

THE STATA JOURNAL

Editor

H. Joseph Newton
Department of Statistics
Texas A & M University
College Station, Texas 77843
979-845-3142; FAX 979-845-3144
jnnewton@stata-journal.com

Associate Editors

Christopher F. Baum
Boston College

Rino Bellocco
Karolinska Institutet, Sweden and
Univ. degli Studi di Milano-Bicocca, Italy

A. Colin Cameron
University of California–Davis

David Clayton
Cambridge Inst. for Medical Research

Mario A. Cleves
Univ. of Arkansas for Medical Sciences

William D. Dupont
Vanderbilt University

Charles Franklin
University of Wisconsin–Madison

Joanne M. Garrett
University of North Carolina

Allan Gregory
Queen's University

James Hardin
University of South Carolina

Ben Jann
ETH Zürich, Switzerland

Stephen Jenkins
University of Essex

Ulrich Kohler
WZB, Berlin

Stata Press Production Manager

Stata Press Copy Editor

Editor

Nicholas J. Cox
Department of Geography
Durham University
South Road
Durham City DH1 3LE UK
n.j.cox@stata-journal.com

Jens Lauritsen
Odense University Hospital

Stanley Lemeshow
Ohio State University

J. Scott Long
Indiana University

Thomas Lumley
University of Washington–Seattle

Roger Newson
Imperial College, London

Marcello Pagano
Harvard School of Public Health

Sophia Rabe-Hesketh
University of California–Berkeley

J. Patrick Royston
MRC Clinical Trials Unit, London

Philip Ryan
University of Adelaide

Mark E. Schaffer
Heriot-Watt University, Edinburgh

Jeroen Weesie
Utrecht University

Nicholas J. G. Winter
University of Virginia

Jeffrey Wooldridge
Michigan State University

Lisa Gilmore
Gabe Waggoner

Copyright Statement: The Stata Journal and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the Stata Journal.

The articles appearing in the Stata Journal may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the Stata Journal.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the Stata Journal, in whole or in part, on publicly accessible web sites, file servers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the Stata Journal or the supporting files understand that such use is made without warranty of any kind, by either the Stata Journal, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the Stata Journal is to promote free communication among Stata users.

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press. Stata and Mata are registered trademarks of StataCorp LP.

Bayesian analysis in Stata with WinBUGS

John Thompson, Tom Palmer, and Santiago Moreno
Department of Health Sciences
University of Leicester
Leicester, UK
john.thompson@le.ac.uk

Abstract. WinBUGS is a program for Bayesian model fitting by Gibbs sampling. WinBUGS has limited facilities for data handling, whereas Stata has no routines for Bayesian analysis; therefore, much can be gained by running Stata and WinBUGS together. We present a set of ado-files that enable data to be processed in Stata and then passed to WinBUGS for model fitting; finally, the results are read back into Stata for further processing.

Keywords: st0115, wbararray, wldata, wbscalar, wbstructure, wbvector, wbrun, wbscript, wbcoda, wbac, wbbgr, wbgeweke, wbintervals, wbsection, wbtrace, wbstats, wbdensity, wbdic, wbbull, wbdecode, Bayesian methods, MCMC, Gibbs sampling, WinBUGS

1 WinBUGS

WinBUGS is a program for Bayesian model fitting that runs under Windows and is available free for download from <http://www.mrc-bsu.cam.ac.uk/bugs/>. The program is fully described on that web site and in the manual that comes with the program. Its help facility also includes many examples consisting of datasets and associated WinBUGS programs. As a standalone program, WinBUGS is usually run interactively through a series of menus and toolbars. However, the current version, WinBUGS 1.4.1, includes a scripting language so that model fitting can be automated and controlled by a script file. With this scripting facility, one can run WinBUGS from within Stata. WinBUGS and Stata complement each other because WinBUGS has limited facilities for data handling, whereas Stata is excellent for this but has no routines for Bayesian model fitting. The main problem in combining the two programs is the different ways in which they store data. To help overcome this difficulty, we describe several ado-files for the transfer of data between the two programs, as well as some other ado-files for summarizing the results of the WinBUGS analysis.

1.1 Bayesian model fitting using Gibbs sampling

The scale of the literature on Bayesian analysis is such that giving a comprehensive review is impossible, but we will give a brief account and some references to more sources of information, which should let anyone follow the way that WinBUGS is run from within Stata. If you are new to WinBUGS, running a few interactive WinBUGS sessions to familiarize yourself with the WinBUGS language would be sensible before trying to combine WinBUGS and Stata.

In traditional statistics we often start by calculating the likelihood of a model, that is, the probability of the observed data given that model, $\Pr(\text{Data} \mid \text{Model})$. Two competing models are compared by the relative sizes of their likelihoods, with the model that was more likely to have produced the observed data being preferred. Bayesian model fitting takes this process one stage further by calculating $\Pr(\text{Model} \mid \text{Data})$; in this way candidate models can be judged by directly comparing their probabilities. To calculate $\Pr(\text{Model} \mid \text{Data})$ from the likelihood requires Bayes' theorem,

$$\Pr(\text{Model} \mid \text{Data}) = \frac{\Pr(\text{Data} \mid \text{Model}) \Pr(\text{Model})}{\Pr(\text{Data})}$$

The left-hand side of this expression represents the probability of the model given the data, and because it can be calculated only after we have seen the data, it is usually referred to as the *posterior probability* of the model. On the right-hand side, $\Pr(\text{Data} \mid \text{Model})$ is the likelihood and $\Pr(\text{Model})$ is the probability of the model without the current data. The easiest way to think of $\Pr(\text{Model})$ is as the probability of the model before we had access to the data; hence, it is called the *prior probability* of the model. Finally, $\Pr(\text{Data})$ is a term that ensures that the posterior probability is scaled between 0 and 1.

Priors

Although widely used, Bayesian methods are still not universally accepted, largely because of the philosophical and practical difficulties associated with the prior. Real data are not actually generated from models; rather, the models are inventions of the researcher that are designed to approximate the real world. Consequently, there is no such thing as the “correct” model. Given this view, models are not equivalent to events and so whether they can have probabilities is questionable. The usual way around this problem is to treat $\Pr(\text{Model})$ as a statement of personal belief, although many would argue that even this paradigm is not really satisfactory.

Even if we accept the existence of prior probabilities, there is still the tricky problem of specifying them. We have to give probabilities to every model that we think could possibly have generated the data. For complex problems, this is virtually impossible to do properly, and even when we can do it, we have to accept that two people might legitimately assign different priors and hence produce different analyses. Because there is no way of judging one prior to be better than another, researchers often try to remain neutral by assigning vague or noninformative priors that give similar prior probabilities to a wide range of models; the hope is that noninformative priors will have little influence on the final answer. With large datasets this approach works well, and a Bayesian analysis with noninformative priors is similar to an investigation of the likelihood. However, with sparse data, one must proceed with great caution because even seemingly noninformative priors can influence the final result.

Markov chain Monte Carlo

The scaling term, $\Pr(\text{Data})$, in Bayes' theorem presents an entirely different problem. Calculating the term involves summing over all candidate models, which typically requires either a very large summation or a multidimensional integral. For many years the difficulty of evaluating $\Pr(\text{Data})$ effectively restricted Bayesian analysis to simple problems in which the integration was tractable. Then, in the late 1980s, Bayesian statisticians started to investigate the use of Monte Carlo integration, that is, integration by simulation, and they imported some established methods from physics. Suppose that the possible models are labeled M_1, M_2, \dots . Then the problem of evaluating the posterior is solved if we can generate a random sequence or chain, such as $M_4, M_9, M_3, M_3, M_4, M_1, \dots$, such that, in the long run, each model occurs with a frequency proportional to $\Pr(\text{Model} | \text{Data})$. When we generate values of the chain so that the next model in the sequence depends only on its immediate predecessor, we have a Markov chain, so the method is referred to as Markov chain Monte Carlo (MCMC). Typically the models depend on an unknown vector of parameters, $\underline{\theta}$, so different models can be indexed by the value of that parameter and our chain can be written as, $\underline{\theta}_4, \underline{\theta}_9, \underline{\theta}_3, \underline{\theta}_3, \underline{\theta}_4, \underline{\theta}_1, \dots$.

Metropolis–Hastings

There are many methods for generating the next model or set of parameter values in an MCMC chain, but the most popular by far is the Metropolis–Hastings algorithm. The key to the algorithm is finding a good proposal distribution, $q(\underline{\theta}^* | \underline{\theta}_t)$, for suggesting a new value, $\underline{\theta}^*$, given the latest value in the chain $\underline{\theta}_t$. The choice of proposal distribution is essentially arbitrary, but some choices will be much more efficient than others in the sense of giving a chain that settles more quickly to the correct long-run probabilities. After generating a random value, $\underline{\theta}^*$, from our chosen proposal distribution, we calculate the ratio,

$$\alpha = \frac{\Pr(\underline{\theta}^* | \text{Data}) q(\underline{\theta}_t | \underline{\theta}^*)}{\Pr(\underline{\theta}_t | \text{Data}) q(\underline{\theta}^* | \underline{\theta}_t)}$$

A uniform value, u , between 0 and 1, is then generated, and if $u < \alpha$ we accept the proposed $\underline{\theta}^*$ as the next value in the chain; otherwise, the next value is a copy of $\underline{\theta}_t$. A poor choice of proposal distribution can get stuck on one set of parameter values, which will slow down the convergence of the chain.

Gibbs sampling

A special case of the Metropolis–Hastings algorithm is Gibbs sampling, which involves cycling through the parameters of the model one at a time rather than treating them as a vector and using the current estimate of the univariate conditional posterior probability distribution as the proposal distribution. Gibbs sampling can be inefficient, but by taking parameters one at a time, it reduces multidimensional problems to a series of univariate calculations and consequently is much easier to program. WinBUGS is a program for applying Gibbs sampling to a flexible class of user-specified models.

MCMC methods are not without their own problems, in particular:

- Successive values in a chain may be strongly dependent, so the values in the chain cannot be treated as a random sample from the posterior.
- The chain requires starting values, and this choice will influence the early part of the chain.
- Deciding how long the chain needs to be run before it adequately represents the posterior probabilities of the candidate models is difficult. The chains of some models converge quickly, whereas others can require hundreds of thousands of simulations.

[Gilks, Richardson, and Spiegelhalter \(1996\)](#) give a good account of the theory and practice of MCMC. [Gelman et al. \(1995\)](#) give a more general overview of practical Bayesian methods that includes a brief account of MCMC. [Lindley \(2000\)](#) discusses the philosophy underlying the Bayesian approach; several of the contributions to the discussion of that article, particularly those of John Nelder and George Barnard, demonstrate why some people still have doubts about the approach.

1.2 Specifying models in WinBUGS

WinBUGS has its own language for specifying models, described in detail in the WinBUGS manual, although many people find learning the language by following the accompanying WinBUGS examples to be easier. We will consider a straightforward model that will show how WinBUGS is run from within Stata. Suppose that we have a simple linear regression involving 5 observations of a response variable, y , and an explanatory variable, x . If the error distribution is assumed to be normal then the candidate models are

$$\begin{aligned}y_i &\sim N(\mu_i, \sigma) & i = 1, \dots, 5 \\ \mu_i &= \alpha + \beta x_i\end{aligned}$$

where the individual models are indexed by the values of the three parameters α , β , and σ .

For a Bayesian analysis we must state our prior belief in each of the possible values of the parameters. This step is complex, but for illustration we will assume that we are happy to put independent, relatively flat, prior distributions on the parameters:

$$\begin{aligned}\alpha &\sim N(0, 5) \\ \beta &\sim N(0, 5) \\ \sigma &\sim \text{Uniform}(0, 4)\end{aligned}$$

Thus we expect α and β to lie within the range of about ± 10 with all values around zero having more or less equal prior probability. σ is equally likely to be any value between 0 and 4, but values more than 4 are completely ruled out by this prior, whatever the data may say. We could describe this model in WinBUGS by using this more or less self-explanatory code:

```
model {
  for(i in 1:5) {
    mu[i] <- alpha + beta*x[i]
    y[i] ~ dnorm(mu[i], tau)
  }
  tau <- 1/(sigma*sigma)
  alpha ~ dnorm(0, 0.04)
  beta ~ dnorm(0, 0.04)
  sigma ~ dunif(0, 4)
}
```

The WinBUGS language is closely modeled on S-plus and R, in which the combined symbol `<-` is used to denote assignment, in preference to the more usual `=`. The symbol `~` denotes a distribution, which for the normal is parameterized by the mean and the precision, that is, one over the variance.

WinBUGS requires two other pieces of information before it can fit the model: the data and some starting values for the MCMC chain. Once again WinBUGS adopts the R style for data structures and so data are usually given as lists. For our problem the data might be supplied as

```
list(x=c(1,2,3,4,5), y=c(4,3.5,5,7.2,8.4))
```

and the initial values could be

```
list(alpha=0, beta=1, sigma=1)
```

The notation is again self-explanatory, with `c()` used to denote that values are combined into a vector.

2 Running WinBUGS from within Stata

We have developed several commands that run, summarize, and assess WinBUGS models to be fitted from within Stata. The commands are listed in table 1. Full details of the

commands can be obtained from a manual that can be downloaded as a PDF from our web site, <http://www.hs.le.ac.uk/research/HCG/winbugsfromstata/> (this URL is case sensitive), and from the help files that accompany the commands. The following sections in this article describe the basic commands and should be sufficient for running a simple analysis.

Table 1: WinBUGS commands

command	description
Writing data to WinBUGS	
<code>wbarray</code>	write data as a WinBUGS array
<code>wbdata</code>	write a mixed list of data
<code>wbscalar</code>	write a list of scalars
<code>wbstructure</code>	write a list containing a two-way structure
<code>wbvector</code>	write a list of vectors
Running WinBUGS	
<code>wbrun</code>	run WinBUGS from within Stata
<code>wbscript</code>	write a WinBUGS script file
Reading WinBUGS results	
<code>wbcoda</code>	read Coda-formatted results into Stata
Assessing convergence and mixing	
<code>wbac</code>	autocorrelation plots
<code>wbbgr</code>	Brooks–Gelman–Rubin convergence plot
<code>wbgeweke</code>	convergence test based on means
<code>wbintervals</code>	interval estimates for sections of a chain
<code>wbsection</code>	convergence check based on density estimates
<code>wbtrace</code>	trace plot of MCMC simulations
Summarizing MCMC simulations	
<code>wbstats</code>	summary statistics from MCMC simulations
<code>wbdensity</code>	posterior density estimates
<code>wbdic</code>	read DIC statistics from a WinBUGS log file into Stata
<code>wbhull</code>	bivariate posterior contour plots
Reading list data into Stata	
<code>wbdecode</code>	read list structures directly into Stata

WinBUGS uses compound files with the `.odc` extension; these can include descriptive text, the model, data, graphs, and results together in one place. So, for example, when reading the data, we merely highlight that part of the compound file. However, we can also use separate text files for the model, data, and initial values—much more convenient if we are to run WinBUGS from within Stata. Let us assume that the model, data, and initial values as given in the previous section are stored in text files called

`model.txt`, `data.txt`, and `inits.txt`, and that these files are stored in the folder `d:/example`. To run the model, all we need is another text file containing the script.

A basic script for our problem might be stored as `script.txt` and consist of the following:

```
display('log')
check('d:/example/model.txt')
data('d:/example/data.txt')
compile(1)
inits(1,'d:/example/inits.txt')
update(500)
set('alpha')
set('beta')
update(1000)
coda(*,'d:/example/out')
quit()
```

Line 1 opens a WinBUGS log window in which we can monitor the progress of our script and possibly see any error messages. Line 2 asks WinBUGS to check that our model description is syntactically correct; here we pick up the typing errors and missed brackets. Next we read the data and then on line 4 the model and data are compiled into a program for analyzing our problem. The “1” in the `compile()` statement tells WinBUGS that we intend to run only one chain. Running several parallel chains is a good way of discovering whether the chains have converged. Line 5 reads the initial values for chain 1 and then on line 6 we create an MCMC chain of length 500. This initial chain is called a *burn-in* and will be discarded by WinBUGS because we have not told it to store any results. The burn-in is intended to allow the chain to stabilize and thereby remove the effects of the initial values; deciding the appropriate length of the burn-in is an art in itself. The two `set` commands tell WinBUGS to store `alpha` and `beta` from any subsequent simulations. Then on line 9 we run another chain of length 1,000. The stored values of `alpha` and `beta` are written to an output file in Coda format. Coda is a program written in S-plus that is designed to examine MCMC output; in particular, Coda can be used to help assess whether the chain has converged. Finally, the `quit()` command closes WinBUGS. If `quit()` is omitted, WinBUGS stays active after the script is executed. This feature can be helpful when you need to debug a program, for instance, if there is an error in the model description.

To run `script.txt` when the WinBUGS executable is called `winbugs14.exe`, we could open a Command window and type

```
> winbugs14 /par script.txt
```

Running this command from within Stata requires either Stata’s `shell` command or the `winexec` command,

```
. shell winbugs14 /par script.txt
```

or

```
. winexec winbugs14 /par script.txt
```

The difference is that `shell` causes Stata to wait for WinBUGS to finish before it becomes active again, whereas with `winexec` the WinBUGS job is run in the background while Stata stays active. WinBUGS programs can sometimes take a long time to run and so it might be convenient to return control to Stata while we wait for WinBUGS. However, if the command were part of an ado-file, then one would probably not want to continue until WinBUGS had finished.

To automate the running of WinBUGS from within Stata, we have written a trivial ado-file called `wbrun.ado`.

Syntax

```
wbrun, script(string) winbugs(string) [batch]
```

<i>option</i>	description
<u>s</u> cript(<i>string</i>)	full path name of the script file
<u>w</u> inbugs(<i>string</i>)	full path name of the WinBUGS executable
<u>b</u> atch	run in background with control returned to Stata

Remarks

Since the location of WinBUGS is likely to change only when new versions of WinBUGS are released, editing the ado-file to drop the `winbugs` option and instead typing the full path of your executable as part of the program might be sensible. If required, there is another Stata command, `wbscript`, that will create the script file.

3 Transferring data from Stata to WinBUGS

Once the data have been transferred to WinBUGS, there will be no further opportunity to edit the data or select subsets. Indeed, WinBUGS will return an error even if you supply data on variables that are not used in the model. We mentioned earlier that WinBUGS uses R-style lists for structuring data. These are extremely flexible, but before giving other examples of their use we will consider the common situation of data in a rectangular array, as one might see in a spreadsheet or Stata's Data Editor.

3.1 Writing data arrays: `wbarray`

The data for the simple regression problem from section 2 could have been represented as a 5×2 data array, called a data frame in R. In WinBUGS, such a structure can be entered by creating a data file containing the following:

```
x[] y[]
1   4
2   3.5
3   5
4   7.2
5   8.4
END
```

`wbarray` writes data held in Stata as a WinBUGS array. The array is written to the Results window, where it can be checked or copied to the clipboard, and if required the array will also be written directly to a text file.

Syntax

```
wbarray varlist [if] [in] [, options]
```

<i>option</i>	description
<code>saving(string, replace)</code>	text file for the output
<code>format(string)</code>	formats for the output
<code><u>noprint</u></code>	do not display in the Results window

Example

If our regression data were stored in Stata as variables `x` and `y`, they could be exported using the command

```
. wbarray x y, saving(d:/example/data.txt, replace)
```

3.2 Writing lists of data: `wbvector`, `wbscalar`, `wbstructure`, and `wbdata`

Lists can contain scalars, vectors, or multiway structures similar to data arrays or matrices.

`wbvector` writes a list structure containing vectors.

Syntax

```
wbvector varlist [if] [in] [, options]
```

<i>option</i>	description
<u>s</u> aving(<i>string</i> , replace)	text file for the output
<u>f</u> ormat(<i>string</i>)	formats for the output
<u>l</u> inesize(<i>#</i>)	maximum number of values per line
<u>n</u> oprint	do not display in the Results window

Example

With the data from our regression example stored in Stata as variables `x` and `y`,

```
. wbvector x y if x < 4 , format(%3.1f)
```

produces

```
list(x=c(1.0,2.0,3.0), y=c(4.0,3.5,5.0))
```

Other commands for writing lists are `wbscalar`, for lists of scalars; `wbstructure`, for two-way data structures; and `wbdata`, for a mixed list of scalars, vectors, and/or two-way structures. WinBUGS can read more than one data file for the same model, so one can mix, say, arrays and scalars by placing them in different files.

4 Transferring data from WinBUGS to Stata

WinBUGS writes the saved MCMC simulations in the format required by the program Coda. Doing so results in an index file listing the variable names and the numbering of the simulations, and data files, one for each chain, that contain the actual values. Thus, when our script generated one chain and then included the line

```
coda(*,'d:/example/out')
```

this created an index file, `d:/example/outIndex.txt`, and a data file, `d:/example/out1.txt`. The index file for our regression problem would contain

```
alpha 1    1000
beta  1001 2000
```

meaning that the chain for `alpha` is found in lines 1–1,000 of the data file and the values for `beta` are in lines 1,001–2,000. The data file itself might start

```
501 0.2346
502 1.3259
... ..
```

The first column refers to the order in the chain and here tells us that we discarded 500 values before starting to save `alpha` and `beta`. The second column contains the simulated parameter values, `alpha` first and then `beta`. Had we run two parallel chains, there would have been another data file, `d:/example/out2.txt`, with the same structure.

The command `wbcoda` will read Coda-formatted files directly into Stata, reshaping them so that each variable is in a separate column. `wbcoda` can also read in multiple chains, in which case the values of a parameter are placed in the same column with values of the second chain stacked below those of the first chain. The chains are distinguished by an indicator variable that takes the value of the chain number, 1, 2, 3, ...

Syntax

```
wbcoda, root(string) clear [options]
```

<i>option</i>	description
<u>r</u> oot(<i>string</i>)	root (with full path) for the index and data file(s)
clear	permission to clear current data from Stata
<u>m</u> ultichain	read multiple chains
<u>c</u> hains(<i>#</i>)	number of chains (multichain mode) or number of the chain to be read
<u>i</u> d(<i>string</i>)	name for the chain identifier
<u>t</u> hin(<i>#</i>)	retain every <i>n</i> th value
<u>k</u> eep(<i>string</i>)	list of parameters to retain
<u>n</u> oreshape	leave the values stacked in one variable

Examples

To read one chain from `d:/example/out1.txt`,

```
. wbcoda, root(d:/example/out) clear
```

and to read three chains from `d:/example/out1.txt`, `d:/example/out2.txt`, and `d:/example/out3.txt`,

```
. wbcoda, root(d:/example/out) clear chain(3) multichain
```

whereas to read just the second chain,

```
. wbcoda, root(d:/example/out) clear chain(2)
```

5 Ado-files for examining MCMC simulations

5.1 Assessing convergence

When the MCMC chain or chains have been created, we must first satisfy ourselves that they have converged; this means checking that the burn-in was long enough so that the early part of the chain is not heavily dependent on the starting values and then checking that the chain has been run long enough to have stabilized the probabilities of the different models. Many methods have been suggested for checking convergence, but none is totally satisfactory; a chain may appear to be stable for a long time and then suddenly discover a new set of plausible parameter values that had not previously been proposed. Most convergence checking is graphical and either compares the results from different chains or divides one chain into sections and compares the sections. If the simulation has not yet converged, then the chains or part-chains will look different when plotted.

wbtrace

wbtrace produces a trace or time-series plot of the values in the chain or of values derived from them. It will show if the chain is drifting, perhaps indicating that the burn-in was not long enough, and it will illustrate the speed of mixing, which is how quickly the chain moves across the distribution. Chains that mix slowly will produce long, slow cycles, and they take longer to converge. Mixing can sometimes be improved by reparameterizing the model.

Syntax

```
wbtrace varlist [if] [in] [, options]
```

<i>option</i>	description
<u>t</u> hin(<i>#</i>)	plot only every <i>n</i> th value
<u>i</u> d(<i>varname</i>)	chain indicator for multiple chains
<u>o</u> rd(er)(<i>varname</i>)	variable to be used for the <i>x</i> axis
<u>b</u> ychain(<i>varname</i>)	chain identifier for parallel chains
<u>o</u> verlay(<i>varname</i>)	overlay parallel chains on the same graph
<u>g</u> options(<i>string</i>)	options passed directly to the plotting commands
<u>c</u> goptions(<i>string</i>)	options passed directly to the <code>graph combine</code> command
<u>s</u> aving(<i>string</i> , <i>replace</i>)	graphics file for saving the plot
<u>e</u> xport(<i>string</i> , <i>replace</i>)	file for exporting the plot as <code>.eps</code> , <code>.png</code> , etc.

Remarks

If *varlist* contains more than one parameter, the traces are combined into one plot, as in figure 1. With multiple chains the traces for each parameter can be superimposed.

wbsection

wbsection divides one chain into subsections of equal length and then uses **kdensity** to produce a smooth density estimate for each section. The densities are superimposed in the plot and shown together with the smooth density for the whole chain. If the chain has converged, the smoothed densities will be similar to one another. A simple measure of convergence, *D*, is returned by the program; it represents the maximum distance between two densities as a percentage of the maximum height of the combined density.

Syntax

wbsection *varlist* [, *options*]

<i>option</i>	description
m (#)	number of subsections into which the chain is divided
bychain (<i>varname</i>)	chain identifier comparison of chains rather than sections
koptions (<i>string</i>)	options passed directly to kdensity
goptions (<i>string</i>)	options passed directly to the plotting command
cgoptions (<i>string</i>)	options passed directly to the graph combine commands
saving (<i>string</i> , replace)	graphics file for saving the plot
export (<i>string</i> , replace)	file for exporting the plot as .eps , .png , etc.
dsaving (<i>string</i> , replace)	file for saving the plotting points

wbac, wbbgr, wbintervals, and wbgeweke

Other commands for assessing convergence and mixing are **wbac**, which is a wrapper for Stata's **ac** and **pac** commands for plotting the autocorrelations; **wbbgr**, which produces the Brooks–Gelman–Rubin plot for assessing convergence from parallel chains; **wbintervals**, which plots interval estimates based on sections of a chain; and **wbgeweke**, which performs a test of the mean of the early part of a chain compared to the mean of the late part of that chain.

5.2 Summarizing MCMC results

wbstats

The standard summary, `wbstats`, for an MCMC chain contains the following for each parameter:

- the mean, which is often taken as the point estimate of the parameter;
- the standard deviation, which describes the spread of the distribution;
- a standard error, which is calculated allowing for autocorrelation giving an indication of whether the chain was long enough for the accuracy we require;
- the median, which is an alternative point estimate for nonsymmetric distributions; and
- a 95% credible interval, which is a central 95% interval calculated from the posterior, similar in spirit to a 95% confidence interval.

Syntax

```
wbstats varlist [if] [in] [, options]
```

<i>option</i>	description
<code>hpd</code>	highest posterior density intervals instead of credible intervals
<code>level(#)</code>	percent level for the intervals; default is <code>level(95)</code>

wbdensity and wbhull

Other commands for summarizing a chain are `wbdensity`, which plots a density estimate of the posterior distribution, and `wbhull`, which shows the contours of the bivariate distribution of pairs of parameters on the basis of their convex hulls.

6 Example: Random-effects logistic regression

This example shows how the various `wb` commands can be combined to create a complete analysis. We have chosen an example taken from the WinBUGS help files that is simple enough to be analyzed in Stata by using `gllamm`. The WinBUGS help file stores this example under the name `seeds` and includes the raw data and the code for the model.

The example is taken from [Crowder \(1978\)](#) and concerns the proportion of seeds that germinated on each of 21 plates arranged according to an unbalanced 2×2 factorial

design of seed variety and type of root extract. The analysis in the WinBUGS help system uses a logistic model with a random effect to allow for overdispersion. For the i th plate, x_{1i} denotes the seed type; x_{2i} denotes the root extract; n_i is the total number of seeds; r_i is the number of seeds that germinated; and p_i is the probability of germination. The model, including a seed by root interaction, can be written algebraically as

$$\begin{aligned} r_i &\sim \text{Bin}(n_i, p_i) \\ \text{logit}(p_i) &= \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_{12} x_{1i} x_{2i} + b_i \\ b_i &\sim N(0, \sigma) \end{aligned} \tag{1}$$

Generating a variable `x1_x2` equal to the product of `x1` and `x2` and then fitting this model with `gllamm` produces

```
. gllamm r x1 x2 x1_x2, i(plate) link(logit) family(binomial) denom(n) adapt
```

r	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]
x1	.0969755	.2780556	0.35	0.727	-.4480035 .6419545
x2	1.337056	.2369515	5.64	0.000	.8726397 1.801473
x1_x2	-.8104534	.3851909	-2.10	0.035	-1.565414 -.0554932
_cons	-.5484376	.1666035	-3.29	0.001	-.8749744 -.2219007

Variances and covariances of random effects

 ***level 2 (plate)

var(1): .05582647 (.05200718)

The estimate of σ is $\sqrt{0.055826} = 0.236$, with a standard error calculated using the delta method of $0.052007/(2\sqrt{0.055826}) = 0.110$.

An alternative Stata analysis could be produced by expanding the data to one line per seed and using the `xtlogit` command. This method is a little quicker than `gllamm` and gives essentially the same answers.

For the Bayesian analysis, the coefficients α_0 , α_1 , α_2 , and α_{12} are given independent noninformative normal priors, whereas the standard deviation of the random effect, σ , has a vague uniform distribution. The complete model becomes

```

model {
  for(i in 1:N) {
    r[i] ~ dbin(p[i], n[i])
    b[i] ~ dnorm(0.0, prec)
    logit(p[i]) <- alpha0 + alpha1*x1[i] + alpha2*x2[i] + ///
      alpha12*x1[i]*x2[i] + b[i]
  }
  prec <- 1/(sigma*sigma)
  alpha0 ~ dnorm(0.0, 1.0E-6)
  alpha1 ~ dnorm(0.0, 1.0E-6)
  alpha2 ~ dnorm(0.0, 1.0E-6)
  alpha12 ~ dnorm(0.0, 1.0E-6)
  sigma ~ dunif(0, 100)
}

```

The initial values given by the WinBUGS manual are

```
list(alpha0 = 0, alpha1 = 0, alpha2 = 0, alpha12 = 0, sigma = 1)
```

and our script file has the same structure as the example in section 2, except that the burn-in has length 1,000 and the actual run is of length 10,000. All the parameters are saved.

Running the analysis in Stata using `wbrun` took 14 seconds compared with 7 seconds for the `gllamm` analysis. This timing is a little artificial since the Bayesian analysis would have converged with a shorter chain, but we chose to use the length suggested in the WinBUGS manual.

Figure 1 was produced by `wbtrace` and shows the traces for the parameters α_1 and α_2 . There is no evidence of drift and the mixing is good, with the full range of each parameter covered in relatively few simulations. The plot also shows the median and 95% credible interval for each parameter.

(Continued on next page)

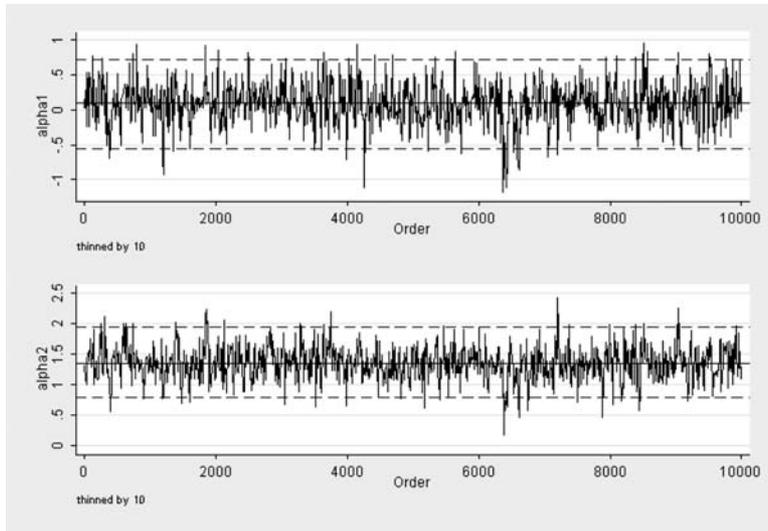


Figure 1: Trace of the chains for α_1 and α_2

Figure 2 was produced by `wbsection` and shows the densities of sections of the chains for the parameters α_1 , α_2 , α_{12} , and σ . They are in good agreement, reinforcing the idea that the chains have converged.

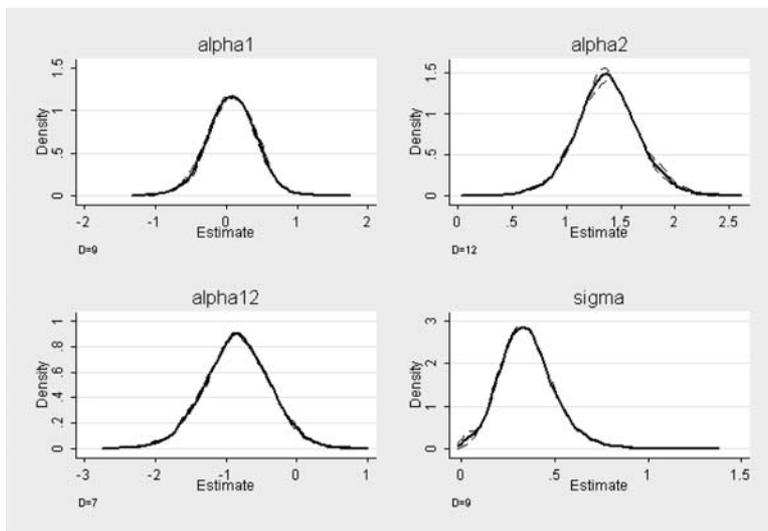


Figure 2: Density estimates for sections of the chains for α_1 , α_2 , α_{12} , and σ . Dashed lines, densities for first and second halves of each chain; solid line, density based on full chain.

`wbstats` summarizes the results. They are similar to those for `gllamm`, although the estimate for σ is larger.

```
. wbstats alpha1 alpha2 alpha12 alpha0 sigma
Parameter      n      mean      sd      se  median      95% CrI
alpha1         10000    0.075    0.345    0.0129  0.078 ( -0.627,  0.735 )
alpha2         10000    1.372    0.290    0.0114  1.368 (  0.791,  1.953 )
alpha12        10000   -0.853    0.471    0.0178 -0.847 ( -1.801,  0.056 )
alpha0         10000   -0.558    0.210    0.0082 -0.561 ( -0.976, -0.142 )
sigma          10000    0.353    0.148    0.0050  0.343 (  0.084,  0.677 )
```

Some editions of the WinBUGS manual contrast this analysis with another based on the same model but with a noninformative gamma prior for the precision of the random effect. This seemingly innocent change alters the estimate of σ to make it much closer to that given by `gllamm`. With this model `wbstats` gives

```
. wbstats alpha1 alpha2 alpha12 alpha0 sigma
Parameter      n      mean      sd      se  median      95% CrI
alpha1         10000    0.081    0.309    0.0102  0.091 ( -0.564,  0.650 )
alpha2         10000    1.361    0.277    0.0104  1.364 (  0.792,  1.914 )
alpha12        10000   -0.829    0.424    0.0146 -0.829 ( -1.669,  0.038 )
alpha0         10000   -0.553    0.194    0.0070 -0.556 ( -0.936, -0.149 )
sigma          10000    0.287    0.140    0.0051  0.278 (  0.046,  0.591 )
```

The density estimates of the posterior distributions for the two priors, as produced by `wbdensity`, are shown in figure 3. Of course, being close to the maximum likelihood solution is not a justification for a particular prior, and both are correct conditional on properly capturing our prior beliefs about the random effect. [Gelman \(2005\)](#) has recently criticized gamma precision priors of the form $G(\epsilon, \epsilon)$ because of their sensitivity to the choice of ϵ when the random effect is small; instead, he recommends a uniform prior on σ as used in our first analysis. If nothing else, the difference illustrates the importance of choosing priors with care, even when they are supposed to be noninformative.

(Continued on next page)

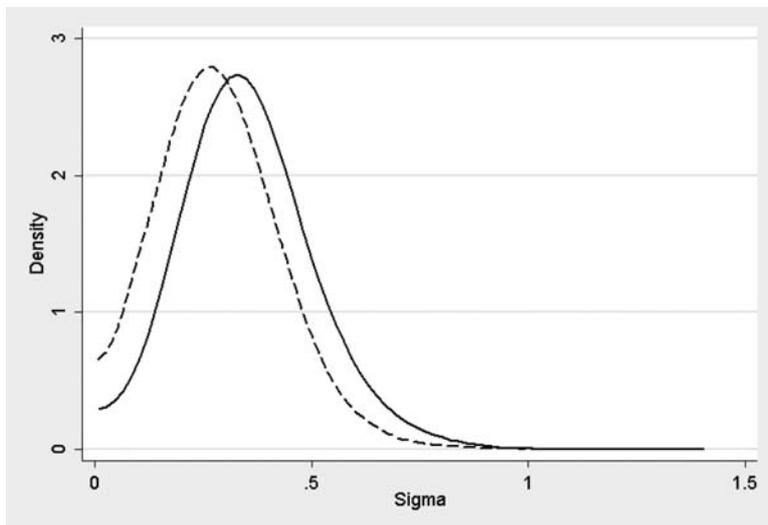


Figure 3: Density estimates for the posterior of σ , calculated using a vague uniform prior for σ (solid line) or vague gamma prior for the precision (dashed line).

7 Final comments

A glance at the examples in the WinBUGS manual will show the great variety of models that can be fitted using MCMC, some of which cannot be fitted by any existing Stata command. The WinBUGS web site also gives details of *GeoBUGS*, a collection of programs for spatial statistics, and *PKBUGS*, programs for pharmacokinetic models. WinBUGS's great flexibility for fitting your own models is what makes it such an attractive package; complex random effects and nonlinear relationships present no special problems. Model construction is so easy in WinBUGS that one of the main failings of recent Bayesian analysis has been a tendency to make models so complex that prior specification, model checking, and model selection become difficult.

If you want to experiment with the examples in the WinBUGS manual, you might find the command `wbdecode` useful. The data for the examples are given as R-style lists, and `wbdecode` is a program for reading such lists directly into Stata.

We have found the `wb` commands particularly useful when running simulations. Stata can be used to simulate a random dataset that can be passed to WinBUGS for model fitting, with the results summarized and stored by Stata. In this way, one can run simulations that are simply impractical when WinBUGS is used interactively. The other big advantages are the ease with which models can be fitted with different patterns of covariates and Stata's better-quality graphical output. Of course, acknowledging that these same advantages are available when using WinBUGS with R is only fair.

All the programs referred to in this article are available from our web site at <http://www.hs.le.ac.uk/research/HCG/winbugsfromstata/> (case-sensitive URL). At the same site, you will find a manual that gives full details of each command listed in table 1 and two further analyses of data taken from the WinBUGS examples. In each case, the analyses of the examples include the Stata code needed to complete the Bayesian model fitting, to check for convergence, and to summarize the results.

8 References

- Crowder, M. J. 1978. Beta-binomial ANOVA for proportions. *Applied Statistics* 27: 34–37.
- Gelman, A. 2005. Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis* 1: 515–534.
- Gelman, A., J. B. Carlin, H. S. Stern, and D. B. Rubin, ed. 1995. *Bayesian Data Analysis*. London: Chapman & Hall.
- Gilks, W. R., S. Richardson, and D. J. Spiegelhalter, ed. 1996. *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.
- Lindley, D. V. 2000. The philosophy of statistics. *Statistician* 49: 293–337.

About the authors

John Thompson is a professor in the Department of Health Sciences at the University of Leicester with a research interest in genetic epidemiology. He teaches in the department's MSc program in medical statistics and is a longtime Stata user and enthusiast.

Tom Palmer is studying for a PhD in genetic epidemiology.

Santiago Moreno is studying for a PhD in medical statistics.