Bayesian Analysis with Stata

John Thompson
Department of Health Sciences
Univeristy of Leicester
john.thompson@le.ac.uk

March 19, 2014

1 Introduction

The manual offers a printed version of the information in the help files of the commands written for the book *Bayesian Analysis with Stata*.

The ado files and help files can be net installed from,

```
http://stata-press.com/data/bas
```

and the library for running Mata versions of the commands can be obtained from the same source using **net get bas**. This downloads a do file that must be run in order to create the library.

Alternatively the files can be downloaded in zipped form from my webpage at **http://www2.le.ac.uk/Members/trj/home**.

command	description
`gbs` *Gibbs samplers*	
`gbsars`	adaptive rejection sampling
`gbsarms`	adaptive rejection metropolis sampling
`gbsgriddy`	griddy sampling
`gbsslice`	univariate slice sampling
`mcmc` *Exploring MCMC simulations*	
`mcmcac`	autocorrelation plots
`mcmcbgr`	Brooks-Gelman-Rubin convergence plot
`mcmccheck`	model checking
`mcmccusum`	cusum plot
`mcmcdensity`	posterior density plot
`mcmcgeweke`	convergence test based on means
`mcmcintervals`	interval plot for sections of a chain
`mcmclength`	run length required for a given accuracy
`mcmcmahal`	Mahanalobis distance plot
`mcmcsection`	convergence check based on plotted densities
`mcmcstats`	summary statistics
`mcmctrace`	trace or history plot
`mhs` *Metropolis-Hasting samplers*	
`mhslogn`	lognormal random walk
`mhsmnorm`	multivariate normal random walk
`mhsnorm`	normal random walk
`mhstrnc`	random walk over a restricted range
`wbs` *Communication with WinBUGS*	
`wbsarray`	write data as a WinBUGS formatted array
`wbscoda`	read WinBUGS results into Stata
`wbsdecode`	read a WinBUGS formatted list into Stata
`wbslist`	write data or initial values in list format
`wbsmodel`	write model in WinBUGS syntax to a text file
`wbsrun`	run WinBUGS from within Stata
`wbsscript`	create a WinBUGS script file
Other programs	
`logdensity`	log-densities of standard distributions
`mcmcrun`	create MCMC simulations without WinBUGS

`gbsars` - Adaptive Rejection Sampler

Syntax

`gbsars` *logpost b ipar [weight] if in*, [*options*]

options	description
`lbound(#)`	lower limit of the range of the grid
`ubound(#)`	upper limit of the range of the grid
`grid(`*numlist*`)`	The initial grid of points
`maxtry(#)`	maximum number of points before the algorithm stops

Description

`gbsars` uses ARS (adaptive rejection sampling) to simulate a value from a one dimensional posterior distribution. Useful in Gibbs sampling. The posterior MUST be log-concave, so the algorithm is not completely general. The grid must consist of at least 3 points placed across the region where the posterior is thought to lie. Values may be generated outside the range of this initial grid but not beyond the upper and lower bounds. The log-posterior is evaluated at each of the points in the grid and those values are used to define an envelope enclosing the log-posterior. The program uses the envelope for rejection sampling but any rejected points are added to the grid in order to improve the fit of the envelope. A warning is printed if the posterior is found not to be log-concave and the program will stop. If the function is not log-concave then the more general `gbsarms` algorithm may be used or the user might try slice sampling with `gbsslice` or a Metropolis-Hastings algorithm such as `mhsnorm`. For speed of execution the algorithm is programmed in Mata, although the user's program for evaluating the log-posterior may be written in either Stata or Mata. The Mata code for the function is stored in the library `libmcmc` which must be installed for the command to work.

Options

The inputs that follow the command word must be, the name of the user's program for calculating the log of the conditional posterior density (log-post), the name of the row vector containing the parameter values (b) and the number of the parameter that is being updated (ipar). When the user's program is called the arguments passed are a scalar that is to contain the value of the log posterior, the row vector of parameters and the current parameter number. See `mhsnorm` for more details on how to write the program for the log posterior.

`lbound(#)` lower limit of the range of the posterior. Defaults to minus infinity.
`ubound(#)` upper limit of the range of the posterior. Defaults to infinity.
`grid(`*numlist*`)` Set of at least 3 ascending values defining the grid for the envelope that encloses the log-posterior.
`maxtry(#)` number of rejection sampling attempts before giving up. Defaults to 50. If the program takes more than a handful of attempts then either the posterior is extremely awkward or the initial grid was very poorly placed.

Returned Results

returns `r(logp)`, the value of the log of the density at the selected point, `r(try)`, the number of function evaluations required before a point was accepted and `r(fail)` a 0,1 indicator of whether the algorithm reached `maxtry` without finding an acceptable point. Evidence of lack of log-concavity will cause gbsars to print a warning and exit with an error.

Mata

To call the Mata function directly requires,
void **gbsars**(*real scalar* logp,*pointer function* f,*real matrix* X,*real rowvector* theta,*real scalar* ipar,*string scalar* grid,*real scalar* maxtry, *real scalar* lbound, *real scalar* ubound | *real scalar* trys,*real scalar* fail)

Example

```
program logpost
   args logp b ipar
 ...
end
matrix b = (2.5, 1.9, 4.0)
gbsars logpost b 2 , grid(1.5 2.0 2.5) lbound(0)
```

`gbsarms` - Adaptive Rejection Metropolis Sampler

Syntax

`gbsarms` *logpost b ipar [weight] if in*, `[` *`options`* `]`

options	description
`lbound(#)`	lower limit of the range of the grid
`ubound(#)`	upper limit of the range of the grid
`grid(`*numlist*`)`	The initial grid of points
`maxtry(#)`	maximum number of points before the algorithm stops
`logp(#)`	the value of log-posterior at the current point

Description

`gbsarms` uses ARMS sampling (adaptive rejection Metropolis sampler) to simulate a value from a one dimensional posterior distribution. Useful in Gibbs sampling. The grid consists of at least 3 points placed across the region where the posterior is thought to lie. Values may be generated outside the range of this initial grid but not beyond the upper and lower bounds. The log-posterior is evaluated at each point in the grid and those values are used to define an envelope enclosing the log-posterior. The algorithm starts by using the envelope for adaptive rejection sampling in which rejected points are added to the grid. To allow for the possibility that the envelope might not always lie above the log-posterior the point obtained from rejection sampling is subjected to a Metropolis-Hastings step. As a Metropolis step is involved in generating the new value, it is not valid to place the initial grid dependent on previous values from the same chain. The user must supply a program for evaluating the log-posterior for any given set of parameter values, see `mhsnorm` for details of the structure of the program. The user's function may be written in Stata or Mata but the calculations for `gbsarms` are programmed in Mata. The Mata code is stored in the library `libmcmc` which must be installed for the command to work.

`gbsars` is similar to ARMS but does not have a Metropolis step. It is slightly more efficient but only works if the log-posterior is log-concave.

Options

The inputs that follow the command word must be, the name of the user's program for calculating the log of the conditional posterior density, the name of the row vector containing the parameter values and the number of the parameter that is being updated. When the user's program is called the arguments passed are a scalar that is to contain the value of the log posterior, the row vector of parameters and the current parameter number. See `mhsnorm` for more details on how to write the program for the log posterior.

`lbound(#)` lower limit of the range of the posterior. Defaults to minus infinity.
`ubound(#)` upper limit of the range of the posterior. Defaults to infinity.
`grid(`*numlist*`)` Set of at least 3 ascending values defining the grid for the envelope that encloses the log-posterior.
`maxtry(#)` number of rejection sampling attempts before giving up. Defaults to 50. If the program takes more than a handful of attempts then either the posterior is extremely awkward or the initial grid was very poorly placed.
`logp(#)` Specifies the value of the log-posterior at the current parameter value. If not specified or set to missing the value is calculated.

Returned Results

returns `r(logp)`, the value of the log of the density at the selected point, `r(try)`, the number of function evaluations required before a point was accepted and `r(fail)` a 0,1 indicator of whether the algorithm reached `maxtry` without finding an acceptable point.

Mata

To call the Mata function directly requires,
void **gbsarms**(*real scalar* logp,*pointer function* f,*real matrix* X,*real rowvector* theta,*real scalar* ipar,*string scalar* grid,*real scalar* maxtry, *real scalar* lbound, *real scalar* ubound | *real scalar* trys,*real scalar* fail)

Example

```
program logpost
   args logp b ipar
 ...
end
matrix b = (2.5, 1.9, 4.0)
gbsarms logpost b 2 , grid(1.5 2.0 2.5 3.0) lbound(0)
```

`gbsgriddy` - Griddy Sampler

Syntax

`gbsgriddy` *logpost b ipar [weight] if in*, `[` *options* `]`

options	description
`grid(`*numlist*`)`	The initial grid of points
`metropolis`	Metropolis acceptance of the sampled point
`histogram`	histogram approximation to the log-function
`logp(#)`	with Metropolis, the value of log-posterior at the current point

Description

`gbsgriddy` uses Griddy sampling to simulate a value from a one dimensional posterior distribution. Useful in Gibbs sampling. Griddy sampling works by approximating the posterior by either a histogram or a set of straight lines drawn through the log posterior evaluated at a grid of points. By default a straight line approximation is used. The grid must consist of at least 4 ascending values specified by the user. Placement of the grid is vital to the success of the algorithm, so it is only useful if the user knows roughly where the posterior will lie. The Metropolis option tests the generated value against the previous value using a Metropolis step. This makes the method exact within the range of the grid. If a Metropolis step is used then the grid must not be placed based on information derived from the same chain. For speed, the algorithm is programmed in Mata although the user's program for evaluating the log-posterior may be written in Stata or Mata. The Mata code is stored in the library `libmcmc` which must be installed for the command to work.

Options

The inputs that follow the command word must be, the name of the user's program for calculating the log of the conditional posterior (logpost), the name of the row vector containing the parameter values (b), the number of the parameter that is being updated (ipar). The smallest and largest values of the grid must be chosen carefully because it is impossible for the algorithm to simulate values outside that range. When the user's program is called the arguments passed are a scalar that will contain the value of the log posterior, the row vector of parameters and the current parameter number. See mhsnorm for more details on how to write the program for the log posterior.

`grid(`*numlist*`)` Set of points defining the initial grid for the envelope that

approximates the log-posterior. The grid must contain at least 4 points.
`metropolis` By default gbsgriddy uses an approximation to the posterior but this can be made exact within the range of the grid by treating the approximation as a proposal distribution and using a Metropolis step to accept or reject the generated value.
`histogram` requests a histogram approximation to the density rather than a straight line approximation.
`logp(#)` log-posterior for the current set of parameter values. Only useful if a Metropolis update is requested. If set to missing, this value is calculated by the program.

Returned Results

Returns `r(logp)` containing the log of the function at the new point.

Mata

To call the Mata function directly requires,
*void***gbsgriddy**(*real scalar* logp,*pointer function* pf,*real matrix* X,*real rowvector* theta,*real scalar* ipar,*string scalar* grid| *real scalar* metropolis,*real scalar* histogram)

`gbsslice` - Slice Sampler

Syntax

`gbsslice` *logpost b ipar [weight] if in*`, [` *options* `]`

options	description
`nstep(#)`	number of expansion steps
`stepsize(#)`	size of expansion steps
`lbound(#)`	lower limit of the range of the grid
`ubound(#)`	upper limit of the range of the grid
`maxtry`	maximum number of attempts before the algorithm stops
`logp(#)`	the value of log-posterior at the current point

Description

`gbsslice` uses slice sampling to sample from a one dimensional posterior distribution. Useful in Gibbs sampling. It is a Metropolis algorithm that creates its proposal by choosing a random height above the current point and then moving outwards by adding steps until a slice is created that has a greater width than the density. Random points are then generated within the slice. The slice is shrunk using rejected points. Calculations are performed by a Mata function but it can be called from Stata as well as from within a Mata program. Requires a user written program for calculating the log-posterior that may be written in Stata or Mata. The Mata code is stored in the library `libmcmc` which must be installed for the command to work.

Options

The inputs that follow the command word must be, the name of the user's program for calculating the log of the conditional posterior density (logpost), the name of the row vector containing the parameter values (b) and the number of the parameter that is being updated (ipar). When the user's program is called the arguments passed are a scalar that is to contain the value of the log posterior, the row vector of parameters and the current parameter number. See `mhsnorm` for more details on how to write the program for the log posterior.
`nstep(#)` maximum number of steps outwards that are attempted when trying to locate the end of the slice. Defaults to 6.
`stepsize(#)` size of the increments that are added in an attempt to locate the end of the slice. Choosing a value that is approximately equal to the range of the conditional distribution usually works well. Defaults to 1.
`lbound(#)` lower bound on the parameter. The slice will not extend below this value. Defaults to minus infinity.
`ubound(#)` upper bound on the parameter. The slice will not extend above this value. Defaults to infinity.
`maxtry(#)` maximum number of attempts to generate a parameter value before the algorithm gives up. Defaults to 50. Typically slice sampling finds a new value within a handful of attempts so exceeding 50 would probably indicate a very poorly chosen stepsize or an error in logpost
`logp(#)` Value of the log-posterior at the current point. If a non-missing value is given that value is used; otherwise the value is calculated using the user's function logpost. Specifying the value when it is known will save one call to the user's function and so speed the algorithm.

Returned Results

Returns `r(logp)` containing the log of the density at the sampled point and `r(try1)` and `r(try2)`the number of function evaluations needed by the algorithm first when expanding the slice and then when sampling points within the slice. `r(fail)` returns 1 if maxtry is reached before sampling a point within the slice.

Mata

To call the Mata function directly requires,
*void***gbsslice**(*real scalar* logp,*pointer function* pf,*real matrix* X,*real rowvector* theta,*real scalar* ipar,*real scalar*stepsize,*real scalar*nstep,*real scalar*lbound,*real scalar*ubound,*real scalar*maxtry| *real scalar*fail, *real scalar*try1,*real scalar*try2)

`logdensity` - log-densities of standard distributions

Syntax

`logdensity` *distribution logp value par1 par2 .. [weight] if in*

Description

`logdensity` evaluates the log-density of a standard distribution and adds the result to a user supplied scalar (logp). The function is intended for use in programs for calculating the log-posterior, see `mhsnorm`. However, for complex models that require a lot of evaluations of the log-posterior, it might be worth replacing the calls to logdensity with user-written code that can be made more efficient by omitting any unnecessary terms that do not depend on the parameter of interest.

Following the command word the user must specify,

options	description
distribution	name of the distribution as a string (see the list below)
logp	name of a scalar to which the result is added
value	the observation that follows the distribution. Either a single value or a variable. If a variable the sum of the log-densities is added to logp.
par1	the first parameter of the distribution. Either a number or a variable.
par2	the second parameter if the distribution has one.

The distributions available with logdensity are list below. Here logp stands for the name of a scalar, y stands for the value and the parameters are then listed in order. Parameter names in upper case must be specified as Stata matrices. For details of the parameterizations see the Appendix to *Bayesian Analysis with Stata* or the last section of this manual.

For the binomial distribution the factorial terms are omitted so that p

options	description
Bernoulli	logdensity bernoulli logp y p
Beta	logdensity beta logp y alpha beta
Binomial	logdensity binomial logp y n p
Categorical	logdensity categorical logp y p
Chi-squared	logdensity chisqr logp y k
Dirichlet	logdensity dirichlet logp y alpha
Exponential	logdensity exponential logp y mu
Gamma	logdensity gamma logp y alpha beta
Generalized Gamma	logdensity gengamma logp y alpha beta gamma
Inverse Gamma	logdensity igamma logp y alpha beta
Inverse Gaussian	logdensity igauss logp y mu lambda
Laplace	logdensity laplace logp y mu b
Logistic	logdensity logistic logp y mu b
Log-normal	logdensity lognormal logp y mu sigma
Multinomial	logdensity multinomial logp y p
Multivariate Normal	logdensity mnormal logp y mu SIGMA
Multivariate t	logdensity mt logp y mu SIGMA k
Negative Binomial	logdensity negbin logp y n p k
Normal/Gaussian	logdensity normal logp y mu sd
Pareto	logdensity pareto logp y a c
Poisson	logdensity poisson logp y lambda
Student' t	logdensity t logp y mu sigma k
Uniform	logdensity uniform logp y a b
Weibull	logdensity weibull logp y v mu
Wishart	logdensity wishart logp Y SIGMA k

is treated as a parameter and n is treated as fixed. For the Wishart distribution k is treated as fixed. The multinomial and Dirichlet distributions assume that y, p and alpha are stored in Stata variables.

The Mata versions of logdensity are in the library libmcmc but return the value of the logdensity directly rather than add it to a scalar.

Example

```
scalar lp = 0
logdensity binomial lp y 10 0.2
```

`mcmcac` - plot the autocorrelations of a MCMC chain

Syntax

`mcmcac` *varlist if in* `[ ,` *options* `]`

options	description
`pac`	partial autocorrelation plots
`acoptions(`*string*`)`	options passed directly to ac or pac
`goptions(`*string*`)`	options passed to the twoway command
`cgoptions(`*string*`)`	options passed to graph combine
`save(`*string*`,replace)`	`.dta` file for saving the autocorrelations

Description

`mcmcac` plots the autocorrelations or partial autocorrelations using the Stata commands `ac` or `pac`. If the varlist contains more than one parameter the separate plots are combined within a single plot using `graph combine`.

Options

`pac` Plot partial autocorrelations. Defaults to autocorrelations.
`acoptions(`*string*`)` Options passed directly to the Stata's `ac` or `pac` command.
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.
`save(`*string*`,replace)` Specifies a filename (.dta) for saving the autocorrelations or partial autocorrelations.

`mcmcbgr` - Brooks-Gelman-Rubin plot

Syntax

`mcmcbgr` *varlist if in* , *chain(varname)* `[` *options* `]`

Description

`mcmcbgr` creates Brooks-Gelman-Rubin plots for assessing the convergence of parallel chains. The plots are most useful when the chains start from widely dispersed initial values. As the chains come closer into agreement the variabaility of the pooled chain should be similar to the average variability of the individual chains. The original suggestion was for a plot based on two estimates of the variance of the posterior distribution, one pooling all of the chains and the other averaging the within chain variance. The ratio of the two variances was called R. Doubt was expressed about the interpretation of this measure when the posterior is non-normal and a robust

options	description
`chain(varname)`	Chain identifier - essential
`m`	Number of plotting points
`variance`	Use variance rather than 80% intervals
`iteration(varname)`	Variable for the x-axis
`bychain`	Plot chain-specific variances or intervals
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine
`save(`*string*`,replace)`	`.dta` file for saving the plotting points

alternative was suggested based on the interval between the 10% and 90% centiles, so that R = the ratio of the interval length from all data to the average interval length of the separate chains. In each case R is calculated for increasing chain sizes. For a subchain consisting of the first N values, R is calculated from the second half of the subchain and then plotted against N. R should approach 1 if the chains have converged. The lower plot shows the variance or interval length based on all chains pooled (solid line) and on the average of the subchains (dashed line). These should stabilise into horizontal lines.

Options

`chain(varname)` Variable taking the values 1,2,3…to identify the chains (essential).
`m(#)` Number of plotting points. Defaults to 20.
`variance` Requests a plot based on within and between chain variances rather than the default plot based on 80% intervals
`iteration(varname)` Variable used for the x-axis. If not specified the points are plotted in the order in which they appear in the dataset at points 1,2,3...
`bychain` Show each individual chain in the lower plot, rather than the average across the chains.
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.
`save(`*string*`,replace)` Specifies a filename (`.dta`) for saving the plotting points.

`mcmccheck` - Bayesian graphical model checking

Syntax

`mcmccheck , [ options ]`

options	description
`predictions(`*string*`)`	Variable(s) containing the predicted values
`pfilename(`*string*`)`	File (`.dta`) containing the variables of predicted values
`data(`*string*`)`	Variable or value of the actual observation(s)
`dfilename(`*string*`)`	File (`.dta`)containing the observed values.
`xaxes(`*string*`)`	Variable or function for the x-axis of the plot
`yaxes(`*string*`)`	Variable for the y-axis of the plot
`plot(`*string*`)`	Type of plot
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine
`save(`*string*`)`	File (.dta) for saving the residuals
`koptions(`*string*`)`	Options passed to kdensity
`by(varname)`	Controls symbols in scatter plots
`lcondition(`*string*`)`	Label points that satisfy this condition
`lvarname(varname)`	Variable containing the labels
`nsamples(#)`	Number of samples for plot type epp

Description

`mcmccheck` performs graphical Bayesian model checking based on residuals. The program calculates the mean of the predictive distribution (fit) and five basic measures of surprise or unusualness of an observation

1. residual: difference of a value from its mean prediction divided by the standard deviation of its predictive distribution
2. absresidual: the absolute of the residual in (1)
3. posterior predictive p-value (ppp): tail area under the predictive distribution that is greater than the observation
4. two-tail posterior predictive p-value (ppp): tail area under the predictive distribution corresponding to values more extreme than the observation
5. relative predictive surprise (rps): height of the predictive density at the observed value as a percentage of the maximum height of the density

Options

`predictions(`*string*`)` Variable or stem of a list of variables that contain the predictions. Each entry is treated as an abbreviated name, thus `x_` would imply `x_1`, `x_2` etc.

`pfile(`*string*`)` A Stata data file containing the variables specified in the `predictions` option.

`data(`*string*`)` Single value or a variable containing the actual data.

`dfile(`*string*`)` A Stata data file containing the variables specified in the `data` option.

`plot(`*string*`)` Type of plot required. One of `histogram, density, scatter, lowess, epp, boxplot` or `summary`. If not specified the default is a histogram if the data option contains a single value, or a scatter plot if it contains a variable.

`yaxis(`*string*`)` The measure of surprise to be used in the residual plots. The options (keyword) are; the residual (`residual`), the absolute residual (`absresidual`), the posterior predictive p-value (`ppp`), two tailed ppp (`tppp`) and the percent relative predictive surprise (`rps`). The default is the residual.

`xaxis(`*string*`)` Variable to be used for the x-axis of a plot. The default is the `fit` (mean prediction). May also be a data variable or a Stata function of the fit or data, for example `log10(fit)`.

`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.

`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine` command when using `plot(summary)`.

`save(`*string*`,replace)` A file (`.dta`) for saving the measures of surprise.

`koptions(`*string*`)` Options passed directly to Stata's `kdensity` command when using `plot(density)` or when calculating the relative predictive surprise.

`by(varname)` Categorical variable that controls the plotting symbol in a scatter plot. Each level of the variable is shown by a different symbol.

`lcondition(`*string*`)` A condition, such as `‘res > 2’`, that is applied when deciding whether to label the points on a scatter plot. Only those points that satisfy the condition are labelled.

`lvariable(varname)` Variable containing the values used to label the points on a scatter plot.

`nsamples(#)` Option that controls the number of samples used to construct an empricial probability plot (epp).

`mcmccusum` - cusum plot of MCMC simulations

Syntax

`mcmccusum` *varlist* `ifin [ ,` *options* `]`

options	description
`reference(#)`	number of reference curves
`chain(varname)`	Variable identifying different chains
`iteration(varname)`	Variable for the x-axis
`overlay`	Requests that chains are overlayed on one graph
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine

Description

`mcmccusum` plots the cusum, that is the cumulative sum of deviations about the mean . Useful for detecting early drift in the chain caused by the choice of inital values. If more than one variable is specified then the separate plots will be combined and placed next to one another.

Options

`reference(#)` Number of reference curves added for comparison with the actual data; that is cusum plots for simulated data with the same mean, standard deviation and correlation as the real data but no drift. Defaults to 0.
`chain(varname)` Gives the chain identifier and requests separate cusum plots for each chain. If omitted, it implies a single chain.
`overlay(varname)` Used together with `chain` to request that the cusum plots from each chain be overlayed on a single graph.
`iteration(varname)` Variable used for the x-axis. If not specified the points are plotted in the order in which they appear in the dataset at points 1,2,3...
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.

`mcmcdensity` - plots posterior density estimates

Syntax

`mcmcdensity` *varlist* `ifin [ ,` *options* `]`

Description

`mcmcdensity` plots the smoothed density estimate from an MCMC chain. This command is a wrapper that calls the stata command kdensity. However it will allow bounds to be set. For example, since the posterior of a standard

options	description
`lbounds(`*string*`)`	Lower bounds for parameter ranges
`ubouns(`*string*`)`	Upper bounds for parameter ranges
`koptions(`*string*`)`	Options passed to kdensity
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine
`addplot(`*string*`)`	Stata code for plots to overlay the density
`save(`*string*`,replace)`	File (`.dta`) for saving the plotting points

deviation is known to be positive it is possible to get a smooth density that does not allocate probability to negative values by setting lbound=0 and ubound=. The boundary is imposed by reflecting the data in the limit as described in Silverman(1986).

Options

`lbounds(`*string*`)` Lower bounds for the ranges of each parameter or a missing value if the parameter is unbounded, for instance with three parameters in *varlist*, we might specify `low(.   .   0)` in order to bound the third parameter. If the option is omitted then no lower bounds are set.
`ubounds(`*string*`)` Upper bound for the range of each parameter. See `lbound`.
`koptions(`*string*`)` Options passed directly to Stata's `kdensity` command.
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.
`addplot(`*string*`)` Stata code for creating a plot to overlay the density, for instance, to draw the prior over the posterior.
`save(`*string*`,replace)` File(`.dta`) for saving the plotting points.

`mcmcgeweke` - version of the Geweke test for convergence

Syntax

`mcmcgeweke` *varlist* `ifin [ ,` *options* `]`

options	description
percentages(# #)	lower and upper percentage of the chain to be compared

Description

`mcmcgeweke` is similar to the Geweke convergence test except that it uses prais regression to estimate the standard errors. The test compares the mean of the early part of a chain (default 10%) with the mean of the late part of a chain (default 50%). Means, st errors and p-values are returned. mcmcgeweke is byable, which makes it easy to analyse multple chains.

Options

`percentages`(# #) Lower and upper percentage of the chain to be compared. Defaults to (10 50), that is to compare the first 10% of the chain with the last 50%.

Saved Results

`mcmcgeweke` returns six values for each parameter. For example, for parameter 1 it returns, `m1_1`, the mean of the early section of the chain, `se1_1`, the standard error of the early section of the chain, `m2_1`, the mean of the late section of the chain, `se2_1`, the standard error of the late section of the chain, `z_1`, the test statistic, `p_1`, p-values for test comparing the means.

`mcmcintervals` - interval estimates for successive parts of a chain

Syntax

`mcmcintervals` *varlist* `ifin [ ,` *options* `]`

options	description
`m(#)`	Number of divisions of the chain
`level(#)`	Percentage within the plotted interval
`chain(varname)`	Variable denoting the chain
`iteration(varname)`	Variable defining the x-axis
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine
`save(`*string*`,replace)`	File (`.dta`) for saving the plotting positions

Description

`mcmcintervals` plots interval estimates for m consecutive sections of the chain. Intervals are calculated from the appropriate centiles of that portion of the chain. If varlist contains more than 1 parameter the separate plots are combined. The chain option plots the intervals for multiple parallel chains and superimposes them on the same plot. A horizonal dashed line shows the overall median.

Options

`m(#)` Number of sections into which each chain is divided. Default 10

`level(#)` Percentage within the interval. Defaults to 80%, i.e. between the 10th and 90th centiles.
`chain(varname)` Variable identifying the chains as 1, 2,... Requests separate intervals for each chain.
`iteration(varname)` Variable defining the x-axis. If not specified the points are plotted in the order in which they appear in the dataset at points 1,2,3...
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.
`save(`*string*`,replace)` File (`.dta`) for saving the plotting points.

`mcmclength` - estimates the required run length

Syntax
`mcmclength` *varlist if in* , [*options*]

options	description
`blocksize(#)`	Bootstrap block size
`target(#)`	Target standard error
`ptarget(#)`	Target percentage error
`level(#)`	Size of credible intervals
`bootopts(`*string*`)`	Bootstrap options
`ac(#)`	Autocorrelation threshold

Description
`mcmclength` uses an existing short MCMC run to estimate the total number of updates that would be required to obtain a given accuracy in the mean, standard deviation or credible interval. Current accuracy (standard error) is assessed using a blocked bootstrap. The target accuracy is expressed either by specifying a standard error directly (target) or by specifying the standard error as a percentage of the estimated mean (ptarget). The number of updates is scaled proportionately depending on the current and target standard errors.

Options
`blocksize(#)` Size of the bootstrap blocks. Defaults to include autocorrelations over 0.05.
`target(#)` Target standard error. Defaults to use `ptarget`.
`ptarget(#)` Target standard error as a percentage of the mean. Defaults to 5, that is the target standard error is always 5% of the mean.

`level(#)` Size of credible intervals.
`bootopts(`*string*`)` Options passed directly to Stata's `bootstrap` command.
`ac(#)` Set blocksize by specifying a limit on the autocorrelation. Defaults to 0.05.

`mcmcmahal` - Mahanalobis Distance Plot

Syntax

`mcmcmahal` *varlist if in* `, [` ***options*** `]`

options	description
`distance(`*newvarname*`)`	Variable to store the Mahanalobis distances
`replace`	Replace the distance variable if it already exists
`noplot`	Calculations but no graphs
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine

Description

`mcmcmahal` is intended for checking the convergence of a set of parameters simulated by an MCMC algorithm and stored in varlist. The Mahalanobis distance is a squared measure of the distance between an individual set of simulated parameters and the mean of the parameters over all simulations. The measure allows for correlation between the parameters. The command creates a combined plot containing trace (`mcmctrace`) and cusum (`mcmccusum`) plots of the distance, a chi-squared (`qchi`) plot and an intervals plot (`mcmcintervals`).

Options

`distance(`*newvarname*`)` Variable to contain the calculated distances
`replace` Permission to over-write the `distance` variable if it already exists
`noplot` Calculate the distances but do not plot them.
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.

`mcmcsection` - comparing posterior densities

Syntax

`mcmcsection` *varlist if in* `[ ,` *options* `]`

Description

`mcmcsection` plots smoothed density estimates for the whole MCMC chain

options	description
`m(#)`	Number of sections
`chain(varname)`	Plot chains rather than sections of one chain
`koptions(`*string*`)`	Options passed to kdensity
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine
`nod`	do not display the measure D
`save(`*string*`,replace)`	File (.dta) for saving the plotting points

(solid line) and for m fractions of the chain (dashed lines). For instance, m=3 plots three smoothed densities for the first, middle and last thirds of the chain. Densities smoothed using `kdensity`. The measure D represents the maximum difference of two sectional densities as a percentage of the maximum height of the density of the whole chain. $D < 20$ usually looks like reasonable agreement, $D < 10$ is good. If varlist contains more than 1 parameter the separate plots are created and then combined.

Options

`m(#)` Number of sections. Defaults to 2.
`chain(varname)` Chain identifier. Requests separate smoothed densities for each chain rather than by section of a single chain. The option `m` is ignored.
`koptions(`*string*`)` Options passed directly to Stata's `kdensity` command.
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.
`nod` The measure of separation `D` is not shown on the graphs.
`save(`*string*`,replace)` File (`.dta`) for saving the plotting points.

`mcmcstats` - summary statistics for an MCMCrun

Syntax
`mcmcstats` *varlist if in* `[ ,` *options* `]`
Description
`mcmcstats` writes a table summarising the MCMC results for varlist including the number of observations, mean, standard deviation, standard error, median and credible interval. The standard error is calculated using prais regression. mcmcstats is byable and returns the calculated summary statistics.

Options

options	description
`hpd`	Highest posterior density intervals instead of credibility intervals
`level(#)`	Percentage for the interval estimates. Defaults to 95
`correlations`	Show correlation matrix
`covariances`	Show covariance matrix
`save(`*string*`)`	File (.dta) for saving the summary statistics.

`hpd` Highest Posterior Density intervals are displayed rather than credible intervals. The calculation is slower as it requires a density estimate and it could fail completely with multi-modal distributions as the HPD interval could consist of several disjoint regions.
`level(#)` Percentage for the interval estimate. Defaults to 95.
`correlations` Show the matrix of correlations between the parameters.
`covariances` Show the matrix of covariances between the parameters.
`save(`*string*`,replace)` File (`.dta`) for saving the summary statistics.

Saved Results
For each parameter `mcmcstats` returns eight values. Thus for parameter one in the list it returns; the parameter name, `r(par1)`, and then the statistics `r(n1)`, `r(mn1)`, `r(sd1)`, `r(se1)`, `r(md1)`, `r(lb1)` and `r(ub1)`. When requested for display the correlation and covariance matrices are returned as `r(C)` and `r(V)`.

`mcmctrace` - trace plot of a set of MCMCsimulations

Syntax
`mcmctrace` *varlist if in* `[ ,` *options* `]`

options	description
`chain(varname)`	Variable identifying the chain
`overlay`	Overlay chains on a single plot
`iteration(varname)`	Variable defining the x-axis
`goptions(`*string*`)`	Options passed to the twoway command
`cgoptions(`*string*`)`	Options passed to graph combine

Description
`mcmctrace` plots the consecutive values from an MCMC run as a time series. The plot also shows the median and the 95% credible interval. If more than

one variable is specified then the separate plots will be combined and placed next to or above one another. If there is more than one chain the plots for the separate chains can either be displayed separately or overlayed onto one plot.

Options

`chain(varname)` Chain identifier. Requests separate traces for each chain.
`overlay` Used together with `chain` to request that the traces from each chain be overlayed on a single plot.
`iteration(varname)` Variable used for the x-axis. If not specified the points are plotted in the order in which they appear in the dataset at points 1,2,3...
`goptions(`*string*`)` Options passed directly to Stata's `twoway` command.
`cgoptions(`*string*`)` Options passed directly to Stata's `graph combine`.

`mcmcrun` - controls the creation of MCMCsimulations

Syntax

`mcmcrun` *logpost [X] b* `[using]` `[`*fweight*`]` *if in*, `[` ***options*** `]`

options	description
`samplers(`*string*`)`	List of samplers
`burnin(#)`	Length of the burn-in
`updates(#)`	Length of the chain
`thin(#)`	Thinning number
`adapt`	Adapt mhssamplers during the burn-in
`parameters(`*string*`)`	Names of the parameters
`predictions(`*string*`)`	Name of a function that calculates the predictions
`replace`	Replace the output file if it exists
`append`	Append to the output file
`jposterior`	Do not recalulate the current log-posterior
`savelogp`	Add log-posterior to the output file
`nodots`	Do not show dots to monitor progress
`mata`	Use Mata rather than Stata
`data(`*string*`)`	Data to be transferred to Mata

Description

`mcmcrun` is a program for controlling the calculation of MCMC simulations in Stata or Mata. It is simply a house-keeping program that calls samplers specified by the user and writes the updated parameter values to the comma delimited (.csv) text file as specified by using. Following the command word the user must give the name of the program that calculates the log-posterior (logpost) and the name of the row vector that contains the initial values of the parameters(b). When using Mata the list must also include the name of the data matrix that is passed to the user's program for calculating the log-posterior (X).

Options

`samplers(`*string*`)` Essential option. A sampler must be specified for each parameter. Each chosen sampler is placed in brackets that must contain a keyword that denotes the sampler followed by the options needed to run that sampler. Recognised keywords are `logn norm mnorm` and `trnc` which refer to the corresponding mhssamplers and `arms ars griddy` and `slice`. Specifying `mhslogn` is equivalent to `logn`. `skip` requests that that parameter is not updated, so that it is left with its initial value. Any other keyword it assumed to refer to a sampler written by the user.
`burnin(#)` Length of the burn-in. The burn-in is not saved in the output file. Defaults to 0.
`updates(#)` Length of the chain. Defaults to 1000.
`thin(#)` Thinning number. Defaults to 1, i.e. no thinning
`adapt` Adapt mhssamplers during the burn-in
`parameters(`*string*`)` Names for the parameters separated by spaces. Dashes are allowed to denote a series of parameter names, for example, `a1-a5` implies `a1 a2 a3 a4 a5`.
`predictions(`*string*`)` Name of a function that calculates the predictions and returns them as a row vector as r(pred). Predictions are added to the output file under the same name as function. Thus if the function creating the predictions is called `ys` then the predictions will be added to the output file under the names `ys1, ys2, ...`
`replace` Replace the output file if it already exists
`append` Append to the output file
`jposterior` Do not recalulate the current log-posterior but take the last calculated value as referring to the current point. Useful when the user's function calculates the full log-posterior rather than the log-conditional posterior,
`savelogp` Add the final log-posterior calculated in any cycle to the output file
`nodots` Do not show dots to monitor progress. In Mata the dots can slow down the program noticeably for single core versions of Stata.
`mata` Use Mata rather than Stata. Requires the user to write their functions

in Mata.
`data(`*string*`)` Data matrices to be transferred to Mata. These matrices are copied from Stata to Mata under the same names. The notation `X=(x1 x2 x3)` can be used to copy three Stata variables `x1`, `x2` and `x3` and save them as a Mata matrix `X`.

writing your own samplers

A sampler with an unrecognised name, i.e. not skip or one of the mhs or gbs samplers, will be assumed to be a sampler written by the user. This might refer to the user's Gibbs sampler for updating a standard distribution or it might be that the user has written their own general purpose sampler. The user's program is called followed by the name of the program for calculating the log-posterior, the name of the row vector containing the parameter values and ipar, the number of the first parameter to be updated. The if and in conditions and any options are passed directly to the user's program with the exception of the option `dim(#)`, which is taken as giving the dimension of the update, i.e. the number of consecutive parameters updated by the user's program. If omitted it is assumed that the program updates a single parameter. `dim` is not passed to the user's program. For samplers written in Mata the arguments are in order, the current value of the log-posterior, a pointer to the function that calculates the log-posterior, the data matrix, the row vector of parameter values and a scalar giving the number of the parameter to be updated. These are followed by any other options in the order in which they are specified by the calling program. Whether you use Stata or Mata you will need to include the argument for the log-posterior function or a dummy replacement if it is not used.

`mhslogn` - MH sampler for a parameter defined over $(0,\infty)$

Syntax

`mhslogn` *logpost b ipar [weight] if in*`, [` *options* `]` **Description**

options	description
`sd(#)`	standard deviation of the proposal distribution
`logp(#)`	the value of log-posterior at the current parameter value

`mhslogn` uses the Metropolis-Hastings algorithm to create a single update of a single parameter. As it uses a log-normal proposal distribution it is suitable for a parameter, such as a variance, that can only take positive values.

Repeated calls to `mhslogn` can be used to create an MCMC algorithm and hence estimate the posterior distribution of the parameter. The efficiency but not the validity of the algorithm is affected by the choice of standard deviation for the proposal. Values between a half and three-quarters of the size of the range of the posterior (on a log scale) usually work well. It is not valid to use previous results from the same chain to alter the standard deviation of the proposal distribution.

Options

The user must supply the name of a program that evaluates the log-posterior (logpost); this program must have the structure described in `mhsnorm`. Parameters are passed in a row vector (b). The number of the parameter that is to be updated is given by ipar.

`sd(#)` Standard deviation for the proposal distribution (on the log-scale)
`logp(#)` Current value of the log-posterior. Evaluated is not given.

Mata

To call the Mata version requires,
mhslogn(*real scalar* logp,*pointer function* pf,*real matrix* X,*real rowvector* theta,*real scalar* ipar,sd[rs] | accept[rs])

`mhsnorm` - MH sampler for a parameter defined over $(-\infty,\infty)$

Syntax

`mhsnorm` *logpost b ipar [weight] if in*, `[` *options* `]` **Description**

options	description
`sd(#)`	standard deviation of the proposal distribution
`logp(#)`	the value of log-posterior at the current parameter value

`mhsnorm` uses the Metropolis-Hastings algorithm to create a single update of a single parameter using a Gaussian (normal) proposal distibution. Since the proposals may take any value this method is suited to parameters that are defined between plus and minus infinity. Repeated calls to `mhsnorm` can be used to create an MCMC algorithm and hence estimate the posterior distribution of a set of parameters. The efficiency but not the validity of the algorithm is affected by the choice of standard deviation of the proposal distribution. Values between a half and three-quarters of the effective range of the posterior usually work well. It is not valid to use previous results from the same chain to alter the standard deviation of the proposal distribution.

Options

The user must supply the name of a program that evaluates the log-posterior (logpost); see below. Parameters are passed in a row vector (b). The number of the parameter that is to be updated is given by ipar.

`sd(#)` Standard deviation for the proposal distribution
`logp(#)` Current value of the log-posterior. Evaluated if not given.

Structure for the user's program The general form of the program must be:

```
program  logpost
   syntax anything [fweight] [if] [in]

   marksample touse
   tokenize `"`anything'"'
   local logp "`1'"
   local b    "`2'"
   local ipar = "`3'"

   local alpha = `b'[1,1]
   local beta  = `b'[1,2]
   ....
      calculation of the log of the posterior
   ....
end
```

If frequency weights and if and in conditions are not required then this can be simplified to

```
program  logpost
   args logp b ipar

   local alpha = `b'[1,1]
   local beta  = `b'[1,2]
   ....
   calculation of the log of the posterior
   ....
end
```

where logp is a scalar that will eventually contain the calculated value of the log posterior, b is the row vector of parameters and ipar is the number of the parameter being updated.

The calculation must return the value of the log posterior although constants in the summation that produces this value can be omitted. Thus only terms that depend on the parameter being updated need to be included. Some gain in efficiency is often possible by altering the calculation depending on which paramter is being updated. For instance,

```
program  logpost
   args logp b ipar

   local alpha = `b'[1,1]
   local beta  = `b'[1,2]
   ....
   if `ipar' == 1 {
      calculation of the log of the conditional posterior for parameter 1
   }
   else ...
      ....
 }
end
```

Mata

To call the Mata version requires,
mhsnorm(*real scalar* logp,*pointer function* pf,*real matrix* X,*real rowvector* theta,*real scalar* ipar,sd[rs] | accept[rs])

`mhstrnc` - MH sampler for a parameter defined over a truncated range

Syntax

`mhstrnc` *logpost b ipar [weight] if in*, [*options*] **Description**

`mhstrnc` uses the Metropolis-Hastings algorithm to create a single update of a single parameter. As it uses a logistic transform, it is suitable for parameters that can only take values over a bounded range; for instance, a proportion defined over the range [0,1]. If the range of the parameter is [a,b] then

options	description
`sd(#)`	standard deviation of the proposal distribution
`lbound(#)`	lower limit of the range of the posterior
`ubound(#)`	upper limit of the range of the posterior
`logp(#)`	the value of log-posterior at the current parameter value

the normal proposal is generated on the transformed scale log[(y-a)/(b-y)]. Repeated calls to `mhstrnc` can be used to create an MCMC algorithm and hence estimate the posterior distribution of the parameter. The efficiency but not the validity of the algorithm is affected by the choice of standard deviation for the proposal. Values between a half and three-quarters of the size of the range of the posterior (on the transformed scale) usually work well. It is not valid to use previous results from the same chain to alter the standard deviation of the proposal distribution.

Options

The user must supply the name of a program that evaluates the log-posterior (logpost); this program must have the structure described in `mhsnorm`. Parameters are passed in a row vector (b). The number of the parameter that is to be updated is given by ipar.

`sd(#)` Standard deviation for the proposal distribution (on the logistic scale)
`lbound(#)` Lower limit of the parameter's range
`ubound(#)` Upper limit of the parameter's range
`logp(#)` Current value of the log-posterior. Evaluated is not given.

Mata

To call the Mata version requires,
mhstrnc(*real scalar* logp,*pointer function* pf,*real matrix* X,*real rowvector* theta,*real scalar* ipar,sd[rs],lbound[rs],ubound[rs] | accept[rs])

`mhsmnorm` - sampler for a multi-dimensional parameter

Syntax

`mhsmnorm` *logpost b ipar [weight] if in*, `[` *options* `]` **Description**
`mhsmnorm` uses the Metropolis algorithm to create a single update of a block of parameters. It uses a multivariate Gaussian proposal distribution and is suitable for parameters that can take any real value although it generally

options	description
`cholesky(name)`	Cholesky decomposition of the variance matrix
`logp(#)`	the value of log-posterior at the current parameter value

works well for parameters with restricted ranges provided that their posterior is well away from the bounds. Repeated calls to `mhsmnorm` can be used to create an MCMC algorithm and hence estimate the posterior distribution of the set of parameters. The efficiency but not the validity of the algorithm is affected by the choice of matrix given in the option cholesky. This matrix represents the cholesky decomposition of the variance-covariance matrix of the proposal distribution. It is not valid to alter this matrix based on previous simulations from the same chain.

Options

The user must supply the name of a program that evaluates the log-posterior (logpost); this program must have the structure described in `mhsnorm`. Parameters are passed in a row vector (b). The number of the parameter that is to be updated is given by ipar.

`cholesky(name)` Cholesky decomposition of the variance matrix of the proposal distribution
`logp(#)` Current value of the log-posterior. Evaluated is not given.

Mata

To call the Mata version requires,
mhsmnorm(*real scalar* logp,*pointer function* pf,*real matrix* X,*real rowvector* theta,*real scalar* ipar,cholesky[rm] | accept[rs])

`wbsarray` - write a rectangular dataset in WinBUGS format

Syntax

`wbsarray` *varlist* [using] *if in* [, *options*]

option	description
`formats(`*string*`)`	Formats for the data
`names(`*string*`)`	Variable names for use in WinBUGS
`replace`	Replace an existing output file

Description

`wbsarray` writes a WinBUGS array to the results window and/or a text file with the filename given by *using*. Useful for rectangular sets of variables with equal length. WinBUGS can read more than one data file so it would be possible to put other arrays, scalars etc into a different file.

Options

`formats(`*string*`)` Gives the format used for writing the data. Defaults to %8.3f. Formats cycle so that `f(%4.0f %3.1f)` would write the variables using %4.0f %3.1f %4.0f ...

`names(`*string*`)` Alternative names for the variables. Enables a variable to be called by a different name in WinBUGS to that used in Stata. For example, `names(a b)` would create two WinBUGS vectors named `a[]` and `b[]`, and `name(A[,1] A[,2])` would create two columns of a WinBUGS matrix called A.

`replace` Replace the output file if it already exists.

`wbscoda` - read data from Coda formatted files

Syntax

`wbscoda [using],` *clear* `[` *options* `]`

options	description
`clear`	Clear existing data from Stata (essential)
`chains(`*numlist*`)`	Chain identifier
`keep(`*string*`)`	List of parameters to be kept
`openbugs`	Read files created by OpenBUGS

Description

Reads a file of CODA formatted data saved from a WinBUGS or OpenBUGS analysis. The root of the filename is given by *using*. The command will read multiple chains if requested. The MCMC simulations produced by WinBUGS are output in the format required by the convergence checking program `CODA`. An index file is created listing the parameter and the posiions in the data file where the results are stored. Data files are created, one for each chain, containing the simulated values. The formatting is awkward and requires this special command in order to read the data into Stata.

WinBUGS programmers often use a full stop as part of a parameter name as in y.sigma. Variable names in Stata cannot contain full stops and so the name is changed to `y_sigma` when the data are read into Stata. When vectors of parameters are saved by WinBUGS the names are also editted; thus `alpha[2]` would become `alpha_2` in Stata. In theory this would result in WinBUGS parameters `alpha.2` and `alpha[2]` being given the same name in Stata, which would cause `wbscoda` to fail. Avoiding this problem within `wbscoda` would result in rather ugly or unpredictable variable names, so the responsibility is left with the user to avoid such clashes when choosing their WinBUGS parameter names.

Options

`clear` Clear any existing data from Stata.
`chains(`*numlist*`)` Numbers of the chains that are to be read. Only relevant when multiple chains were run in WinBUGS or OpenBUGS. Defaults to read a run with a single chain.
`keep(`*string*`)` Parameters to keep. Only those parameters will be read from the CODA file. Specify the names as used by WinBUGS. Any match to the start of the name will be read. So `keep(a)` would read `alpha`, `a[1]`, `a[2]`, etc. Defaults to read all parameters.
`openbugs` Files created by OpenBUGS. Defaults to WinBUGS.

`wbsdecode` - read lists of WinBUGS data into Stata

Syntax

`wbsdecode using , clear [`*options*`]`

options	description
`clear`	Permission to replace the current data in Stata (essential)
`array`	Read an array rather than a list

Description

`wbsdecode` reads data from a text file when the data are stored in WinBUGS format, that is, as a list structure or an array. If the data are part of a compound document they must be copied to a text file before using wbsdecode. wbsdecode is useful for reading the data from the examples supplied with WinBUGS so that the data can be processed in Stata, but it would also read data from R or S-plus structures. Any existing data in Stata will be lost so you must specify the clear option.

Options

`clear` Overwrite the current data in Stata (essential).

`array` Read a WinBUGS array. Defaults to reading a list.

Warning `wbsdecode` reads list structures by searching for the next "=" and then locating the variable name that precedes it and the structure type that follows it. Spreading this information over more that one line would cause wbsdecode to misread the data. So
y = c(
1 ,2 ,3)
is OK. But
y
= c(1, 2, 3)
would fail

`wbslist` - write a list of data or initial values

Syntax

`wbslist` *wbsargs* [*using*] `[ ,` *options* `]`

options	description
`replace`	replace the output file if it exists

Description

`wbslist` writes data or initial values to a text file in an R-like list structure suitable for reading into WinBUGS. The data may be any combination of Stata scalars, variables, matrices, two-way structures created from Stata variables, multi-way tables or values typed directly into the command by the user.

WinBUGS can also read arrays of data, but these must be created using the wbarray command and stored in a separate file. WinBUGS can read multiple data files.

Following the command word the separate components of the list are place in brackets (). The contents of each bracket may begin with one of the keywords scalar, vector, matrix or structure (variable is an allowed as a synonym for vector and the command words may be abbreviated). A structure is a matrix within WinBUGS that is created from a set of Stata variables. If the contents of the bracket do not begin with any of these keywords then the contents are treated as text and are written directly to the output file. Thus a call to wbslist might be, cmd . wbslist (mat R S) (struc y1 y2 y3, name(Y)) using data.txt, replace where R and S are Stata matrices and y1, y2 and y3 are Stata variables.

Tables are mutli-way structures created by specifying in order , a variable containing the table contents and then one variable for each dimension containing values 1,2,... If there is more than one entry for a particular cell their sum is plac

Options
`replace` overwrite the output file if it already exists.

Suboptions
These are options placed with the brackets `()` that control the listing of each component.
`formats(`*string*`)` Specifies the formats used for writing the data. See `wbsarray`.
`linesize(#)` Number of values per line in the output file. Defaults to 10.
`names(`*string*`)` Names to be used in the WinBUGS data file. Defaults to use the Stata name. In this way data may have a different name in WinBUGS to that used in Stata. If used the number of names must match the number of items. When a structure is formed from several Stata variables it must be given a single name, otherwise renaming is optional.

`vector`, & `structure` also allow 'if' and 'in' qualifiers to limit the data that are written to the output file. These are placed within the appropriate set of brackets, as in

```
wbslist (vector x in 1/5, name(x1)) (vector x in 6/10, name(x2))
using data.txt, replace
```

Writing Strings
When the brackets do not start with a keyword, the contents are simply written to the output file. However, prior to writing, two expansions are applied based on curly {} and square `[]` brackets. These features are primarily intended for generating initial values. {} are repeatedly evaluated prior to writing. So,

```
   wbslist (alpha=c(5{0})) using inits.txt, replace
```

would write a list containing `alpha=c(0,0,0,0,0)` and

```
   wbslist (alpha=c(5{rnormal()})) using inits.txt, replace
```

would write a list containing 5 different values generated by evaluating the Stata command `rnormal()`.
Square brackets refer to the contents of a Stata variable. So,

`wbslist (alpha=c(5{a[3]})) using inits.txt, replace`
would write a list in which alpha had 5 values taken from rows 3, 4, 5, 6 and 7 of the Stata variable a.

`wbsmodel` - copy WinBUGS model file

Syntax
`wbsmodel` *thisfile modelfile* [*identifier*]

Description
`wbsmodel` copies some WinBUGS syntax commands to a text file (model file). It was originally called `tomodel`. It provides a record of the exact model file that was used when calling WinBUGS from within Stata. The WinBUGS code must follow the wbsmodel command line and be commented out using /* ... */. In this way the WinBUGS code is copied to the model file but the commenting stops Stata from trying to run those lines. The program works by re-reading the do file until this tomodel command is found and then its writes anything that follows and is commented out to the model file. Specifying the name of the model file will usually uniquely define the command line that marks the starting position. However, if you have a do file with several inserted sections of WinBUGS code and you choose to write them to model files with the same name then it will need a further identifier to uniquely determine the correcting starting position.

Options
`thisfile` Name (and path) of the current do file that contains the tomodel command and the WinBUGS code. If the path contains spaces then enclose in quotes.
`modelfile` Name (and path) of the model file that is to contain the WinBUGS code. If the path contains spaces then enclose in quotes. Automatically overwrites any existing file with the same name.
`identifier` Any text that unqiuely determines this wbsmodel command line. Usually unnecessary as the modelfile will be enough to determine the start position of the WinBUGS code. However, if two blocks of code are written to files that have the same name then place an identifer at the end of the line to distinguish the two calls to wbsmodel.

`wbsrun` - run WinBUGS from with Stata

Syntax
`wbsrun` [using], [*options*]

Description

options	description
Optional	
executable(*string*)	Path to the executable
openbugs	Run OpenBUGS rather than WinBUGS
background	Run WinBUGS in the background
headless	Do not display the OpenBUGS interface.

`wbsrun` executes a WinBUGS script. If the script ends with quit() then WinBUGS closes after the script has executed and control is returned to Stata. Without quit() in the script file, WinBUGS remains open after the commands have executed and may be used interactively. When interactively closed, control will return to Stata. To save giving the path to WinBUGS each time a new program is run, the location can be saved in a file called `executables.txt` in the PERSONAL folder. `wbsrun` detects Unix/Linux and modifies its operation according. When Unix is found openbugs is assumed.

Options

`executable(`*string*`)` Path to the WinBUGS or OpenBUGS executable. Only needed if the path is not stored in the `executables.txt` file.
`openbugs` Run OpenBUGS rather than WinBUGS.
`background` Run in the background returning control to Stata. The default is for Stata to freeze until WinBUGS has finished.
`headless` Do not show the OpenBUGS interface (OpenBugs only).

`wbsscript` - write a WinBUGS script file

Syntax

`wbsscript [using] ,` *options*

Description

writes a script file for fitting a model in WinBUGS or OpenBUGS. The scripting languages of WinBUGS and OpenBUGS are different so script files cannot be used interchangeably. The ado file checks the operating system and will write scripts using Unix style paths if Unix is detected.

Options

`path(`*string*`)` path placed before the data, init, model, coda and log files within the script. This option saves typing the same path for every file

options	description
modelfile(*string*)	Name of the text file containg the model
datafile(*string*)	Name(s) of the data file(s)
initsfile(*string*)	Name of the initial value files
codafile(*string*)	Root of the name for the coda files
set(*string*)	Parameters to be saved
logfile(*string*)	Name for saving the log file
burnin(#)	Length of the burn-in
updates(#)	Length of the MCMCrun
thin(#)	Save every nth simulation
dic	Write the DIC to the log file
noquit	Close WinBUGS on completion
path(*string*)	Path to the folder containing the input files
seed(#)	Starting point for the random number generator (OpenBUGS only)
openbugs	Create an OpenBUGS script file
replace	Replace the script file if it exists

when they are stored together in the same folder. It is important to give the full path names for every file because when WinBUGS runs you cannot assume that it will start in the same working directory as you are using in Stata. If `path` is omitted then all files without an explicit path will be assumed to be in the current working directory and the full path will automatically be added.

`modelfile(`*string*`)` Name of a text file containing the WinBUGS model.

`datafile(`*string*`)` Names of the data file(s). When there are several data files they should all be included in this option separated by +'s.

`initsfile(`*string*`)` Name of the file(s) of initial values. The names of the initial values files should be separated by '+' as with data files, for intance, `( in1.txt+int2.txt+int3.txt)` When multiple initial values files are specified a separate chain is run for each set of initial values. A gen.inits command is automatically placed after the reading of inital values so that WinBUGS will try to randomly generate inital values for any parameters not explicitly mentioned in the initsfiles. WinBUGS is not always able to generate initial values and in rare circumstances this can cause a program to fail.

`codafile(`*string*`)` String to form the start of the names of the coda files. WinBUGS outputs an index file and data file(s) (one for each chain) with names derived from this root (see `wbscoda`).

`set(`*string*`)` List of parameters that are to be saved.

`logfile(`*string*`)` Name for saving the log file, a record of the steps in the WinBUGS analysis.
`burnin(#)` Length of the burnin, that is, the initial set of simulations that are discarded. Defaults to 0 (no burn-in).
`updates(#)` Length of the chain(s). Defaults to 1000.
`thin(#)` Save every nth simulation. Defaults to zero, no thinning.
`seed(#)` Select a different run of random numbers. Values between 1 and 14. Defaults to 1. OpenBUGS only.
`dic` Write the deviance information criterion to the logfile.
`openbugs` Prepare a script for use in OpenBUGS. Defaults to WinBUGS commands.
`noquit` Do not quit WinBUGS/OpenBUGS when the analysis is completed.
`replace` Permission to erase the script file if it already exists.

Parameterization used in logdensity

Bernoulli

$$p(y;p) = p^y(1-p)^{1-y} \qquad y = 0,1 \quad 0 \le p \le 1$$

Beta

$$p(y;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} y^{\alpha-1}(1-y)^{\beta-1} \qquad 0 \le y \le 1, \quad 0 < \alpha,\beta$$

Binomial

$$p(y;p,n) = \frac{n!}{y!(n-y)!} p^y(1-p)^{n-y} \qquad y = 0,1,\ldots,n, \quad 0 \le p \le 1$$

Categorical

$$p(y) = p_y \qquad y = 1,2...,n \quad \sum_{y=1}^{n} p_y = 1$$

Chi-squared

$$p(y;k) = \frac{y^{k/2-1} exp(-y/2)}{2^{k/2}\Gamma(k/2)} \qquad 0 < y, \quad 0 < k$$

Dirichlet

$$p(y[];\alpha[]) = \frac{\Gamma(\sum_{i=1}^{n} \alpha_i)}{\prod_{i=1}^{n} \Gamma(\alpha_i)} \prod_{i=1}^{n} y_i^{\alpha_i-1} \qquad 0 < y_i < 1, \quad \sum_{i=1}^{n} y_i = 1, \quad 0 < \alpha_i$$

Exponential

$$p(y;\mu) = \frac{exp(-y/\mu)}{\mu} \qquad 0 < y, \quad 0 < \mu$$

Gamma

$$p(y;\alpha,\beta) = \frac{y^{\alpha-1} exp(-y/\beta)}{\beta^\alpha \Gamma(\alpha)} \qquad 0 < y \quad 0 < \alpha,\beta$$

Generalized Gamma

$$p(y;\alpha,\beta,\gamma) = \frac{\gamma y^{\alpha-1} exp(-[y/\beta]^\gamma)}{\beta^\alpha \Gamma(\frac{\alpha}{\gamma})} \qquad 0 < y \quad 0 < \alpha,\beta,\gamma$$

Inverse Gamma

$$p(y;\alpha,\beta) = \frac{\beta^{\alpha} y^{-\alpha-1}}{\Gamma(\alpha)} exp(-\beta/y) \qquad 0 < y \quad 0 < \alpha, \beta$$

Inverse Gaussian

$$p(y;\lambda,\mu) = \left[\frac{\lambda}{2\pi y^3}\right]^{1/2} \exp\left\{\frac{-\lambda(y-\mu)^2}{2\mu^2 y}\right\} \qquad y > 0 \quad 0 < \lambda, \mu$$

Laplace

$$p(y;\mu,b) = \frac{1}{2\phi} \exp\left(\frac{-|y-\mu|}{\phi}\right) \qquad -\infty < y < \infty, \quad 0 < \phi$$

Logistic

$$p(y;\mu,\sigma) = \frac{\exp\left(-\frac{y-\mu}{\sigma}\right)}{\sigma\left[1+\exp\left(-\frac{y-\mu}{\sigma}\right)\right]^2} \qquad -\infty < y, \mu < \infty, \ \ \sigma > 0$$

Log-normal

$$p(y;\mu,\sigma) = \frac{1}{y\sigma\sqrt{2\pi}} \exp\left\{-\frac{(\log(y)-\mu)^2}{2\sigma^2}\right\} \qquad y > 0, \quad -\infty < \mu < \infty, \quad 0 < \sigma$$

Multinomial

$$p(y;p,n) = \frac{n!}{\prod_i y_i!} \prod_i p_i^{y_i} \qquad \sum_i y_i = n \qquad \sum p_i = 1 \quad 0 < p_i < 1 \quad y_i = 0, 1..., n \ \ i = 1...k$$

Multivariate Normal

$$p(y;\mu,\Sigma) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(y-\mu)'\Sigma^{-1}(y-\mu)\right]$$

Multivariate Student-t

$$p(y;\mu,\Sigma) = \frac{\Gamma((k+p)/2)}{\Gamma(k/2)(k\pi)^{p/2}|\Sigma|^{1/2}} \left[1+(y-\mu)'\Sigma^{-1}(y-\mu)/k\right]^{-(k+p)/2}$$

Negative Binomial

$$p(y;p,k) = \frac{(y+k-1)!}{y!(k-1)!} p^k (1-p)^y \qquad y = 0, 1, \ldots, \quad 1 \leq r, \quad 0 < p < 1$$

Normal

$$p(y;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{1}{2\sigma^2}(y-\mu)^2\right] \qquad -\infty < y < \infty \quad -\infty < \mu < \infty, \quad 0 < \sigma$$

Pareto

$$p(y;a,c) = ac^a y^{-(a+1)} \qquad c \leq y \quad 0 < a$$

Poisson

$$p(y;\lambda) = exp(-\lambda)\frac{\lambda^y}{y!} \qquad y = 0, 1, \ldots, \quad 0 < \lambda$$

Student-t

$$p(y;\mu,\sigma,k) = \frac{\Gamma\left(\frac{k+1}{k}\right)}{\sigma\sqrt{k\pi}\Gamma\left(\frac{k}{2}\right)}\left(1+\frac{(y-\mu)^2}{k\sigma^2}\right)^{-\frac{k+1}{2}} \qquad 0 < a$$

Uniform

$$p(y;\alpha,\beta) = \frac{1}{\beta-\alpha} \quad \alpha < y < \beta$$

Weibull

$$p(y;\upsilon,\mu) = \frac{\upsilon(y/\mu)^{\upsilon-1}}{\mu}\exp(-(y/\mu)^{\upsilon}) \qquad y > 0, \quad 0 < \upsilon, \mu$$

Wishart

$$p(Y;S,k) = \frac{|Y|^{(k-p-1)/2}}{2^{kp/2}\Gamma(k/2)|S|^{k/2}}\exp\left[-\frac{1}{2}Tr(S^{-1}Y)\right] \qquad p \leq k$$