The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–18

Bayesian Analysis in Stata using WinBUGS

John Thompson
Department of Health Sciences
University of Leicester
Leicester, UK
john.thompson@le.ac.uk

Tom Palmer
Department of Health Sciences
University of Leicester
Leicester, UK

Santiago Moreno
Department of Health Sciences
University of Leicester
Leicester, UK

Abstract. WinBUGS is a program for Bayesian model fitting by Gibbs sampling. WinBUGS has very limited facilities for data handling while Stata has no routines for Bayesian analysis, and as a result there is a lot to be gained by running Stata and WinBUGS in combination. A set of ado files are presented that enable data to be processed in Stata, passed to WinBUGS for model fitting and the results read back into Stata for further processing.

Keywords: st0001, 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 `www.mrc-bsu.cam.ac.uk/bugs`. The program is fully described on that website and in the manual that comes with the program and its help facility also includes a large number of examples consisting of data sets and associated `WinBUGS` programs. As a stand alone program `WinBUGS` is usually run interactively through a series of menus and toolbars. However the current version, `WinBUGS1.4.1`, includes a scripting language so that model fitting can be automated and controlled by a script file. By making use of this scripting facility it is possible to run `WinBUGS` from within `Stata`. `WinBUGS` and `Stata` complement each other very well because `WinBUGS` has very limited facilities for data handling, while `Stata` is excellent for this but has no routines for Bayesian model fitting. The main problem in combining the two programs is the very different ways in which they store data. To help overcome this difficulty we describe a number of ado files for the transfer of data between the two programs and some other ados 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 it is impossible to give a comprehensive review, but we will give a brief account and some references to further sources of information and this should be sufficient to enable anyone to follow the way that `WinBUGS` is run from within `Stata`. If you are new to `WinBUGS` it would be sensible to run a few interactive `WinBUGS` sessions in order to familiarize yourself with the `WinBUGS` language 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, P(Data|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 P(Model|Data), in this way candidate models can be judged by directly comparing their probabilities. To calculate P(Model|Data) from the likelihood requires Bayes theorem,

$$P(Model|Data) = \frac{P(Data|Model)P(Model)}{P(Data)} \tag{1}$$

The left hand side of this expression represents the probability of the model given the data and as it can only be calculated after we have seen the data, it is usually referred to as the *posterior probability* of the model. On the right hand side, P(Data|Model) is the likelihood and P(Model) is the probability of the model without the current data. The easiest way to think of this 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 P(Data) is a term that ensures that the posterior probability is scaled between 0 and 1.

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 but 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 it is questionable whether they can have probabilities. The usual way around this problem is to treat P(Model) as a statement of personal belief, although many would argue that even this is not really satifactory.

Even if we accept the existence of prior probabilities, there is still the very tricky problem of specifying them. We have to give probabilities to *every* model that we consider 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, quite legitimately, assign different priors and hence produce different analyses. As there is no way of judging one prior to be better than another, researchers often try to remain neutral by assigning *vague* or *non-informative* priors that give similar prior probabilities to a wide range of models; the hope is that non-informative priors will have little influence on the final answer. With large data sets this works well and a Bayesian analysis with non-informative priors is very similar to an investigation of the likelihood. However with sparse data, it is necessary to proceed with great caution because even seemly non-informative priors can influence the final result.

The scaling term, P(Data), in Bayes theorem presents an entirely different problem in that it is very difficult to calculate because it involves summing over all candidate models; typically this requires either a very large summation or a multidimensional integral. For many years the difficulty of evaluating P(Data) effectively restricted Bayesian analysis to simple problems in which the integration was tractable. Then, in the late 1980's, 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 labelled M_1, M_2, ... 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, ...such that, in the long run, each model occurs with a frequency proportion to P(Model|Data). When we generate values of the chain so that the next model in the sequence only depends on its immediate predecessor, we have a Markov chain, so the method is referred to as Markov chain Monte Carlo or 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, $\underline{\theta}_4$, $\underline{\theta}_9$, $\underline{\theta}_3$, $\underline{\theta}_3$, $\underline{\theta}_4$, $\underline{\theta}_1$, ...

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 to find 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 prosposal 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,

$$\alpha = \frac{P(\underline{\theta}^*|Data)\ q(\underline{\theta}_t|\underline{\theta}^*)}{P(\underline{\theta}_t|Data)\ q(\underline{\theta}^*|\underline{\theta}_t)} \tag{2}$$

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 disribution can get stuck on one set of parameter values, which will slow down the convergence of the chain.

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 very inefficient, but by taking parameters one at a time it reduces multidimensional problems to a series of univariate calculations and consequently it is much easier to program. `WinBUGS` is a program for applying Gibbs sampling to a very 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
- It is difficult to decide how long the chain needs to be run for before it adequately represents the posterior probabilities of the candidate models. The chains of some models converge very quickly, while others can require 100,000's of simulations.

Gilks, Richardson, and Spiegelhalter [1996] give a good account of the theory and practice of MCMC. Gelman, Carlin, Stern, and Rubin [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 and several of the contributions to the discussion of that article, particlarly 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 and this is described in detail in the `WinBUGS` manual, although many people find it easier to learn the language by following the accompanying `WinBUGS` examples. We will consider a very straightforward model that will be sufficient to illustrate 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,

$$\begin{aligned} y_i &\sim N(\mu_i, \sigma) \qquad i = 1, \ldots, 5 \\ \mu_i &= \alpha + \beta x_i \end{aligned} \tag{3}$$

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 is complex, but for illustration we will assume that we are happy to put independent, relatively flat, prior distributions on the parameters

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

Thus we expect alpha and beta to lie within the range of about ± 10 with all values around zero having more or less equal prior probability. Sigma is equally likely to be any value between 0 and 4 but values over 4 are completely ruled out by this prior, whatever the data may say. We could describe this model in `WinBUGS` using the 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 modelled 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 in the case of the normal is parameterised by the mean and the precision, which 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

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, it is equally possible to use separate text files for the model, data and initial values and if we are to run WinBUGS from within Stata, this is much more convenient. 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,

```
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 will be able to follow the progress of our script and possibly see any error messages. Line 2 asks WinBUGS to check that our model description is syntactially correct, it is at this stage that 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 analysing our problem. The one in the compile statement tells `WinBUGS` that we only intend running 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 stabilise and so remove the effects of the initial values; deciding on 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 a further chain of length 1000. 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 for examining MCMC output, in particular `Coda` can be used to help assess whether the chain has converged. Finally the quit() command closes `WinBUGS`. If the quit() is omitted then `WinBUGS` stays active after the script is executed. This 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
```

To run 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, while 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.

2.1 wbrun

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 ]`

option	*description*
script(*string*)	full path name of the script file
winbugs(*string*)	full path name of the WinBUGS executable
batch	run in background with control returned to Stata

Remarks

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

3 Transferring data from Stata to WinBUGS

It is important to remember that once the data have been transferred to WinBUGS there will be no further opportunity to edit the data or select subsets, indeed WinBUGS will even return an error if you supply data on variables that are not used in the model. We mentioned in the previous section 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 very 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 by 2 data array, called a dataframe in R. In WinBUGS such a structure can be entered by creating a data file containing,

```
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 in can be checked or copied to the clipboard, and if required the array will also we written directly to a text file.

Syntax

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

option	*description*
saving(*string*,replace)	text file for the output
format(*string*)	formats for the output
noprint	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

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

3.3 wbvector

Writes a list structure containing vectors.

Syntax

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

option	*description*
saving(*string*,replace)	text file for the output
format(*string*)	formats for the output
linesize(#)	maximum number of values per line
noprint	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))
```

3.4 wbscalar, wbstructure and wbdata

Other commands for writing lists are; `wbscalar`, for lists of scalars, `wbstructure` for twoway data structures, and `wbdata` for a mixed list of scalars, vectors and/or twoway structures. `WinBUGS` can read more than one data file for the same model so it is possible to 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`. This 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 to 1000 of the data file and the values for beta are in lines 1001 to 2000. 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, then beta. Had we run two parallel chains there would have been another data file `d:/example/out2.txt` with the same structure.

4.1 wbcoda

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*]

option	*description*
root(*string*)	root (with full path) for the index and data file(s)
clear	permission to clear current data from Stata
multichain	read multiple chains
chains(#)	number of chains (multichain mode) or number of the chain to be read
id(*string*)	name for the chain identifier
thin(#)	retain every nth value
keep(*string*)	list of parameters to retain
noreshape	leave the values stacked in a single variable

Examples

To read a single chain from `d:/example/out1.txt`

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

and to read 3 chains from `d:/example/out1.txt`, `d:/example/out2.txt` and `d:/example/out3.txt`

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

while to read just the second chain

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

5 Ado files for examining the MCMC simulations

5.1 Assessing convergence

When the MCMC chain or chains have been created we must first satisfy ourselves that the 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 for long enough for the probabilities of the different models to have stabilised. Many methods have been suggested for checking convergence but no one 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.

5.2 wbtrace

This command 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, that 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 reparameterising the model.

Syntax

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

option	*description*
thin(#)	only plot every nth value
id(*varname*)	chain indicator for multiple chains
order(*varname*)	variable to be used for the x-axis
bychain(*varname*)	chain identifier for parallel chains
overlay(*varname*)	overlay parallel chains on the same graph
goptions(*string*)	options passed directly to the plotting commands
cgoptions(*string*)	options passed directly to the graph combine command
saving(*string*,replace)	graphics file for saving the plot
export(*string*,replace)	file for exporting the plot as eps, png etc.

Remarks

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

5.3 wbsection

This command divides a single chain into subsections of equal length and then uses `kdensity` to produce a smooth density estimates 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 very 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*]

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

5.4 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.

5.5 Summarising the MCMC results

5.6 wbstats

The standard summary for an MCMC chain contains for each parameter; the mean, which is often taken as the point estimate of the parameter, the standard deviation to describe the spread of the distribution, a standard error calculated allowing for autocorrelation, which gives an indication of whether the chain was long enough for the accuracy we require, the median, an alternative point estimate for non-symmetric 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*]

option	*description*
hpd	highest posterior density intervals instead of credible intervals
level(#)	percentage level for the intervals. Default 95.

5.7 wbdensity and wbhull

Other commands for summarising 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 based on 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 analysed in `Stata` 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 by 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 over-dispersion. 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,

$$\begin{aligned} r_i &\sim Bin(n_i, p_i) \\ logit(p_i) &= \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_{12} x_{1i} x_{2i} + b_i \\ b_i &\sim N(0, \sigma) \end{aligned} \tag{5}$$

Generating a variable $x_1\_x_2$ equal to the product of x_1 and x_2 and then fitting this model using `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 is a little quicker than `gllamm` and gives essentially the same answers.

For the Bayesian analysis the coefficients $\alpha_0, \alpha_1, \alpha_2, \alpha_{12}$ are given independent non-informative normal priors while 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 1000 and the actual run is of length 10000. All of the parameters are saved.

Running the analysis in `Stata` using `wbrun` took 14 seconds compared to 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 quite 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.

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. The dashed lines show the densities for the first and second halves of each chain and the solid line shows the density based on

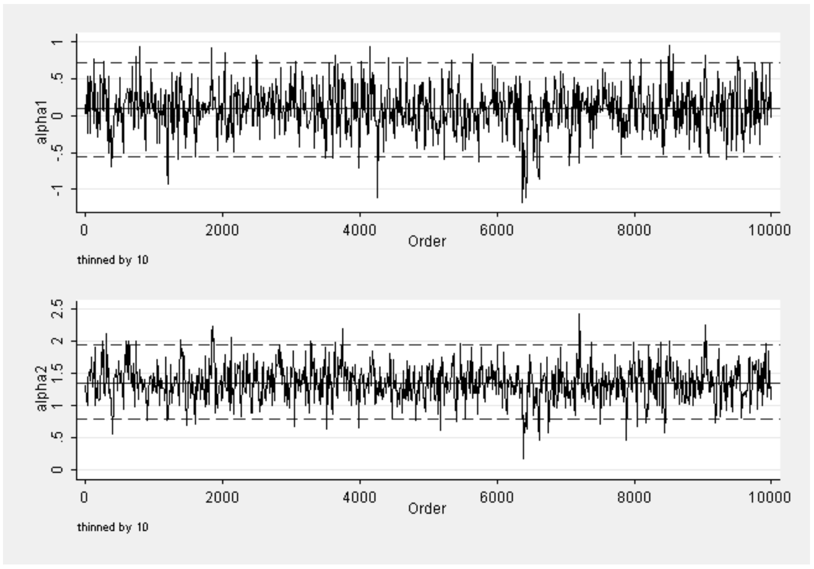

Figure 1: Trace of the chains for α_1 and α_2

the full chain. They are in very good agreement re-enforcing the idea that the chains have converged

`wbstats` summarises 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 non-informative gamma prior for the precision of the random effect. This seemingly innocent change alters the estimate of sigma to make it much closer to that given by `gllamm`. With this model `wbstats` gives,

```
. wbstats alpha0 alpha1 alpha2 alpha12 sigma
Parameter        n    mean      sd      se  median         95% CrI
alpha0       10000  -0.553   0.194  0.0070  -0.556 (  -0.936,  -0.149 )
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 )
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

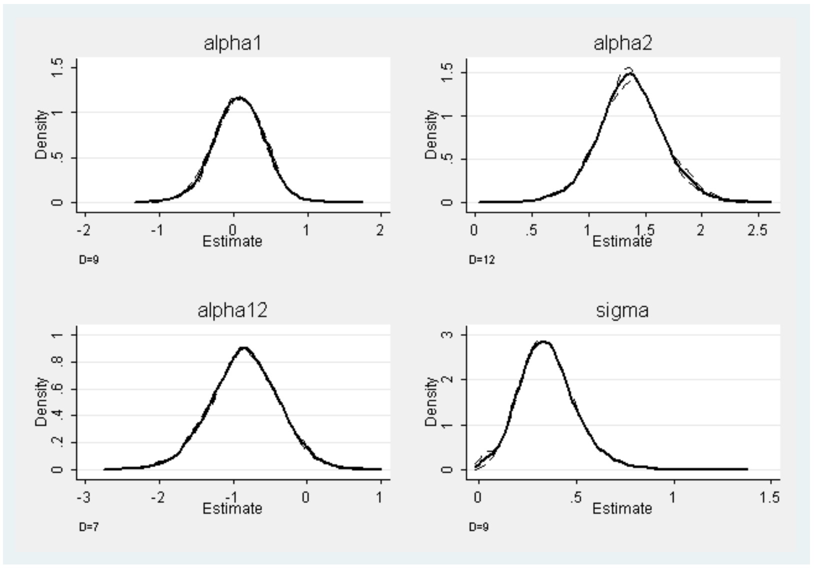

Figure 2: Density estimates for sections of the chains for α_1, α_2, α_{12} and σ

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 criticised gamma precision priors of the form G(ϵ,ϵ) 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 chosing priors with care.

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 current `Stata` command. The `WinBUGS` website also gives details of `GeoBUGS` a collection of programs for spatial statistics and `PKBUGS`, programs for pharmacokinetic models. It is the great flexibility offered by `WinBUGS` for fitting your own models that makes it such an attractive package; complex random effects and non-linear 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.

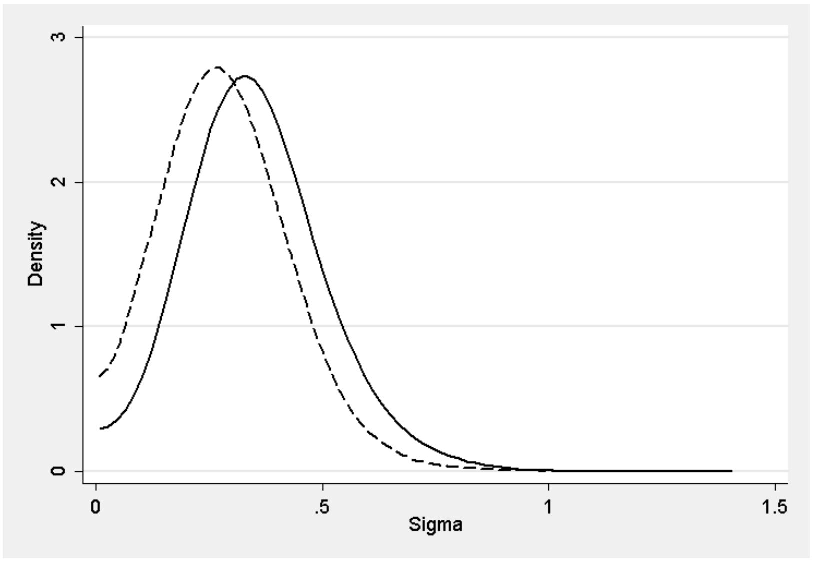

Figure 3: Density estimates for the posterior of σ using a vague uniform prior for σ (solid line) or vague gamma prior for the precision (dashed line).

We have found the `wb` commands particularly useful when running simulations. `Stata` can be used to simulate a random data set which can be passed to `WinBUGS` for model fitting with the results summarised and stored by `Stata`. In this way it is possible to 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 the better quality graphical output provided by Stata. Of course, it is only fair to acknowledge that these same advantages are available when using `WinBUGS` in conjunction with `R`.

All of the programs referred to in this article are available from our website at `www.hs.le.ac.uk/research/HCG/winbugsfromstata/`.

7.1 Author information

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 on the department's MSc 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.

8 References

Crowder, M. J. 1978. Beta-binomial Anova for Proportions. *Applied Statistics* 27(1): 34–37.

Gelman, A. 2005. Prior distributions for variance parameters in hierarchical models. *Bayesian analysis* 1: 1–19.

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 philossophy of statistics. *The Statistician* 49: 293–337.