Title

mi impute mvn — Impute using multivariate normal regression

Description

`mi impute mvn` fills in missing values of one or more continuous variables using multivariate normal regression. It accommodates arbitrary missing-value patterns. You can perform separate imputations on different subsets of the data by specifying the `by()` option. `mi impute mvn` uses an iterative Markov chain Monte Carlo (MCMC) method to impute missing values. See *Remarks and examples* for details.

Menu

Statistics > Multiple imputation

Syntax

`mi impute mvn` *ivars* [`=` *indepvars*] [*if*] [`,` *impute_options options*]

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
Main	
noconstant	suppress constant term
MCMC options	
burnin(#)	specify number of iterations for the burn-in period; default is burnin(100)
burnbetween(#)	specify number of iterations between imputations; default is burnbetween(100)
prior(*prior_spec*)	specify a prior distribution; default is prior(uniform)
mcmconly	perform MCMC for the length of the burn-in period without imputing missing values
initmcmc(*init_mcmc*)	specify initial values for the MCMC procedure; default is initmcmc(em) using the EM estimates for initial values
wlfwgt(*matname*)	specify weights for the worst linear function
savewlf(*filename*[, ...])	save the worst linear function from each iteration in *filename*.dta
saveptrace(*fname*[, ...])	save MCMC parameter estimates from each iteration in *fname*.stptrace; see [MI] **mi ptrace**
Reporting	
emlog	display iteration log from EM
emoutput	display intermediate output from EM estimation
mcmcdots	display dots as MCMC iterations are performed
alldots	display dots as intermediate iterations are performed
nolog	do not display information about the EM or MCMC procedures
Advanced	
emonly[(*em_options*)]	perform EM estimation only

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

prior_spec	Description
uniform	use the uniform prior distribution; the default
jeffreys	use the Jeffreys noninformative prior distribution
ridge, df(#)	use a ridge prior distribution with degrees of freedom #

init_mcmc	Description
em[, *em_options*]	use EM to obtain starting values for MCMC; the default
initmatlist	supply matrices containing initial values for MCMC

em_options	Description
iterate(#)	specify the maximum number of iterations; default is iterate(100)
tolerance(#)	specify tolerance for the changes in parameter estimates; default is tolerance(1e-5)
init(*init_em*)	specify initial values for the EM algorithm; default is init(ac)
nolog	do not show EM iteration log
saveptrace(*fname*[, ...])	save EM parameter estimates from each iteration in *fname*.stptrace; see [MI] **mi ptrace**

init_em	Description
ac	use all available cases to obtain initial values for EM; the default
cc	use only complete cases to obtain initial values for EM
initmatlist	supply matrices containing initial values for EM

initmatlist is of the form *initmat* [*initmat* [...]]

initmat	Description
betas(# | *matname*)	specify coefficient vector; default is betas(0)
sds(# | *matname*)	specify standard deviation vector; default is sds(1)
vars(# | *matname*)	specify variance vector; default is vars(1)
corr(# | *matname*)	specify correlation matrix; default is corr(0)
cov(*matname*)	specify covariance matrix

In the above, # is understood to mean a vector containing all elements equal to #.

Options

Main

noconstant; see [R] **Estimation options**.

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

MCMC options

burnin(#) specifies the number of iterations for the initial burn-in period. The default is burnin(100). This option specifies the number of iterations necessary for the MCMC 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 the MCMC method* and examples 2 and 4. The provided default may be sufficient in many cases, but you are responsible for determining that sufficient iterations are performed.

burnbetween(#) specifies a number of iterations of the MCMC to perform between imputations, the purpose being to reduce correlation between sets of imputed values. The default is burnbetween(100). As with burnin(), you are responsible for determining that sufficient iterations are performed. See *Convergence of the MCMC method* and examples 2 and 4.

`prior(`*prior_spec*`)` specifies a prior distribution to be used by the MCMC procedure. The default is `prior(uniform)`. The alternative prior distributions are useful when the default estimation of the parameters using maximum likelihood becomes unstable (for example, estimates on the boundary of the parameter space) and introducing some prior information about parameters stabilizes the estimation.

prior_spec is

`uniform` | `jeffreys` | `ridge, df(#)`

`uniform` specifies the uniform (flat) prior distribution. Under this prior distribution, the posterior distribution is proportional to the likelihood function and thus the estimate of the posterior mode is the same as the maximum likelihood (ML) estimate.

`jeffreys` specifies the Jeffreys, noninformative prior distribution. This prior distribution can be used when there is no strong prior knowledge about the model parameters.

`ridge, df(#)` specifies a ridge, informative prior distribution with the degrees of freedom #. This prior introduces some information about the covariance matrix by smoothing the off-diagonal elements (correlations) toward zero. The degrees of freedom, `df()`, which may be noninteger, regulates the amount of smoothness—the larger this number, the closer the correlations are to zero. A ridge prior is useful to stabilize inferences about the mean parameters when the covariance matrix is poorly estimated, for example, when there are insufficient observations to estimate correlations between some variables reliably because of missing data, causing the estimated covariance matrix to become non–positive definite (see Schafer [1997, 155–157] for details).

`mcmconly` specifies that `mi impute mvn` run the MCMC for the length of the burn-in period and then stop. This option is useful in combination with `savewlf()` or `saveptrace()` to examine the convergence of the MCMC prior to imputation. No imputation is performed when `mcmconly` is specified, so `add()` or `replace` is not required with `mi impute mvn, mcmconly`, and they are ignored if specified. The `mcmconly` option is not allowed with `emonly`.

`initmcmc()` may be specified as `initmcmc(em` [`,` *em_options*]`)` or `initmcmc(`*initmatlist*`)`.

`initmcmc()` specifies initial values for the regression coefficients and covariance matrix of the multivariate normal distribution to be used by the MCMC procedure. By default, initial values are obtained from the EM algorithm, `initmcmc(em)`.

`initmcmc(em`[`,` *em_options*]`)` specifies that the initial values for the MCMC procedure be obtained from EM. You can control the EM estimation by specifying *em_options*. If the uniform prior is used, the initial estimates correspond to the ML estimates computed using EM. Otherwise, the initial values are the estimates of the posterior mode computed using EM.

em_options are

`iterate(#)` specifies the maximum number of EM iterations to perform. The default is `iterate(100)`.

`tolerance(#)` specifies the convergence tolerance for the EM algorithm. The default is `tolerance(1e-5)`. Convergence is declared once the maximum of the relative changes between two successive estimates of all model parameters is less than #.

`init()` may be specified as `init(ac)`, `init(cc)`, or `init(`*matlist*`)`

`init()` specifies initial values for the regression coefficients and covariance matrix of the multivariate normal distribution to be used by the EM algorithm. `init(ac)` is the default.

`init(ac)` specifies that initial estimates be obtained using all available cases. The initial values for regression coefficients are obtained from separate univariate regressions of

each imputation variable on the independent variables. The corresponding estimates of the residual mean-squared error are used as the initial values for the diagonal entries of the covariance matrix (variances). The off-diagonal entries (correlations) are set to zero.

`init(cc)` specifies that initial estimates be obtained using only complete cases. The initial values for regression coefficients and the covariance matrix are obtained from a multivariate regression fit to the complete cases only.

`init(`*initmatlist*`)` specifies to use manually supplied initial values for the EM procedure and syntactically is identical to `mcmcinit(`*initmatlist*`)`, described below, except that you specify `init(`*initmatlist*`)`.

`nolog` suppresses the EM iteration log when `emonly` or `emoutput` is used.

`saveptrace(`*fname*`[ , replace ])` specifies to save the parameter trace log from the EM algorithm to a file called *fname*`.stptrace`. If the file already exists, the `replace` suboption specifies to overwrite the existing file. See [MI] **mi ptrace** for details about the saved file and how to read it into Stata.

`initmcmc(`*initmatlist*`)`, where *initmatlist* is

initmat [*initmat* [...]]

specifies manually supplied initial values for the MCMC procedure.

initmat is

`betas(#|`*matname*`)` specifies initial values for the regression coefficients. The default is `betas(0)`, implying a value of zero for all regression coefficients. If you specify `betas(#)`, then # will be used as the initial value for all regression coefficients. Alternatively, you can specify the name of a Stata matrix, *matname*, containing values for each regression coefficient. *matname* must be conformable with the dimensionality of the specified model. That is, it can be one of the following dimensions: $p \times q$, $q \times p$, $1 \times pq$, or $pq \times 1$, where p is the number of imputation variables and q is the number of independent variables.

`sds(#|`*matname*`)` specifies initial values for the standard deviations (square roots of the diagonal elements of the covariance matrix). The default is `sds(1)`, which sets all standard deviations and thus variances to one. If you specify `sds(#)`, then the squared # will be used as the initial value for all variances. Alternatively, you can specify the name of a Stata matrix, *matname*, containing individual values. *matname* must be conformable with the dimensionality of the specified model. That is, it can be one of the following dimensions: $1 \times p$ or $p \times 1$, where p is the number of imputation variables. This option cannot be combined with `cov()` or `vars()`. The `sds()` option can be used in combination with `corr()` to provide initial values for the covariance matrix.

`vars(#|`*matname*`)` specifies initial values for variances (diagonal elements of the covariance matrix). The default is `vars(1)`, which sets all variances to one. If you specify `vars(#)`, then # will be used as the initial value for all variances. Alternatively, you can specify the name of a Stata matrix, *matname*, containing individual values. *matname* must be conformable with the dimensionality of the specified model. That is, it can be one of the following dimensions: $1 \times p$ or $p \times 1$, where p is the number of imputation variables. This option cannot be combined with `cov()` or `sds()`. The `vars()` option can be used in combination with `corr()` to provide initial values for the covariance matrix.

`corr(#|`*matname*`)` specifies initial values for the correlations (off-diagonal elements of the correlation matrix). The default is `corr(0)`, which sets all correlations and, thus, covariances to zero. If you specify `corr(#)`, then all correlation coefficients will be set to #. Alternatively, you can specify the name of a Stata matrix, *matname*, containing individual values. *matname*

can be a square $p \times p$ matrix with diagonal elements equal to one or it can contain the corresponding lower (upper) triangular matrix in a vector of dimension $p(p+1)/2$, where p is the number of imputation variables. This option cannot be combined with `cov()`. The `corr()` option can be used in combination with `sds()` or `vars()` to provide initial values for the covariance matrix.

`cov(`*matname*`)` specifies initial values for the covariance matrix. *matname* must contain the name of a Stata matrix. *matname* can be a square $p \times p$ matrix or it can contain the corresponding lower (upper) triangular matrix in a vector of dimension $p(p+1)/2$, where p is the number of imputation variables. This option cannot be combined with `corr()`, `sds()`, or `vars()`.

`wlfwgt(`*matname*`)` specifies the weights (coefficients) to use when computing the worst linear function (WLF). The coefficients must be saved in a Stata matrix, *matname*, of dimension $1 \times d$, where $d = pq + p(p+1)/2$, p is the number of imputation variables, and q is the number of predictors. This option is useful when initial values from the EM estimation are supplied to data augmentation (DA) as matrices. This option can also be used to obtain the estimates of linear functions other than the default WLF. This option cannot be combined with `by()`.

`savewlf(`*filename*`[ , replace ])` specifies to save the estimates of the WLF from each iteration of MCMC to a Stata dataset called *filename*`.dta`. If the file already exists, the `replace` suboption specifies to overwrite the existing file. This option is useful for monitoring convergence of the MCMC. `savewlf()` is allowed with `initmcmc(em)`, when the initial values are obtained using the EM estimation, or with `wlfwgt()`. This option cannot be combined with `by()`.

`saveptrace(`*fname*`[ , replace ])` specifies to save the parameter trace log from the MCMC to a file called *fname*`.stptrace`. If the file already exists, the `replace` suboption specifies to overwrite the existing file. See [MI] **mi ptrace** for details about the saved file and how to read it into Stata. This option is useful for monitoring convergence of the MCMC. This option cannot be combined with `by()`.

Reporting

`dots`, `noisily`, `nolegend`; see [MI] **mi impute**. Also, `noisily` is a synonym for `emoutput`. `nolegend` suppresses group legends that may appear when the `by()` option is used. It is a synonym for `by(, nolegend)`.

`emlog` specifies that the EM iteration log be shown. The EM iteration log is not displayed unless `emonly` or `emoutput` is specified.

`emoutput` specifies that the EM output be shown. This option is implied with `emonly`.

`mcmcdots` specifies to display all MCMC iterations as dots.

`alldots` specifies to display all intermediate iterations as dots in addition to the imputation dots. These iterations include the EM iterations and the MCMC burn-in iterations. This option implies `mcmcdots`.

`nolog` suppresses all output from EM or MCMC that is usually displayed by default.

Advanced

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

`emonly[ (`*em_options*`) ]` specifies that `mi impute mvn` perform EM estimation and then stop. You can control the EM process by specifying *em_options*. This option is useful at the preliminary stage to obtain insight about the length of the burn-in period as well as to choose a prior specification. No

imputation is performed, so `add()` or `replace` is not required with `mi impute mvn, emonly`, and they are ignored if specified. The `emonly` option is not allowed with `mcmconly`.

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`.

Incomplete continuous data with arbitrary pattern of missing values

As we described in detail in *Multivariate imputation* in [MI] **mi impute**, imputation of multiple variables with an arbitrary pattern of missing values is more challenging than when the missing-data pattern is monotone.

One approach for dealing with an arbitrary missing-value pattern is to assume an explicit tractable parametric model for the data and draw imputed values from the resulting distribution of the missing data given observed data. One of the more popular parametric models is the Gaussian normal model; see Rubin (1987) for other recommendations. Although a multivariate normal model is straightforward, difficulty arises in the simulation from the corresponding, more complicated, distribution of the missing data. One solution is to use one of the Bayesian iterative Markov chain Monte Carlo (MCMC) procedures to approximate the distribution of missing data.

Multivariate imputation using data augmentation

`mi impute mvn` uses data augmentation (DA) —an iterative MCMC procedure—to generate imputed values assuming an underlying multivariate normal model. For details about DA as a general MCMC procedure, see Gelman et al. (2014), Tanner and Wong (1987), and Li (1988), among others. For applications of DA to incomplete multivariate normal data, see, for example, Little and Rubin (2020) and Schafer (1997). Below we briefly describe the idea behind DA; see *Methods and formulas* for details.

Consider multivariate data $\mathbf{X} = (\mathbf{X}_o, \mathbf{X}_m)$, decomposed into the observed part $\mathbf{X}_o$ and the missing part $\mathbf{X}_m$, from a normal distribution $\Pr(\mathbf{X}|\boldsymbol{\theta}) = N(\boldsymbol{\beta}, \boldsymbol{\Sigma})$, where $\boldsymbol{\theta}$ denotes the unknown model parameters (regression coefficients $\boldsymbol{\beta}$ and unique elements of the covariance matrix $\boldsymbol{\Sigma}$). The goal is to replace missing values in $\mathbf{X}_m$ with draws from the distribution (or the predictive distribution in Bayesian terminology) of the missing data given observed data, $\Pr(\mathbf{X}_m|\mathbf{X}_o)$. The actual predictive distribution $\Pr(\mathbf{X}_m|\mathbf{X}_o)$ is difficult to draw from directly because of an underlying dependence on the posterior distribution of the unknown parameters $\boldsymbol{\theta}$, $\Pr(\boldsymbol{\theta}|\mathbf{X}_o)$.

Originally, DA was used to approximate the posterior distribution of the model parameters, $\Pr(\boldsymbol{\theta}|\mathbf{X}_o)$, in Bayesian applications with incomplete data. The idea of DA is to augment the observed data, $\mathbf{X}_o$, with the latent (unobserved) data, $\mathbf{X}_m$, such that the conditional posterior distribution $\Pr(\boldsymbol{\theta}|\mathbf{X}_o, \mathbf{X}_m)$ becomes more tractable and easier to simulate from. Then the procedure becomes as follows. For a current $\boldsymbol{\theta}^{(t)}$, draw $\mathbf{X}_m^{(t+1)}$ from its conditional predictive distribution given the observed data and $\boldsymbol{\theta}$, $\Pr(\mathbf{X}_m|\mathbf{X}_o, \boldsymbol{\theta}^{(t)})$. Next draw $\boldsymbol{\theta}^{(t+1)}$ from its conditional posterior distribution given the augmented data, $\Pr(\boldsymbol{\theta}|\mathbf{X}_o, \mathbf{X}_m^{(t+1)})$. Continue to iterate until the sequence $\{(\mathbf{X}_m^{(t)}, \boldsymbol{\theta}^{(t)}) : t = 1, 2, \ldots\}$, an MCMC sequence, converges to a stationary distribution $\Pr(\boldsymbol{\theta}, \mathbf{X}_m|\mathbf{X}_o)$. This way a complicated task of simulating from $\Pr(\boldsymbol{\theta}|\mathbf{X}_o)$ is replaced by a sequence of simpler simulation tasks of iteratively sampling from $\Pr(\boldsymbol{\theta}|\mathbf{X}_o, \mathbf{X}_m)$ and $\Pr(\mathbf{X}_m|\mathbf{X}_o, \boldsymbol{\theta})$. How is this procedure related to imputation? The sequence $\{\mathbf{X}_m^{(t)} : t = 1, 2, \ldots\}$ contains draws from an approximate predictive distribution $\Pr(\mathbf{X}_m|\mathbf{X}_o)$, and thus $\mathbf{X}_m^{(t)}$'s are, in fact, imputations. The convergence of this procedure was studied by Li (1988).

The functional forms of the conditional distributions $\Pr(\boldsymbol{\theta}|\mathbf{X}_o, \mathbf{X}_m)$ and $\Pr(\mathbf{X}_m|\mathbf{X}_o, \boldsymbol{\theta})$ are determined from the assumed distribution of the data, $\mathbf{X}$, and a prior distribution for the model parameters, $\boldsymbol{\theta}$, $\Pr(\boldsymbol{\theta})$. `mi impute mvn` assumes a normal distribution for the data and supports three prior distributions: uniform, Jeffreys, and ridge.

The prior distributions are categorized into noninformative (or also vague, diffuse, flat, reference) and informative prior distributions. The noninformative priors provide no extra information about model parameters beyond that already contained in the data. These priors are recommended when no strong prior knowledge is available about the parameters. Informative prior distributions are used when there is some a priori knowledge about the distribution of the parameters. For example, prior information about cancer mortality rates in a Poisson model can be assigned based on the available worldwide estimate. The uniform and Jeffreys priors are noninformative priors. The ridge prior is an informative prior.

The uniform prior assumes that all values of the parameters are equally probable. Under this prior specification, the posterior distribution of the parameters is equivalent to the likelihood function, and so the Bayesian and frequentist methods coincide. The Jeffreys prior is another widely used noninformative prior distribution, and with small samples, it may be preferable to the uniform prior. A ridge prior is often used when the estimated covariance matrix becomes singular (or nearly singular), as may occur with sparse missing data if there are not enough observations to estimate reliably all aspects of the covariance matrix. A ridge prior smooths the estimate of the covariance matrix toward a diagonal structure depending on the chosen degrees of freedom; the larger the degrees of freedom, the closer is the estimated covariance matrix to the diagonal matrix (see Schafer [1997, 155–157] for details).

Convergence of the MCMC method

For a brief overview of convergence of MCMC, see *Convergence of iterative methods* in [MI] **mi impute**.

The MCMC procedure DA is iterated until an MCMC sequence $\{(\mathbf{X}_m^{(t)}, \boldsymbol{\theta}^{(t)}) : t = 1, 2, \ldots\}$ converges to a stationary distribution. Unlike maximum likelihood, EM, or other optimization-based procedures, the DA procedure does not have a simple stopping rule that guarantees the convergence of the chain to a stationary distribution. Thus the question of how long to iterate to achieve convergence arises. In addition to determining convergence of MCMC, we must also investigate the serial dependence known to exist among the MCMC draws to obtain independent imputations.

Suppose that after an initial burn-in period, b, the sequence $\{(\mathbf{X}_m^{(b+t)}) : t = 1, 2, \ldots\}$ (imputations) can be regarded as an approximate sample from $\Pr(\mathbf{X}_m|\mathbf{X}_o)$. In general, this sample will not contain independent observations because the successive iterates of the MCMC tend to be correlated. To achieve independence among imputations, we can sample the chain. To do that, we need to determine the number of iterations, k, such that $\mathbf{X}_m^{(t)}$ and $\mathbf{X}_m^{(t+k)}$ are approximately independent. Then imputations can be obtained as the chain values of $\mathbf{X}_m$ from iterations $b, b+k, b+2k, \ldots, b+mk$, where m is the required number of imputations. In our definition, b is the number of iterations necessary for the chain to achieve stationarity and k is the number of iterations between imputations necessary to achieve independent values of the chain.

Before we proceed, we notice that from the properties of MCMC, the convergence of the chain $\{(\mathbf{X}_m^{(t)}, \boldsymbol{\theta}^{(t)}) : t = 1, 2, \ldots\}$ to $\Pr(\boldsymbol{\theta}, \mathbf{X}_m|\mathbf{X}_o)$ is equivalent to the convergence of $\{(\boldsymbol{\theta}^{(t)}) : t = 1, 2, \ldots\}$ to $\Pr(\boldsymbol{\theta}|\mathbf{X}_o)$ or, alternatively, of $\{(\mathbf{X}_m^{(t)}) : t = 1, 2, \ldots\}$ to $\Pr(\mathbf{X}_m|\mathbf{X}_o)$. Because the parameter series are usually of lower dimension, we examine convergence using the series of parameter estimates rather than the series of imputations.

How to determine convergence and, in particular, to choose values for b and k, has received much attention in the MCMC literature. In practice, convergence is often examined visually from the trace and autocorrelation plots of the estimated parameters. Trace plots are plots of estimated parameters against iteration numbers. Long-term trends in trace plots and high serial dependence in autocorrelation plots are indicative of a slow convergence to stationarity. A value of b can be inferred from a trace plot as the earliest iteration after which the chain does not exhibit a visible trend and the parameter series stabilize, which is to say the fluctuations in values become more regular. A value of k can be chosen from autocorrelation plots as the lag k for which autocorrelations of all parameters decrease to zero. When the initial values are close to the posterior mode, the initial number of iterations, b, and number of iterations between imputations, k, will be similar. When the initial values are far off in the tails of the posterior distribution, the initial number of iterations will generally be larger.

In practice, when the number of parameters in the model is large, it may not be feasible to monitor the convergence of all the individual series. One solution is to find a function of the parameters that would be the slowest to converge to stationarity. The convergence of the series for this function will then be indicative of the convergence of other functions and, in particular, individual parameter series. Schafer (1997, 129–131) suggests the worst linear function (WLF), the function corresponding to the linear combination of the parameter estimates where the coefficients are chosen such that this function has the highest asymptotic rate of missing information; see *Methods and formulas* for computational details. He found that when the observed-data posterior distribution is nearly normal, this function is among the slowest to approach stationarity. Thus we can determine b and k by monitoring the convergence of the WLF. When the observed-data posterior is not normal and some aspects of the model are poorly estimated, the WLF may not be the slowest to converge. In such cases, we recommend exploring convergence of other functions or of individual parameter series.

The number of iterations necessary for DA to converge depends on the rate of convergence of DA. The rate of convergence of DA mainly depends on the fractions of missing information and initial values. The higher the fractions of missing information and the farther the initial values are from the posterior mode, the slower the convergence, and thus the larger the number of iterations required. Initial values for the DA procedure can be obtained from the EM algorithm for incomplete data (for example, Dempster, Laird, and Rubin [1977]). In addition, the number of iterations necessary for the DA procedure to converge can be inferred based on the number of iterations that the EM algorithm took to converge (Schafer 1997).

The convergence of the chain and the required number of iterations can be also inferred by running multiple independent MCMC sequences using overdispersed initial values, that is, initial values from a distribution with greater variability than that of the posterior distribution (Gelman and Rubin 1992;

Schafer 1997, 126–128). Then the number of iterations can be taken to be the largest iteration number for which the series in all the chains stabilize.

Although the graphical summaries described above are useful in checking convergence, they must be used with caution. They can be deceptive in cases when the observed-data posterior has an odd shape or has multiple modes, which may happen with small sample sizes or sparse missing data. Examination of the data and missing-data patterns, as well as the behavior of the EM algorithm, are highly recommended when investigating the MCMC convergence. How one checks for convergence will be shown in examples 2 and 4.

Using mi impute mvn

`mi impute mvn` imputes missing data using DA, an iterative MCMC method, assuming the multivariate normal distribution for the data. For the discussion of options, such as `add()` and `replace`, common to all imputation methods, see [MI] **mi impute**. Here we focus on the options and functionality specific to `mi impute mvn`.

The two main options are `burnin()` (which specifies the number of iterations necessary for the MCMC to converge, b) and `burnbetween()` (which specifies the number of iterations between imputations, k). We discussed how to choose these values in the previous section. By default, these values are arbitrarily set to be 100 each.

You can choose from the three prior specifications. You can use `prior(uniform)` (the default) to specify the uniform prior, `prior(jeffreys)` to specify the Jeffreys prior, or `prior(ridge, df())` to specify a ridge prior. You must also choose the degrees of freedom with a ridge prior.

For initial values, `mi impute mvn` uses the estimates from the EM algorithm for incomplete data (`initmcmc(em)`). When the uniform prior distribution is used, the estimates obtained from EM are MLEs. Under other prior specifications, the estimates from EM correspond to the posterior mode of the respective posterior distribution of the model parameters. Using the estimates from EM as initial values in general accelerates the convergence of MCMC. To determine convergence, it may also be useful to try different sets of initial values. You can do this by creating Stata matrices containing the initial values and supplying them in the respective `initmcmc()` suboptions `betas()`, `cov()`, etc.

You can save the estimates of the WLF and parameter series from MCMC iterations by using the `savewlf()` and `saveptrace()` options. These options are useful when examining convergence of MCMC, as we will demonstrate in examples 2 and 4. You can use `mi impute mvn` to run the MCMC without imputing the data if you specify the `mcmconly` option. This option is useful in combination with `savewlf()` or `saveptrace()` when examining convergence of MCMC. When `mcmconly` is specified, the DA procedure is performed for the number of iterations as specified in `burnin()` and no imputations are performed.

You can also perform the EM estimation without MCMC iterations if you specify the `emonly()` option. This option is useful for detecting convergence problems prior to running MCMC. The number of iterations EM takes to converge can be used as an approximation for the burn-in period. Also, slow convergence of the EM algorithm can reveal problems with estimability of certain model parameters.

Examples

▷ Example 1: Monotone-missing data

Recall the heart attack example from *Multivariate imputation* in [MI] **mi impute**, where we used `mi impute mvn` to impute missing values for `age` and `bmi` that follow a monotone-missing pattern:

```
. use https://www.stata-press.com/data/r17/mheart5s0
(Fictional heart attack data)
. mi impute mvn age bmi = attack smokes hsgrad female, add(10)
Performing EM optimization:
note: 12 observations omitted from EM estimation because of all imputation
      variables missing.
  observed log likelihood = -651.75868 at iteration 7
Performing MCMC data augmentation ...
Multivariate imputation                     Imputations =       10
Multivariate normal regression                    added =       10
Imputed: m=1 through m=10                       updated =        0
Prior: uniform                               Iterations =     1000
                                                burn-in =      100
                                                between =      100
```

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

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

In the above, we omitted the `nolog` option that was present in the example in [MI] **mi impute**.

In addition to the output reported by all imputation methods, `mi impute mvn` also provides some specific information.

As we previously explained, `mi impute mvn` uses an iterative MCMC technique to impute missing values. The two phases of `mi impute mvn` are 1) obtaining initial values (unless supplied directly) and 2) performing the MCMC procedure from which imputations are obtained. These two phases are noted in the output header.

In this example, the initial values are obtained using the EM method (the default). We see from the output that EM converged in seven iterations. A note displayed thereafter reports that 12 observations contain missing values for both `bmi` and `age` and were omitted. The note is just explanatory and should not cause you concern. Those 12 observations would contribute nothing to the likelihood function even if they were included, although the algorithm would take longer to converge.

The estimates from EM are used as initial values for DA. The first part of the table header, containing the information about the method used and the number of imputations, was described in detail in [MI] **mi impute**. The second part of the table header is specific to `mi impute mvn`. From the output, a total of 1,000 iterations of MCMC are performed. The first 100 iterations (the default) are used for the burn-in period (`burn-in = 100`), the first imputation calculated from the last iteration; thereafter, each subsequent imputation is calculated after performing another 100 iterations. The default uniform prior is used for both the EM estimation and the MCMC procedure. Under this prior, the parameter estimates obtained are MLEs.

◁

▷ Example 2: Checking convergence of MCMC

In example 1, the monotone missingness of `age` and `bmi` as well as the quick convergence of EM suggest that the MCMC must converge rapidly. In fact, we know that under a monotone-missing pattern, no iterations are needed to obtain imputed values (see [MI] **mi impute monotone**). Let's examine the convergence of the MCMC procedure for the above heart attack data, the point being to see what quick convergence looks like.

As we discussed earlier, convergence is often assessed from the trace plots of the MCMC parameter estimates. Because of a possibly large number of estimated parameters, this approach may be tedious. Alternatively, we can plot the WLF for which the convergence is generally the slowest.

We use the `savewlf(wlf)` option to save estimates of the WLF to a Stata dataset called `wlf.dta`. To examine the convergence of MCMC, we do not need imputation, and so we use the `mcmconly` option to perform the MCMC procedure without subsequent imputation. We use a total of $1000 = 10 \times 100$ iterations (`burnin(1000)` option), corresponding to the length of the MCMC to obtain 10 imputations:

```
. mi impute mvn age bmi = attack smokes hsgrad female, mcmconly burnin(1000)
> rseed(2232) savewlf(wlf)

Performing EM optimization:
note: 12 observations omitted from EM estimation because of all imputation
      variables missing.
  observed log likelihood = -651.75868 at iteration 7

Performing MCMC data augmentation ...

Note: No imputation performed.
```

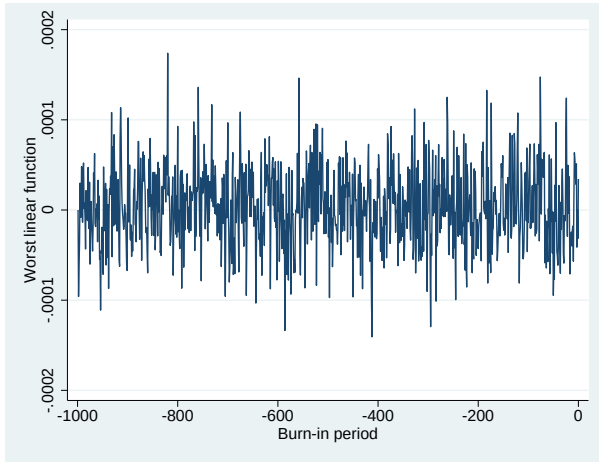
We also specified the `rseed(2232)` option so that we can reproduce our results.

The created dataset contains three variables: `iter`, `m`, and `wlf`. The `iter` variable records iterations (the burn-in iterations are recorded as negative integers). The `m` variable records imputation numbers to which the iteration sequence corresponds (`m` contains `0` if `mcmconly` is used). The `wlf` variable records the WLF estimates.

```
. use wlf, clear

. describe

Contains data from wlf.dta
 Observations:         1,000
    Variables:             3                  30 Apr 2021 22:50
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
iter            long    %12.0g
m               long    %12.0g
wlf             double  %10.0g
-------------------------------------------------------------------------------
Sorted by:
```

We use the time-series commands `tsline` and `ac` (see [TS] **tsline** and [TS] **corrgram**) to plot the estimates and autocorrelations of `wlf` with respect to the iteration number. We first use `tsset` to set `iter` as the "time" variable and then use `tsline` to obtain a trace plot:

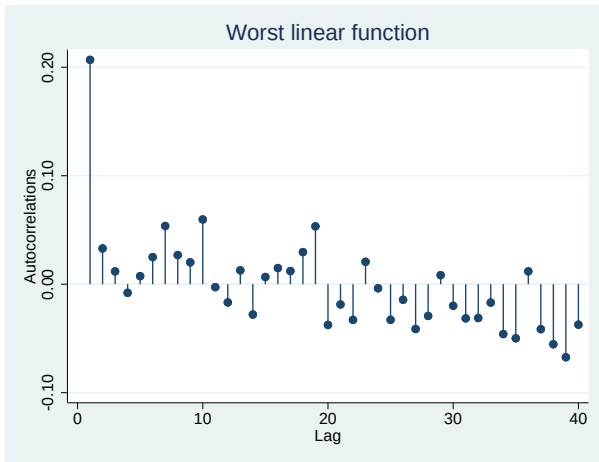
```
. tsset iter

Time variable: iter, -999 to 0
        Delta: 1 unit

. tsline wlf, ytitle(Worst linear function) xtitle(Burn-in period)
```



The graph shows no visible trend in the estimates of the WLF, just as we expected. Convergence of MCMC by the 100th iteration should be assured. In fact, taking into account the declared convergence of the EM algorithm in only seven iterations, we would be comfortable with using a much smaller burn-in period of, say, 10 iterations.

We next examine the autocorrelation in the WLF to obtain an idea of how many iterations to use between imputations to ensure their approximate independence:

```
. ac wlf, title(Worst linear function) ytitle(Autocorrelations)
> ciopts(astyle(none)) note("")
```



From the graphical output, the autocorrelations die off quickly. This suggests that we can use a smaller number, say, 10 or 20, rather than the default 100 iterations for the burn-between period.

◁

We considered an example with a monotone-missing pattern. `mi impute mvn` is designed to accommodate arbitrary missing-data patterns, so let's consider an example with them.

▷ Example 3: Arbitrary missing-data pattern

Consider data on house resale prices provided by the Albuquerque Board of Realtors and distributed by the Data and Story Library. You can find a detailed description of the data at http://www.pmean.com/00files/housing.htm.

```
. use https://www.stata-press.com/data/r17/mhouses1993
(Albuquerque home prices Feb15-Apr30, 1993)
. describe
Contains data from https://www.stata-press.com/data/r17/mhouses1993.dta
 Observations:           117                  Albuquerque home prices
                                                Feb15-Apr30, 1993
    Variables:             8                  19 Jun 2020 10:50
                                              (_dta has notes)
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
price           int     %8.0g                 Sale price (hundreds)
sqft            int     %8.0g                 Square footage of living space
age             byte    %10.0g                Home age (years)
nfeatures       byte    %8.0g                 Number of certain features
ne              byte    %8.0g                 Located in northeast (largest
                                                residential) sector of the city
custom          byte    %8.0g                 Custom build
corner          byte    %8.0g                 Corner location
tax             int     %10.0g                Tax amount (dollars)
-------------------------------------------------------------------------------
Sorted by:
```

The dataset includes eight variables. The primary variable of interest is `price`, and other variables are used as its predictors.

We investigate the missing-data patterns of these data using `misstable`:

```
. misstable pattern
   Missing-value patterns
     (1 means complete)

                |  Pattern
   Percent      |  1  2
----------------+-------
       56%      |  1  1
                |
       35       |  1  0
        7       |  0  0
        2       |  0  1
----------------+-------
      100%      |
  Variables are  (1) tax  (2) age
. misstable nested
     1.  tax(10)
     2.  age(49)
```

We see from the output only 56% of observations are complete; the remaining 44% contain missing values of `age` or `tax`. The `tax` variable contains 10 missing values, and the `age` variable contains 49 missing values. `misstable nested` reports that missing values of `age` and `tax` are not nested because there are two statements describing the missing-value pattern; see [R] **misstable** for details.

Let's use `mi impute mvn` to impute missing values of `age` and `tax`. Before we do that, a quick examination of the data revealed that the distribution for `age` and `tax` are somewhat skewed. As such, we choose to impute the variables on a log-transformed scale.

Following the steps as described in *Imputing transformations of incomplete variables* of [MI] **mi impute**, we create new variables containing the log values,

```
. generate lnage = ln(age)
(49 missing values generated)
. generate lntax = ln(tax)
(10 missing values generated)
```

and register them as imputed variables,

```
. mi set mlong
. mi register imputed lnage lntax
(51 m=0 obs now marked as incomplete)
. mi register regular price sqft nfeatures ne custom corner
```

We `mi set` our data as mlong and register the complete variables as regular. For the purpose of this analysis, we leave passive variables `age` and `tax` unregistered. (Note that all missing values of the created `lnage` and `lntax` variables are eligible for imputation; see [MI] **mi impute** for details.)

We now use `mi impute mvn` to impute values of `lnage` and `lntax`:

```
. mi impute mvn lnage lntax = price sqft nfeatures ne custom corner, add(20)
Performing EM optimization:
note: 8 observations omitted from EM estimation because of all imputation
      variables missing.
  observed log likelihood =   112.1464 at iteration 48
Performing MCMC data augmentation ...
Multivariate imputation                     Imputations =       20
Multivariate normal regression                    added =       20
Imputed: m=1 through m=20                       updated =        0
Prior: uniform                               Iterations =     2000
                                                burn-in =      100
                                                between =      100
```

	Observations per m			
Variable	Complete	Incomplete	Imputed	Total
lnage	68	49	49	117
lntax	107	10	10	117

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

◁

▷ Example 4: Checking convergence of MCMC

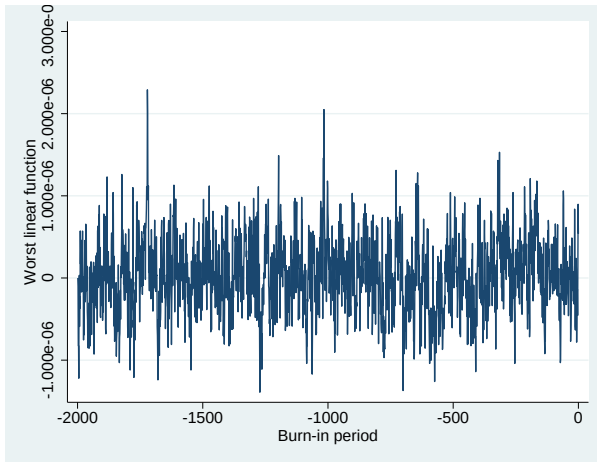
In the above example, we arbitrarily created 20 imputations. The output is similar to that of the earlier example. Here the EM algorithm converges by the 48th iteration. This suggests that, again, the default 100 iterations for the burn-in period should be sufficient for the convergence of MCMC. Nevertheless, we choose to confirm this visually by repeating the steps from example 2.

We run the MCMC for a total of 2,000 iterations (as would be necessary to obtain 20 imputations) without imputing data and set the seed for reproducibility. We overwrite the existing `wlf.dta` file to contain the new estimates of the WLF by specifying `replace` within `savelwf()`:

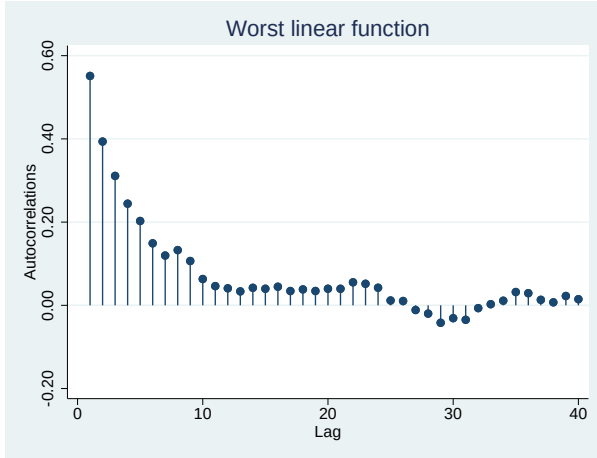
```
. mi impute mvn lnage lntax = price sqft nfeatures ne custom corner,
> mcmconly burnin(2000) rseed(23) savewlf(wlf, replace)

Performing EM optimization:
note: 8 observations omitted from EM estimation because of all imputation
      variables missing.
  observed log likelihood =   112.1464 at iteration 48

Performing MCMC data augmentation ...

Note: No imputation performed.
```

We generate the same graphs as in example 2, this time using the new estimates of the WLF:

```
. preserve

. use wlf, clear

. tsset iter

Time variable: iter, -1999 to 0
        Delta: 1 unit

. tsline wlf, ytitle(Worst linear function) xtitle(Burn-in period)
```



```
. ac wlf, title(Worst linear function) ytitle(Autocorrelations)
> ciopts(astyle(none)) note("")
```



Compared with the earlier graphs, the time-series graphs do not reveal any apparent trend, but the autocorrelation dies out more slowly. The default values of 100 for the initial burn-in and between-imputation iterations should be sufficient.

◁

▷ Example 5: Alternative prior distribution

Consider some hypothetical data:

```
. use https://www.stata-press.com/data/r17/mvnexample0
(Fictional data for -mi impute mvn-)
. mi describe
Style: mlong
       last mi update 22dec2020 10:41:23, 51 days ago
Observations:
   Complete             3
   Incomplete          17  (M = 0 imputations)
   ---------------------
   Total               20
Variables:
   Imputed: 3; x1(16) x2(5) x3(17)
   Passive: 0
   Regular: 0
   System:  3; _mi_m _mi_id _mi_miss
   (there are no unregistered variables)
```

Continuous normally distributed variables `x1`, `x2`, and `x3` contain missing values. For illustration purposes, we consider an extreme case when some variables (`x1` and `x3` here) contain only a few complete observations.

We use `mi impute mvn` to impute missing values and create 30 imputations. Notice that in this example, we do not have complete predictors, and so the right-hand-side specification is empty:

```
. mi imp mvn x1-x3, add(30) rseed(332247)
Performing EM optimization:
note: 4 observations omitted from EM estimation because of all imputation
      variables missing.
  observed log likelihood =  6.5368927 at iteration 100
  (EM did not converge)
Performing MCMC data augmentation ...
Iteration 145: variance-covariance matrix (Sigma) became not
positive definite posterior distribution is not proper
error occurred during imputation of x1 x2 x3 on m = 2
r(498);
```

`mi impute mvn` terminates with an error reporting that the estimated variance–covariance matrix became non–positive definite. `mi impute mvn` terminated because the posterior predictive distribution of missing data is not proper, but notice also that EM did not converge after the default 100 iterations.

There are two issues here. First, because EM did not converge after 100 iterations, we suspect that the default 100 iterations used for the burn-in period may not be large enough for MCMC to converge. Second, the observed missing-data pattern presents difficulties with estimating the covariance matrix reliably, which leads to a non–positive-definite estimate during the MCMC iteration.

The first issue may be resolved by increasing the maximum number of iterations for EM by using EM's `iterate()` suboption. Convergence of EM, however, does not guarantee convergence of the MCMC by the same number of iterations. For one, the convergence of EM is relative to the specified tolerance, and more stringent conditions may lead to a nonconvergent result. As such, we recommend that you always examine the obtained MCMC results.

The second issue is not surprising. Recall that `x1` and `x3` have very few complete observations. So the aspects of the covariance structure involving those variables (for example, the covariance between `x1` and `x2`) are difficult to estimate reliably based on the information from the observed data only. The default uniform prior may not be viable here.

One solution is to introduce prior information to stabilize the estimation of the covariance matrix. We can do this by specifying a ridge prior using the `prior()` option. We introduce only a small amount of information by using a degrees of freedom value of 0.1:

```
. mi imp mvn x1-x3, add(30) prior(ridge, df(0.1)) rseed(332247)

Performing EM optimization:
note: 4 observations omitted from EM estimation because of all imputation
      variables missing.
  observed log posterior =   -1.13422 at iteration 100
  (EM did not converge)

Performing MCMC data augmentation ...

Multivariate imputation                     Imputations =       30
Multivariate normal regression                    added =       30
Imputed: m=1 through m=30                       updated =        0

Prior: ridge, df=.1                          Iterations =     3000
                                                burn-in =      100
                                                between =      100
```

Variable	Observations per m			
	Complete	Incomplete	Imputed	Total
x1	4	16	16	20
x2	15	5	5	20
x3	3	17	17	20

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

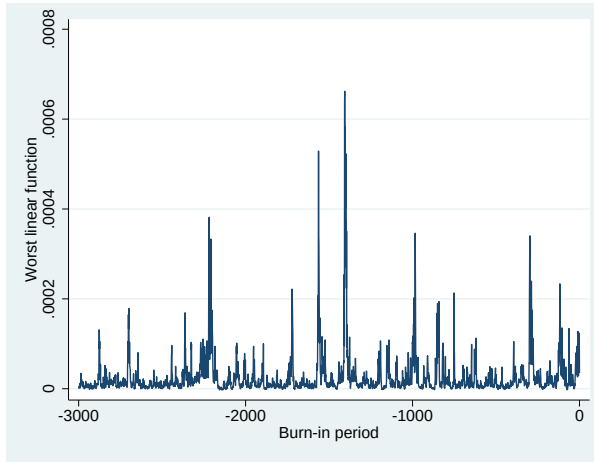
This appears to be enough to alleviate the problem of a non–positive-definite estimate of the covariance matrix. Still, EM did not converge.

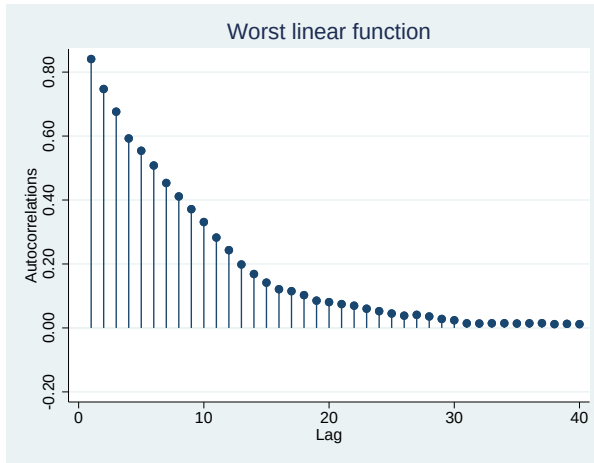
We will fix that and examine the resulting MCMC sequence. We will use the same random-number seed and this time save the WLF. Rather than imputing the data as before, we will simply run the MCMC for the same number of iterations it takes to obtain 30 imputations using the default settings, namely, $30 \times 100 = 3000$.

```
. mi imp mvn x1-x3, mcmconly prior(ridge, df(0.1))
> initmcmc(em, iter(200) nolog) burnin(3000) savewlf(wlf, replace)
> rseed(332242)

Performing EM optimization:
note: 4 observations omitted from EM estimation because of all imputation
      variables missing.
  observed log posterior = -1.1341806 at iteration 152

Performing MCMC data augmentation ...

Note: No imputation performed.
```

We increased the maximum number of iterations for the EM algorithm to 200; it converged in iteration 152.

We use the results from `wlf.dta` to obtain the trace and autocorrelation plots as we did in the earlier examples:





The serial correlation decreases slowly. There is no obvious trend in the WLF estimates, but we notice high variability and several spikes, some distinctive. The high variability and spikes are not surprising considering that certain model parameters could not be estimated reliably from the observed data and considering that we did not introduce enough prior information to obtain less variable estimates; we introduced only enough to achieve nonsingularity.

We could decrease the variability of the estimates by obtaining more data or introducing stronger prior information. For example, we could increase the number of degrees of freedom with a ridge prior to constrain the covariance matrix toward a diagonal structure:

```
. mi imp mvn x1-x3, replace prior(ridge, df(10)) burnin(300) rseed(332247)
(output omitted)
```

If we create and examine the trace plots and autocorrelations of the WLF under the new prior specification, we find that variability of the estimates and serial dependence decrease greatly at a cost of bias if the prior assumptions are false.

◁

▷ Example 6: Saving all parameter series

The examples above used the WLF to monitor convergence of MCMC because in most applications it is sufficient. Although the WLF series often behave as the worst-case scenario, exceptions exist in practice. Sometimes, examining individual parameter series may be necessary.

We can save all parameter series from MCMC by using the `saveptrace()` option. These parameter series are saved in a parameter-trace file, a special file with extension `.stptrace`. Although the resulting file is not a Stata dataset, it can easily be loaded into Stata using `mi ptrace use`; see [MI] **mi ptrace** for details.

Let's look at several parameter series from the above example.

```
. use https://www.stata-press.com/data/r17/mvnexample0, clear
. mi imp mvn x1-x3, mcmconly prior(ridge, df(0.1)) initmcmc(em, iter(200) nolog)
> burnin(3000) rseed(332247) saveptrace(parms)
```

We save all parameter series to a file called `parms` by using `stptrace(parms)`.

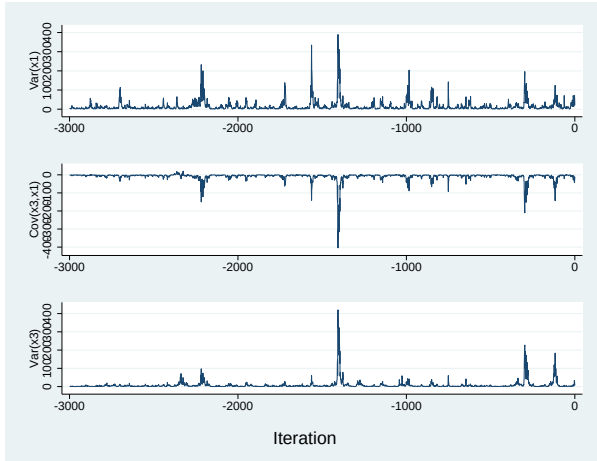
We first describe the contents of the `parms` file and then read it into Stata:

```
. mi ptrace describe parms
  file parms.stptrace created on 30 Apr 2021 22:51 contains 3,000 records
  (obs) on
      m                       1 variable
      iter                    1 variable
      b[y, x]                 3 variables (3 x 1)
      v[y, y]                 6 variables (3 x 3, symmetric)
  where y and x are
      y: (1) x1  (2) x2  (3) x3
      x: (1) _cons
. mi ptrace use parms, clear
```

The output from `mi ptrace describe` reports that the file contains imputation numbers, iteration numbers, estimates of three regression coefficients (`b[x1,_cons]`, `b[x2,_cons]`, and `b[x3,_cons]`, which are effectively the means of `x1`, `x2`, and `x3`), and estimates of six covariances (`v[x1,x1]`, `v[x2,x1]`, `v[x2,x2]`, and so on).

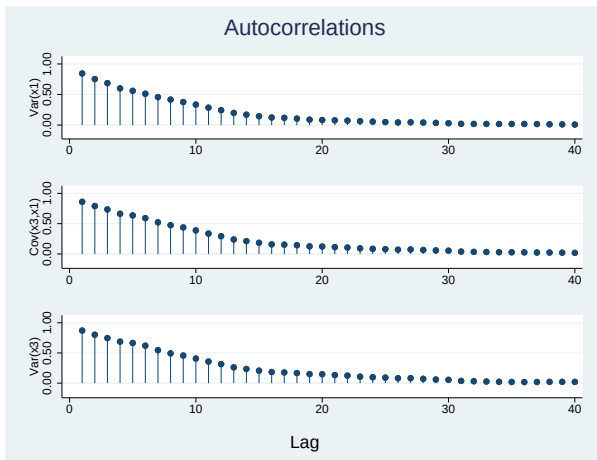
Because `x1` and `x3` contain the least number of complete observations, we examine the series containing their variance and covariance estimates. We generate graphs separately for each series and then combine them in one graph by using `graph combine`; see [G-2] **graph combine**.

```
. tsset iter
Time variable: iter, -2999 to 0
        Delta: 1 unit
. tsline v_y1y1, name(gr1) nodraw ytitle(Var(x1)) xtitle("") ylabel(#4)
. tsline v_y3y1, name(gr2) nodraw ytitle(Cov(x3,x1)) xtitle("") ylabel(#4)
. tsline v_y3y3, name(gr3) nodraw ytitle(Var(x3)) xtitle("") ylabel(#4)
. graph combine gr1 gr2 gr3, xcommon cols(1) b1title(Iteration)
```



We repeat the same for the autocorrelation graphs:

```
. ac v_y1y1, ytitle(Var(x1)) xtitle("") ciopts(astyle(none)) note("")
> name(gr1, replace) nodraw ylabel(#4)
. ac v_y3y1, ytitle(Cov(x3,x1)) xtitle("") ciopts(astyle(none)) note("")
> name(gr2, replace) nodraw ylabel(#4)
. ac v_y3y3, ytitle(Var(x3)) xtitle("") ciopts(astyle(none)) note("")
> name(gr3, replace) nodraw ylabel(#4)
. graph combine gr1 gr2 gr3, xcommon cols(1) title(Autocorrelations) b1title(Lag)
```



We can see that the trace plot and autocorrelations corresponding to the variance of `x1` resemble the patterns of the earlier WLF estimates. We also notice that all series have high serial dependence within the first 20 iterations.

Again, if we switch to using a ridge prior with 10 degrees of freedom and repeat the steps above, the obtained trace plots will be more precise and more regular. The serial dependence in the series will be lower.

◁

Stored results

mi impute mvn 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(burnbetween)	number of burn-between iterations
r(df_prior)	prior degrees of freedom (stored only with prior(ridge))
r(N_em)	number of observations used by EM (including omitted missing observations)
r(N_e_em)	number of observations used by EM in estimation (excluding omitted missing observations)
r(N_mis_em)	number of incomplete observations within the EM estimation sample
r(N_S_em)	number of unique missing-value patterns
r(niter_em)	number of iterations EM takes to converge
r(llobs_em)	observed log likelihood (stored with prior(uniform))
r(lpobs_em)	observed log posterior (stored with priors other than uniform)
r(converged_em)	convergence flag for EM
r(emonly)	1 if performed EM estimation only, 0 otherwise
r(mcmconly)	1 if performed MCMC only without imputing data, 0 otherwise
r(N_g)	number of imputed groups (1 if by() is not specified)

Macros

r(method)	name of imputation method (mvn)
r(ivars)	names of imputation variables
r(rngstate)	random-number state used
r(prior)	prior distribution
r(init_mcmc)	type of initial values (em or user)
r(ivarsorder)	names of imputation variables in the order used in the computation
r(init_em)	type of initial values used by EM (ac, cc, or user)
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)
r(Beta0)	initial values for regression coefficients used by DA
r(Sigma0)	initial variance–covariance matrix used by DA
r(wlf_wgt)	coefficients for the WLF (stored with initmcmc(em) or if wlfwgt() is used)
r(Beta_em)	estimated regression coefficients from EM
r(Sigma_em)	estimated variance–covariance matrix from EM
r(Beta0_em)	initial values for regression coefficients used by EM
r(Sigma0_em)	initial variance–covariance matrix used by EM
r(N_pat)	minimum, average, and maximum numbers of observations per missing-value pattern

r(N_pat) and results with the _em suffix are stored only when the EM algorithm is used (with emonly or initmcmc(em)).

Methods and formulas

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be a random sample from a p-variate normal distribution recording values of p imputation variables. Consider a multivariate normal regression

$$\mathbf{x}_i = \boldsymbol{\Theta}'\mathbf{z}_i + \boldsymbol{\epsilon}_i,\ i = 1, \ldots, N$$

where $\mathbf{z}_i$ is a $q \times 1$ vector of independent (complete) variables from observation i, $\boldsymbol{\Theta}$ is a $q \times p$ matrix of regression coefficients, and $\boldsymbol{\epsilon}_i$ is a $p \times 1$ vector of random errors from a p-variate normal distribution with a zero mean vector and a $p \times p$ positive-definite covariance matrix $\boldsymbol{\Sigma}$. We refer to $\boldsymbol{\Theta}$ and $\boldsymbol{\Sigma}$ as model parameters. Consider the partition $\mathbf{x}_i = (\mathbf{x}_{i(m)}, \mathbf{x}_{i(o)})$ corresponding to missing and observed values of imputation variables in observation i for $i = 1, \ldots, N$.

Methods and formulas are presented under the following headings:

Data augmentation

`mi impute mvn` uses data augmentation (DA) to fill in missing values in $\mathbf{x}_i$ independently for each observation $i = 1, \ldots, N$. Data augmentation consists of two steps, an I step (imputation step) and a P step (posterior step), performed at each iteration $t = 0, 1, \ldots, T$. At iteration t of the I step, missing values in $\mathbf{x}_i$ are replaced with draws from the conditional posterior distribution of $\mathbf{x}_{i(m)}$ given observed data and current values of model parameters independently for each $i = 1, \ldots, N$. During the P step, new values of model parameters are drawn from their conditional posterior distribution given the observed data and the data imputed in the previous I step. Mathematically, this process can be described as follows:

I step:

$$\mathbf{x}_{i(m)}^{(t+1)} \sim P\left(\mathbf{x}_{i(m)}|\mathbf{z}_i, \mathbf{x}_{i(o)}, \boldsymbol{\Theta}^{(t)}, \boldsymbol{\Sigma}^{(t)}\right),\ i = 1, \ldots, N \tag{1}$$

P step:

$$\begin{aligned}\boldsymbol{\Sigma}^{(t+1)} &\sim P\left(\boldsymbol{\Sigma}|\mathbf{z}_i, \mathbf{x}_{i(o)}, \mathbf{x}_{i(m)}^{(t+1)}\right)\\ \boldsymbol{\Theta}^{(t+1)} &\sim P\left(\boldsymbol{\Theta}|\mathbf{z}_i, \mathbf{x}_{i(o)}, \mathbf{x}_{i(m)}^{(t+1)}, \boldsymbol{\Sigma}^{(t+1)}\right)\end{aligned} \tag{2}$$

The above two steps are repeated until the specified number of iterations, T, is reached. The total number of iterations, T, is determined by the length of the initial burn-in period, b, and the number of iterations between imputations, k. Specifically, $T = b + M_{new} \times k$, where M_{new} contains the number of added and updated imputations. `mi impute mvn` saves imputed values $\mathbf{x}_{i(m)}^{(t_1)}, \mathbf{x}_{i(m)}^{(t_2)}, \ldots, \mathbf{x}_{i(m)}^{(t_{M_{\text{new}}})}$ as final imputations, where iteration $t_i = b + (i-1)k$.

By default, `mi impute mvn` uses default values of 100 for b and k. These values may be adequate in some applications and may be too low in others. In general, b and k must be determined based on the properties of the observed Markov chain $\left(\mathbf{X}_m^{(1)}, \boldsymbol{\Theta}^{(1)}, \boldsymbol{\Sigma}^{(1)}\right), \left(\mathbf{X}_m^{(2)}, \boldsymbol{\Theta}^{(2)}, \boldsymbol{\Sigma}^{(2)}\right), \ldots$, where $\mathbf{X}_m^{(t)}$ denotes all values imputed at iteration t. b must be large enough so that the above chain converges to the stationary distribution $P(\mathbf{X}_m, \boldsymbol{\Theta}, \boldsymbol{\Sigma}|\mathbf{Z}, \mathbf{X}_o)$ by iteration $t = b$. k must be large enough so that random draws (imputations) $\mathbf{x}_{i(m)}^{(t_1)}, \mathbf{x}_{i(m)}^{(t_2)}, \ldots$ are approximately independent. See *Convergence of the MCMC method* for more details.

The functional form of the conditional posterior distributions (1) and (2) depends on the distribution of the data and a prior distribution of the model parameters. `mi impute mvn` assumes an improper uniform prior distribution for $\boldsymbol{\Theta}$ and an inverted Wishart distribution (Mardia, Kent, and Bibby 1979, 85) $W_p^{-1}(\Lambda, \lambda)$ for $\boldsymbol{\Sigma}$ under which the prior joint density function is

$$f(\boldsymbol{\Theta}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\left(\frac{\lambda+p+1}{2}\right)} \exp\left(-\frac{1}{2}\text{tr}\Lambda^{-1}\boldsymbol{\Sigma}^{-1}\right)$$

Under the multivariate normal model and the above prior distribution, the I and P steps become (Schafer 2008; Schafer 1997, 181–185) the following:

$$\text{I step:} \qquad \mathbf{x}_{i(m)}^{(t+1)} \sim N_{p_i}\left(\boldsymbol{\mu}_{m\cdot o}^{(t)}, \boldsymbol{\Sigma}_{mm\cdot o}^{(t)}\right),\ i = 1, \ldots, N$$

$$\text{P step:} \qquad \boldsymbol{\Sigma}^{(t+1)} \sim W^{-1}(\Lambda_\star^{(t+1)}, \lambda_\star)$$

$$\operatorname{vec}\left(\boldsymbol{\Theta}^{(t+1)}\right) \sim N_{pq}\left\{\operatorname{vec}\left(\widehat{\boldsymbol{\Theta}}^{(t+1)}\right), \boldsymbol{\Sigma}^{(t+1)} \otimes (\mathbf{Z}'\mathbf{Z})^{-1}\right\}$$

where p_i is the number of imputation variables containing missing values in observation i and $\otimes$ is the Kronecker product. Submatrices $\boldsymbol{\mu}_{m\cdot o}^{(t)}$ and $\boldsymbol{\Sigma}_{mm\cdot o}^{(t)}$ are the mean and variance of the conditional normal distribution of $\mathbf{x}_{i(m)}$ given $\mathbf{x}_{i(o)}$ based on $(\mathbf{x}_{i(m)}, \mathbf{x}_{i(o)}|\mathbf{z}_i) \sim N_p\left(\boldsymbol{\Theta}^{(t)\prime}\mathbf{z}_i, \boldsymbol{\Sigma}^{(t)}\right)$. See, for example, Mardia, Kent, and Bibby (1979, 63) for the corresponding formulas of the conditional mean and variance of the multivariate normal distribution. The matrix $\widehat{\boldsymbol{\Theta}}^{(t+1)} = (\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}^{(t+1)}$ is the OLS estimate of the regression coefficients based on the augmented data $\mathbf{X}^{(t+1)} = (\mathbf{X}_o, \mathbf{X}_m^{(t+1)})$ from iteration t. The posterior cross-product matrix $\Lambda_\star^{(t+1)}$ and the posterior degrees of freedom $\lambda_\star$ are defined as follows:

$$\Lambda_\star^{(t+1)} = \left\{\Lambda^{-1} + (\mathbf{X}^{(t+1)} - \mathbf{Z}\widehat{\boldsymbol{\Theta}}^{(t+1)})'(\mathbf{X}^{(t+1)} - \mathbf{Z}\widehat{\boldsymbol{\Theta}}^{(t+1)})\right\}^{-1}$$

and

$$\lambda_\star = \lambda + N - q$$

Prior distribution

As we already mentioned, `mi impute mvn` assumes an improper uniform prior distribution for $\boldsymbol{\Theta}$ and an inverted Wishart distribution for $\boldsymbol{\Sigma}$ under which the prior joint density function is

$$f(\boldsymbol{\Theta}, \boldsymbol{\Sigma}) \propto |\boldsymbol{\Sigma}|^{-\left(\frac{\lambda+p+1}{2}\right)} \exp\left(-\frac{1}{2}\operatorname{tr}\Lambda^{-1}\boldsymbol{\Sigma}^{-1}\right)$$

Parameters of the inverted Wishart prior distribution, the prior cross-product matrix Λ, and the prior degrees of freedom λ are determined based on the requested prior distribution.

By default, `mi impute mvn` uses the uniform prior distribution under which $\lambda = -(p+1)$ and $\Lambda^{-1} = \mathbf{0}_{p\times p}$. Under the uniform prior, the log-likelihood and log-posterior functions are equivalent, and so the ML estimates of the parameters are equal to the posterior mode.

Under the noninformative Jeffreys prior distribution, $\lambda = 0$ and $\Lambda^{-1} = \mathbf{0}_{p\times p}$.

Under a ridge prior distribution, λ is equal to the user-specified value, and $\Lambda^{-1} = \lambda\boldsymbol{\Sigma}_\star$, where the diagonal matrix $\boldsymbol{\Sigma}_\star$ contains the diagonal elements of the estimate of the covariance matrix using all available cases. The variances (diagonal estimates) are the estimates of the mean squared error from regression of each imputation variable on the complete predictors. See Schafer (1997, 155–157) for details. With $\lambda = 0$, this prior specification reduces to the Jeffreys prior.

Initial values: EM algorithm

Initial values $\boldsymbol{\Theta}^{(0)}$ and $\boldsymbol{\Sigma}^{(0)}$ for DA are obtained from the EM algorithm for the incomplete multivariate normal data (for example, Dempster, Laird, and Rubin [1977], Little and Rubin [2020], Schafer [1997]). The EM algorithm iterates between the expectation step (E step) and the maximization step (M step) to maximize the log-likelihood (or log-posterior) function.

The observed-data log likelihood is

$$l_l(\boldsymbol{\Theta}, \boldsymbol{\Sigma}|\mathbf{X}_o) = \sum_{s=1}^{S} \sum_{i \in I(s)} \left\{ -0.5 \ln(|\boldsymbol{\Sigma}_s|) - 0.5(\mathbf{x}_{i(o)} - \boldsymbol{\Theta}_s' \mathbf{z}_i)' \boldsymbol{\Sigma}_s^{-1} (\mathbf{x}_{i(o)} - \boldsymbol{\Theta}_s' \mathbf{z}_i) \right\}$$

where S is the number of unique missing-value patterns, $I(s)$ is the set of observations from the same missing-value pattern s, and $\boldsymbol{\Theta}_s$ and $\boldsymbol{\Sigma}_s$ are the submatrices of $\boldsymbol{\Theta}$ and $\boldsymbol{\Sigma}$ that correspond to the imputation variables, which are observed in pattern s.

The observed-data log posterior is

$$l_p(\boldsymbol{\Theta}, \boldsymbol{\Sigma}|\mathbf{X}_o) = l_l(\boldsymbol{\Theta}, \boldsymbol{\Sigma}|\mathbf{X}_o) + \ln\{f(\boldsymbol{\Theta}, \boldsymbol{\Sigma})\} = l_l(\boldsymbol{\Theta}, \boldsymbol{\Sigma}|\mathbf{X}_o) - \frac{\lambda + p + 1}{2} \ln(|\boldsymbol{\Sigma}|) - \mathrm{tr}(\Lambda^{-1} \boldsymbol{\Sigma}^{-1})$$

The E step and M step of the EM algorithm are defined as follows (see Schafer [2008; 1997, 163–175] for details).

Let $T_1 = \sum_{i=1}^{N} \mathbf{z}_i \mathbf{x}_i'$ and $T_2 = \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i'$ denote the sufficient statistics for the multivariate normal model. Consider the submatrices $\boldsymbol{\Theta}_{i(o)}$ and $\boldsymbol{\Theta}_{i(m)}$ of $\boldsymbol{\Theta}$, and the submatrices $\boldsymbol{\Sigma}_{i(mm)}$, $\boldsymbol{\Sigma}_{i(mo)}$, and $\boldsymbol{\Sigma}_{i(oo)}$ of $\boldsymbol{\Sigma}$ corresponding to the observed and missing columns of $\mathbf{x}_i$. Let $O(s)$ and $M(s)$ correspond to the column indexes of the observed and missing parts of $\mathbf{x}_i$ for each missing-values pattern s.

During the E step, the expectations $E(T_1)$ and $E(T_2)$ are computed with respect to the conditional distribution $\Pr(\mathbf{X}_m|\mathbf{X}_o, \boldsymbol{\Theta}^{(t)}, \boldsymbol{\Sigma}^{(t)})$ using the following relations:

$$E(x_{ij}|\mathbf{X}_o, \boldsymbol{\Theta}^{(t)}, \boldsymbol{\Sigma}^{(t)}) = \begin{cases} x_{ij}, & \text{for } j \in O(s) \\ x_{ij}^{\star}, & \text{for } j \in M(s) \end{cases}$$

and

$$E(x_{ij}x_{il}|\mathbf{X}_o, \boldsymbol{\Theta}^{(t)}, \boldsymbol{\Sigma}^{(t)}) = \begin{cases} x_{ij}x_{il}, & \text{for } j, l \in O(s) \\ x_{ij}^{\star}x_{il}, & \text{for } j \in M(s), l \in O(s) \\ c_{ij} + x_{ij}^{\star}x_{il}^{\star}, & \text{for } j, l \in M(s) \end{cases}$$

where $x_{ij}^{\star}$ is the jth element of the vector $\boldsymbol{\Theta}_{i(m)}' \mathbf{z}_i + \boldsymbol{\Sigma}_{i(mo)} \boldsymbol{\Sigma}_{i(oo)}^{-1} \left(\mathbf{x}_{i(o)} - \boldsymbol{\Theta}_{i(o)}' \mathbf{z}_i \right)$, and c_{ij} is the element of the matrix $\boldsymbol{\Sigma}_{i(mm)} - \boldsymbol{\Sigma}_{i(mo)} \boldsymbol{\Sigma}_{i(oo)}^{-1} \boldsymbol{\Sigma}_{i(mo)}'$.

During the M step, the model parameters are updated using the computed expectations of the sufficient statistics:

$$\boldsymbol{\Theta}^{(t+1)} = (\mathbf{Z}'\mathbf{Z})^{-1} E(T_1)$$

$$\boldsymbol{\Sigma}^{(t+1)} = \frac{1}{N + \lambda + p + 1} \left\{ E(T_2) - E(T_1)' (\mathbf{Z}'\mathbf{Z})^{-1} E(T_1) + \Lambda^{-1} \right\}$$

EM iterates between the E step and the M step until the maximum relative difference between the two successive values of all parameters is less than the default tolerance of 1e–5 (or the specified `tolerance()`).

Worst linear function

The worst linear function (WLF) is defined as follows (Schafer 1997, 129–131):

$$\xi(\theta) = \widehat{v}_1'(\theta - \widehat{\theta})$$

where θ and $\widehat{\theta}$ are column vectors of the unique model parameters and their respective EM estimates; $\widehat{v}_1 = \theta^{(t)} - \theta^{(t-1)}$, where $\theta^{(t)} = \widehat{\theta}$ and $\theta^{(t-1)}$ are the estimates from the last and one before the last iterations of the EM algorithm. This function is regarded to be the WLF because it has the highest asymptotic rate of missing information among all linear functions. This function is derived based on the convergence properties of the EM algorithm (see Schafer [1997, 55–59] for details).

References

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* 39: 1–38. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x.

Gelman, A., J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. 2014. *Bayesian Data Analysis*. 3rd ed. Boca Raton, FL: Chapman & Hall/CRC.

Gelman, A., and D. B. Rubin. 1992. Inference from iterative simulation using multiple sequences. *Statistical Science* 7: 457–472. https://doi.org/10.1214/ss/1177011136.

Li, K.-H. 1988. Imputation using Markov chains. *Journal of Statistical Computation and Simulation* 30: 57–79. https://doi.org/10.1080/00949658808811085.

Little, R. J. A., and D. B. Rubin. 2020. *Statistical Analysis with Missing Data*. 3rd ed. Hoboken, NJ: Wiley.

Mardia, K. V., J. T. Kent, and J. M. Bibby. 1979. *Multivariate Analysis*. London: Academic Press.

Rubin, D. B. 1987. *Multiple Imputation for Nonresponse in Surveys*. New York: Wiley.

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

——. 2008. *NORM: Analysis of incomplete multivariate data under a normal model, Version 3*. Software package for R. University Park, PA: The Methodology Center, Pennsylvania State University.

Tanner, M. A., and W. H. Wong. 1987. The calculation of posterior distributions by data augmentation (with discussion). *Journal of the American Statistical Association* 82: 528–550. https://doi.org/10.2307/2289457.

Also see

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

[MI] **mi impute chained** — Impute missing values using chained equations

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

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

[MI] **Intro** — Introduction to mi

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

[MI] **Glossary**