. F. M. Smith. 1990. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association* 85: 398–409. https://doi.org/10.1080/01621459.1990.10476213.

Geman, S., and D. Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6: 721–741. https://doi.org/10.1109/TPAMI.1984.4767596.

Raghunathan, T. E., J. M. Lepkowski, J. Van Hoewyk, and P. Solenberger. 2001. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey Methodology* 27: 85–95.

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.

——. 2009. Multiple imputation of missing values: Further update of ice, with an emphasis on categorical variables. *Stata Journal* 9: 466–477.

van Buuren, S. 2007. Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research* 16: 219–242. https://doi.org/10.1177/0962280206074463.

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. https://doi.org/10.1002/(SICI)1097-0258(19990330)18:6⟨681::AID-SIM71⟩3.0.CO;2-R.

van Buuren, S., J. P. L. Brand, C. G. M. Groothuis-Oudshoorn, and D. B. Rubin. 2006. Fully conditional specification in multivariate imputation. *Journal of Statistical Computation and Simulation* 76: 1049–1064. https://doi.org/10.1080/10629360600810434.

Welch, C., J. W. Bartlett, and I. Petersen. 2014. Application of multiple imputation using the two-fold fully conditional specification algorithm in longitudinal clinical data. *Stata Journal* 14: 418–431.

White, I. R., R. M. Daniel, and P. Royston. 2010. Avoiding bias due to perfect prediction in multiple imputation of incomplete categorical data. *Computational Statistics & Data Analysis* 54: 2267–2275. https://doi.org/10.1016/j.csda.2010.04.005.

White, I. R., P. Royston, and A. M. Wood. 2011. Multiple imputation using chained equations: Issues and guidance for practice. *Statistics in Medicine* 30: 377–399. https://doi.org/10.1002/sim.4067.

Also see

[MI] **mi impute** — Impute missing values

[MI] **mi impute monotone** — Impute missing values in monotone data

[MI] **mi impute mvn** — Impute using multivariate normal regression

[MI] **mi estimate** — Estimation using multiple imputations

[MI] **Intro** — Introduction to mi

[MI] **Intro substantive** — Introduction to multiple-imputation analysis

[MI] **Glossary**