Using the WinBUGS interface in MLwiN

Dr. William Browne.
Centre for Multilevel Modelling,
Institute of Education,
London.

22nd October 2001.

Background to MLwiN

MLwiN is a statistical software package designed to fit multilevel models and developed at the Institute of Education. It is the latest in a line of multilevel modelling packages developed over the last 15 years and forerunners include ML2, ML3 and MLN. It is produced by a team of academics at the Institute under the leadership of Professor Harvey Goldstein. It is sold for a small fee and currently has a user community of around 3,000 users worldwide, primarily academic researchers in the social and medical sciences.

The project team consists of 3 research officers: Jon Rasbash who is the package's main programmer, Min Yang who is in charge of user support and William Browne who is the programmer of the MCMC options in the package. There are also two lecturers at the Institute, Fiona Steele and Ian Plewis who are associated with the project and a group of project fellows from other institutions around the UK.

MLwiN has different aims and attracts a different user base from WinBUGS. It consists of a user-friendly Windows interface (written in Visual Basic) on top of fast estimation engines (written in C++). It is designed to fit particular classes of statistical model and so is less ambitious than WinBUGS, which attempts to fit almost all statistical models. However more attention is given to creating fast algorithms for the particular classes of models that MLwiN can fit and more documentation is provided.

MLwiN allows the user to perform both frequentist and Bayesian analysis of multilevel and other models. The frequentist analyses are performed using least squares based methods such as IGLS (Goldstein 1986) and RIGLS whilst the Bayesian analyses use both Gibbs sampling and Metropolis Hastings sampling depending on the problem. Users specify their models via an equation based interface and running the analysis involves simply clicking on a start button.

MLwiN to WinBUGS interface

A new 'development' release of MLwiN is due out early next year and contains many new MCMC features. In particular the new version will contain the ability to convert any model set up in MLwiN into WinBUGS code that can be run using WinBUGS version 1.3. Models available include:

- Normal response linear and multilevel models.
- Multilevel logistic and probit regression models.
- Multilevel Poisson regression models.
- Measurement error models.
- Cross-classified and multiple membership models.
- Spatial CAR models.
- Multivariate normal response models.
- Multilevel factor analysis models.

The new release of MLwiN will coincide with the release of a detailed manual '*MCMC Estimation in MLwiN*' by W.J. Browne (2002) and the remainder of this poster is an adaptation of a chapter on the WinBUGS interface taken from this book.

The MCMC features in MLwiN are fairly new and we currently fit only models of particular types although we are constantly extending the number of models that can be fitted. If, however, a user wishes to fit a model that cannot be currently fitted, for example fitting an alternative distribution for the school level random effects, there are three main options. Firstly wait for a later version of MLwiN that will fit their model; secondly write their own code to fit their model; or thirdly try an alternative software package, for example WinBUGS.

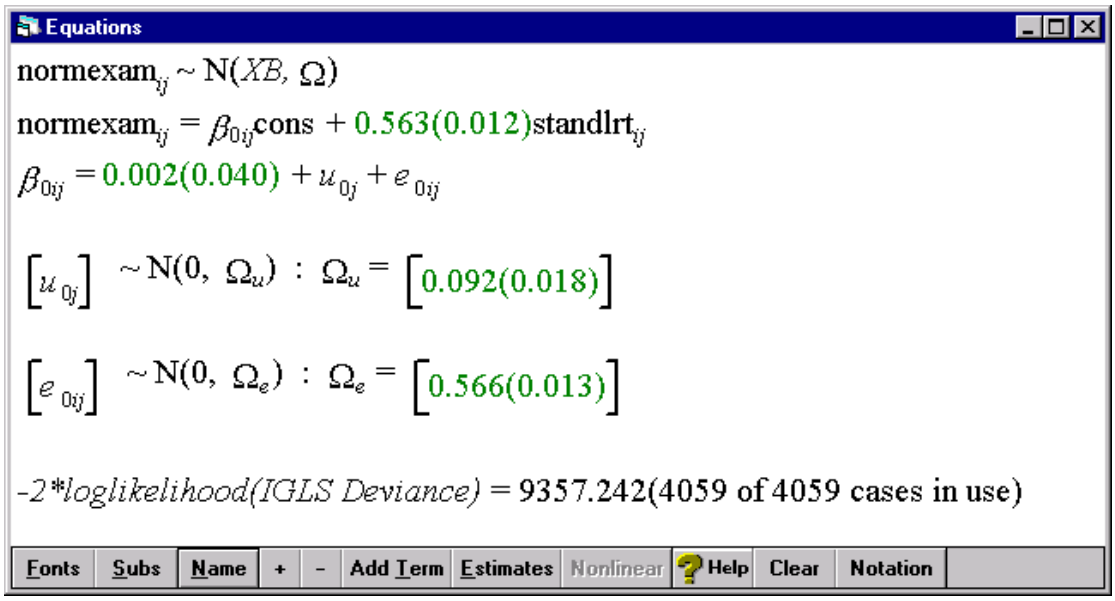
WinBUGS (Spiegelhalter et al., 2000, freely available from http://www.mrc-bsu.cam.ac.uk/bugs) in its earlier guise of BUGS was one of the first Bayesian software packages and is a more general purpose Bayesian estimation engine than the MCMC engine in MLwiN.

They work on a different philosophy of fitting models that can be represented by directed acyclic graphs (DAGs). BUGS has a compiled language which allows the user to specify their model through statements of two types: logical and distributional which between them describe the structure of the DAG and hence the model. Then BUGS compiles this user code and constructs an MCMC estimation engine for the user's model that can be run to give chains of estimates in a similar way to the MLwiN engine.

We will firstly consider a normal response variance components model and show how to fit this model in WinBUGS. We will then go on to consider a model that MLwiN cannot fit which has t-distributed residuals at the school level. It should be noted at this point that most multilevel models are large and so cannot be run using the educational version of WinBUGS and so you will need to have the release version of WinBUGS. Note that you will need to update from the educational version of WinBUGS to run the multilevel models listed here.

Variance Components models in WinBUGS

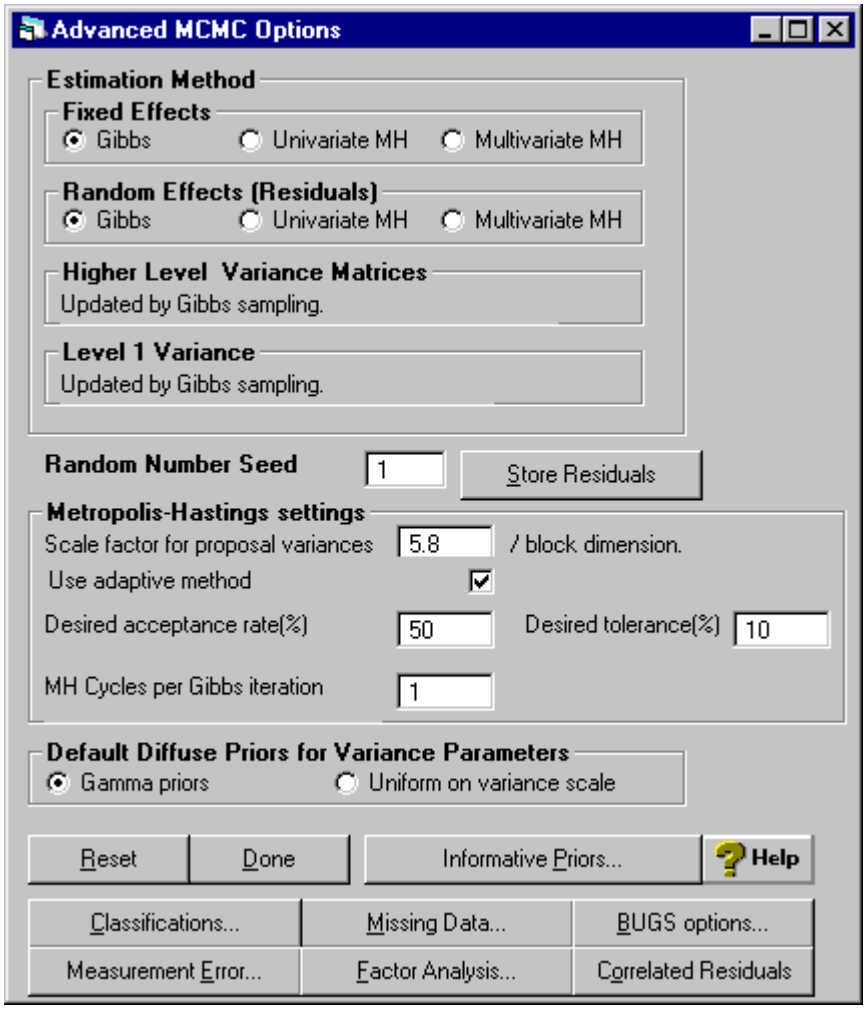
We will consider here an educational dataset where our response is a (normalized) total exam score at age 16 for 4059 children in 65 schools in the UK. We have set up a variance components model with one explanatory variable which is a standardized (London) reading test taken by all the children at age 11 and acts as an intake ability indicator. The model was run using the IGLS (MLE) method in MLwiN.
On convergence we get the following estimates:



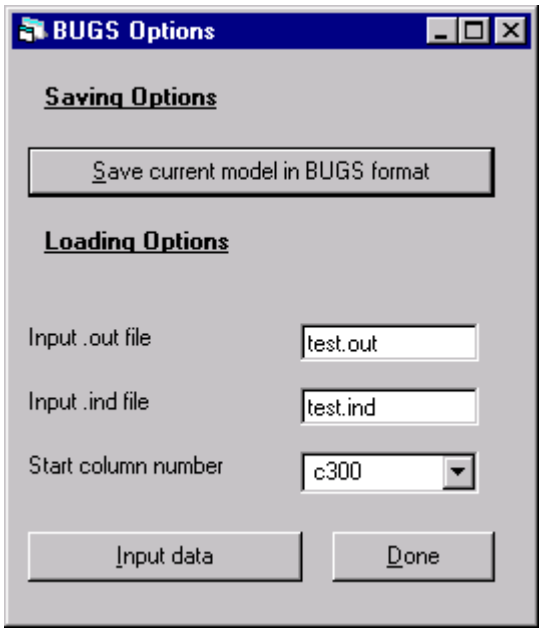
We could now fit this model using MCMC in MLwiN but here we will consider instead using WinBUGS. To get to the BUGS options in MLwiN we need to do the following

Click on the **Estimation Contro**l button
Select the **MCMC** tab
Click on the **Advanced MCMC Options** button

This will bring up the **Advanced MCMC Options** screen that shows the possible MCMC samplers available in MLwiN and looks as follows:



From this screen we can bring up the **BUGS options** by pressing on its button and the following screen appears:



From this screen we can save the BUGS code for the currently set up model or read in the output files that contain parameter traces from BUGS for use in MLwiN (see later). For now we will save our current model in BUGS format so

Click on the large button at the top of the screen.

This will bring up a file save window similar to those for inputting and saving worksheets. For now we will save the file in the default directory as *tutorial.bug*. This will create a file that contains the BUGS model definition, initial values and data. For users of classic BUGS who are used to having three separate files, in WinBUGS the file *tutorial.bug* contains the information from these three files with dividing comment lines between them.

For background information on using WinBUGS it is strongly suggested that the user reads some of the user manual and examples documentation that comes with the package, in particular to become familiar with the WinBUGS notation. For now to fit our model in WinBUGS, we must start the WinBUGS program and read in the file tutorial.bug (from the directory it was saved in) as a text file. Note that you will have to change the Files of type box to '*All files (*.*)*' to see the file tutorial.bug. Having read in the file a window headed *tutorial.bug* will appear containing the information needed by BUGS for this model.

As mentioned earlier the WinBUGS code is split into 3 sections and we will consider these here in turn. Firstly a model definition is required and this consists of a description of the structure of the current problem. The code for our simple variance components problem is as follows:

```
# WINBUGS code generated from MLwiN program

#----MODEL Definition----------

model
{
# Level 1 definition
for(i in 1:N) {
normexam[i] ~ dnorm(mu[i],tau)
mu[i]<- beta[1] * cons[i]
+ beta[2] * standlrt[i]
+ u2[school[i]] * cons[i]
}
# Higher level definitions
for (j in 1:n2) {
u2[j] ~ dnorm(0,tau.u2)
}
# Priors for fixed effects
for (k in 1:2) { beta[k] ~ dflat() }
# Priors for random terms
tau ~ dgamma(0.001,0.001)
sigma2 <- 1/tau
tau.u2 ~ dgamma(0.001,0.001)
sigma2.u2 <- 1/tau.u2
}
```

WinBUGS is a more general modelling package and so there is no standard order to the model description although when the code is generated from MLwiN it will generally have a similar structure. We firstly define the relationship between the response (in this example normexam) and the fixed and random predictor variables.

Note that the column names from MLwiN are used as the variable names in WinBUGS. WinBUGS does have some differences in what it allows as a variable name so if the WinBUGS code will not work it may be that some of your variable names are illegal, for example a column name like '*1995*' will be interpreted as a number in WinBUGS so it is worth renaming such columns in MLwiN.

So we see here that our response is normally distributed and that we have two fixed effects, *beta*[1] and *beta*[2] (always defined as *beta* by the code generator) and one set of random effects, *u2* (always defined as *u*# where # is the level/classification indicator). Note that in WinBUGS the fixed effects, *beta*, and all other vectors always start with index 1 and not 0 so that there will probably not be direct correspondence between the MLwiN and WinBUGS indexing. Next the code defines the random effects *u2* as being Normally distributed before finally giving the priors for the fixed effects and the variances.

WinBUGS has two types of relationship: distributional relationships that are described by the ~ symbol and deterministic relationships that are described by the <- symbol which is also used in the S-plus package. Note that the normal distribution definition in WinBUGS, *dnorm* has two parameters that are the mean and the precision (NOT the variance), hence the deterministic relationship used to calculated the variance. The prior distributions are identical to those used in the MCMC options in MLwiN.

Before running a model in WinBUGS we first need to read in the particular elements of the model using the **Specification** window available from the **Model** menu. After selecting the window containing the model by clicking on it, clicking on the **check model** button should give the message '*model is syntactically correct*' at the bottom of the screen. Next we need to load in the data for the model. Due to the fact that the data is generally the largest part of the file generated by MLwiN it is included after the initial values. For the tutorial.bug example the data section begins as follows:

```
#----Data File---------------------------------

list(N= 4059, n2 = 65,
school = c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
....
```

Again BUGS borrows its notation from S-plus using the convention c(..) to represent a vector of observations. Here we see the first two constants, *N* and *n2* that define the number of level 1 units and level 2 units followed by a list of the *school* identifier for each observation. To load the data into BUGS we need to highlight the *list* identifier at the start of the data list and click on the **load data** button in the **specification** window. If this is successful the message '*data loaded*' will appear at the bottom of the screen. Next we have to combine the data and model definition by clicking on the

compile button. Again if this operation is successful a message appears at the bottom of the screen, this time stating that '*model compiled*'.

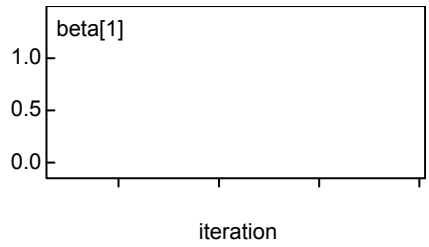
Finally as BUGS uses MCMC methods all unknown parameters will need starting values. These are included in the initial values part of the file that for our example is as follows:

```
#----Initial values file----------------------------

list(beta= c(0.002391,0.563371),
u2 = c( 0.373760,0.502043,0.503888,0.018131,0.240431,0.541395,0.379001,-0.026173,-0.135181,-0.337021,
0.179300,-0.061863,-0.149648,-0.165593,-0.182923,-0.409983,-0.172781,-0.084464,-0.011510,0.214462,
0.244016,-0.435732,-0.489244,0.209408,-0.230472,-0.023543,0.023121,-0.610002,0.240626,0.158475,
0.033280,-0.006457,0.029589,-0.137882,0.128634,-0.181341,-0.189077,-0.153068,0.130317,-0.234439,
0.211543,0.092819,-0.089927,-0.247556,-0.109729,-0.352728,-0.042628,-0.045058,0.042845,-0.302413,
-0.051373,0.381929,0.723313,-0.547252,0.503474,0.009972,0.031894,0.138115,-0.658368,0.225656,
-0.039551,-0.054029,0.535641,0.087691,-0.165765),
tau= 1.767625,
tau.u2= 10.854517)
```

This gives the estimates from the IGLS run for the fixed effects and precisions, and an MLwiN **RESI** command for the initial values for *u2* that are exactly what the MCMC routine in MLwiN uses as starting values. To use these values in WinBUGS we need to highlight the *list* identifier at the start of the initial values and click on the **load inits** button on the **specification** window. This will then give the final message '*initial values loaded; model initialized*'. Note that WinBUGS will generate starting values for any parameters that have not explicitly been given starting values but here we have given all parameters starting values. Note that using the IGLS estimates as starting values is generally more efficient than allowing WinBUGS to choose its own.

We are now ready to run the Gibbs sampler in the WinBUGS package. Before we start the estimation engine we have to tell WinBUGS which parameters we wish to monitor. We will choose the same parameters as MLwiN uses. From the **Inference** menu select the **Samples** options and a window will appear that allows the user to specify which parameters to monitor. In this window we will firstly select the fixed effects by typing *beta* in the **node** box. Note that when a correctly typed parameter is input the **set** button will become enabled. We will also want to use a burnin of 500 iterations so we will also modify the beg value from 1 to 501. After this press the **set** button and the parameter will be set for monitoring. We now need to repeat this procedure with the two variance parameters *sigma2* and *sigma2.u2*.

It should be noted that it is possible in WinBUGS to get dynamic traces of the parameters like those in the **trajectories** window in MLwiN via the **sample** window. If we either type *beta* again in the **node** box or use the scroll button at the side of the box to select *beta* you will see that now all the buttons become enabled. Clicking the **trace** button will give 2 empty trace plots for *beta[1]* (as shown below) and *beta[2]* which will become dynamic when we start updating. Similar traces can be brought up for the two variance parameters.



We are now ready to set the estimation engine running and this is done via the **Update** window found in the **Model** menu. We need to specify the number of updates (including the burn-in) and so we will replace the 1000 here with 5500 as is used in MLwiN. As in MLwiN you can also specify how often to refresh the screen, whether to use thinning and, if WinBUGS needs to use Metropolis Hastings sampling, whether to adapt. There is also the option to use a technique called over-relaxation that improves the mixing of the MCMC chains but takes longer per iteration. The current version of WinBUGS (1.3) will choose the MCMC routines it uses for you depending on the form of the conditional distribution (see section 1.3 of the WinBUGS manual for details).

Now that we have set the number of iterations press the **update** button to start the sampler. After a few minutes (depending on the speed of your processor and how many traces you are viewing) the update counter will reach 5500 and the sampling will be finished. WinBUGS has the nice feature that it will give you a message at the bottom of the screen, for example '*updates took 88s*', stating how long the sampling took which is useful for comparing model run times etc. Generally WinBUGS is slower for models that can also be run in MLwiN but as we will see later it has greater flexibility in the models it can fit and sometimes the MCMC methods it uses are more efficient than the Metropolis Hastings methods used in MLwiN for binomial and Poisson response models.

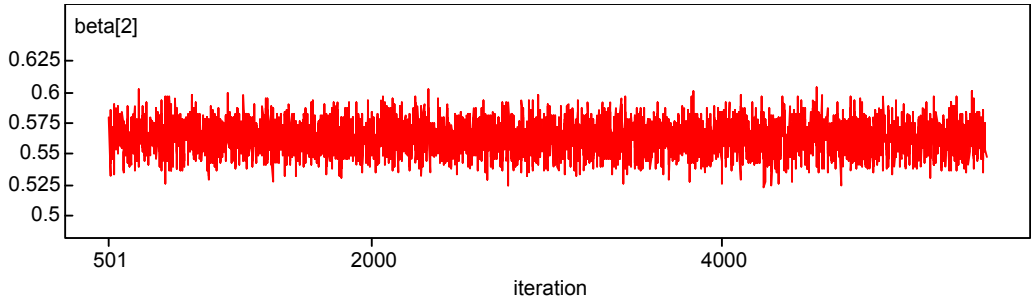
Once the sampling has finished we can now look at the estimates, plots and other information again via the **sample** window. To get summary information, select *beta* in the **node** box and click on the **stats** button. A **node statistics** window will appear giving the following

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
beta[1]	0.002979	0.03995	0.002516	-0.07465	0.004435	0.07912	501	5000
beta[2]	0.5634	0.01264	1.997E-4	0.5388	0.5636	0.5882	501	5000

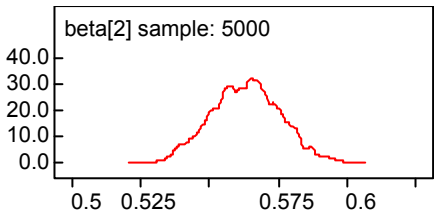
These results are similar to those obtained from MLwiN, which is reassuring, and we can also get similar results for the other parameters as shown below.

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
sigma2.u2	0.09662	0.02019	3.256E-4	0.06415	0.09446	0.1426	501	5000
node	**mean**	**sd**	**MC error**	**2.5%**	**median**	**97.5%**	**start**	**sample**
sigma2	0.5661	0.0127	1.653E-4	0.5416	0.5662	0.5905	501	5000

We can also get trace plots and kernel density plots via the **history** and **density** buttons respectively. Below we see the trace plot for the parameter beta[2]



and the kernel plot.

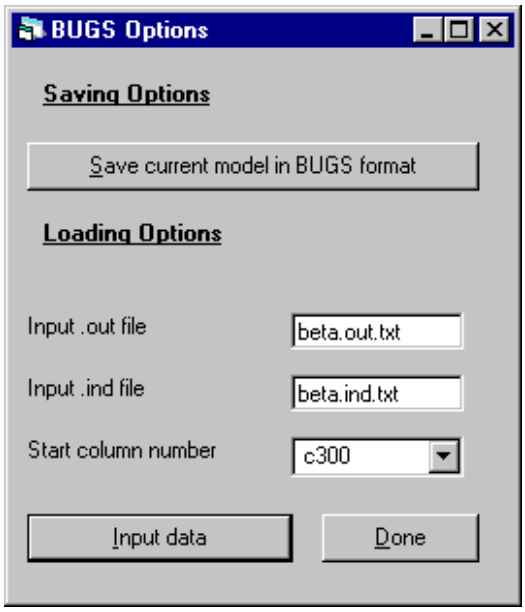


Currently WinBUGS does not allow the smoothing parameter to be changed for the kernel plots so that they look rather crude but this will be changed in later versions.

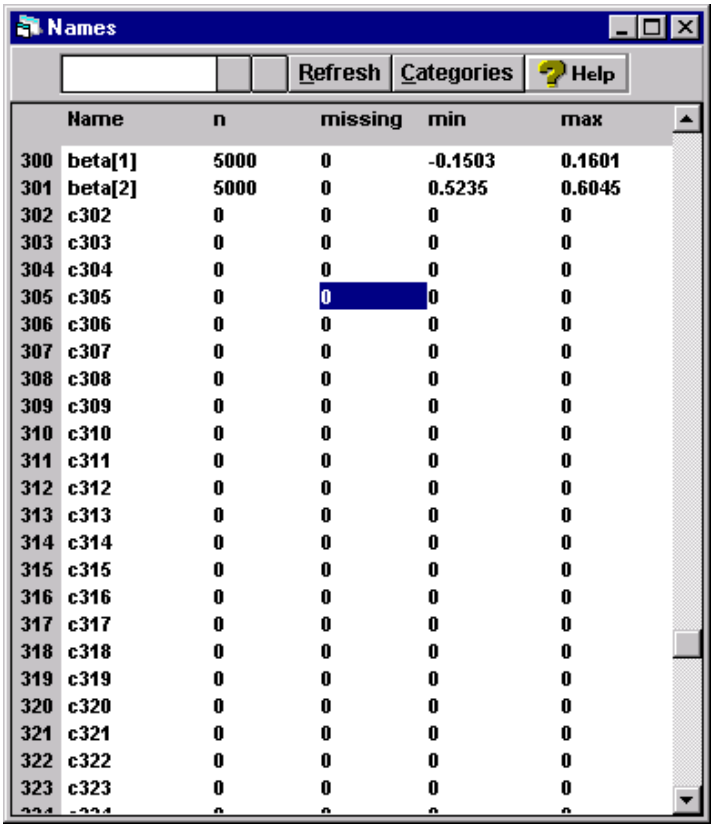
WinBUGS currently produces limited summary statistics and plots itself. Historically the plots and MCMC diagnostics were provided via a suite of S-plus functions called CODA (Best et al. 1995), and WinBUGS also has the option to produce the input files that CODA requires. MLwiN can also use these files to input the parameter chains from WinBUGS into columns in MLwiN.

Here we will only consider the fixed parameter vector *beta* so select this parameter in the node box and press the **coda** button on the **sample** window. This will produce two windows that are labelled '*CODA index*' which contains the variable names and '*CODA for chain 1*' which contains the values for the parameter chains. We will now save these files as text files by clicking on the respective windows and then choosing **Save As** from the **File** menu. We will need to save the files in *plain text* (*.txt) format. We will store the '*CODA index*' file as *beta.ind* and the '*CODA for chain 1*' file as *beta.out* in the same directory as tutorial.bug. Note that these are the extensions that the classic BUGS used for these files but, as we have selected the plain text format, WinBUGS will add an additional .txt to each filename and so the files are actually saved as *beta.ind.txt* and *beta.out.txt*.

Now back in MLwiN if you want to input the beta traces return to the **BUGS options** window that we used earlier (available from the **MCMC advanced options** window).



Here we will need to modify the .out and .in file fields to *beta.out.txt* and *beta.ind.txt* respectively. Note that if you did not put these files in the current directory you will have to include their full path names in the respective boxes. The window should then look as above. Pressing the **Input data** button will now load the chains into columns c300 and c301. To confirm this bring up the **Names** window from the **Data Manipulation** window and scroll down to c300 and you will see the following



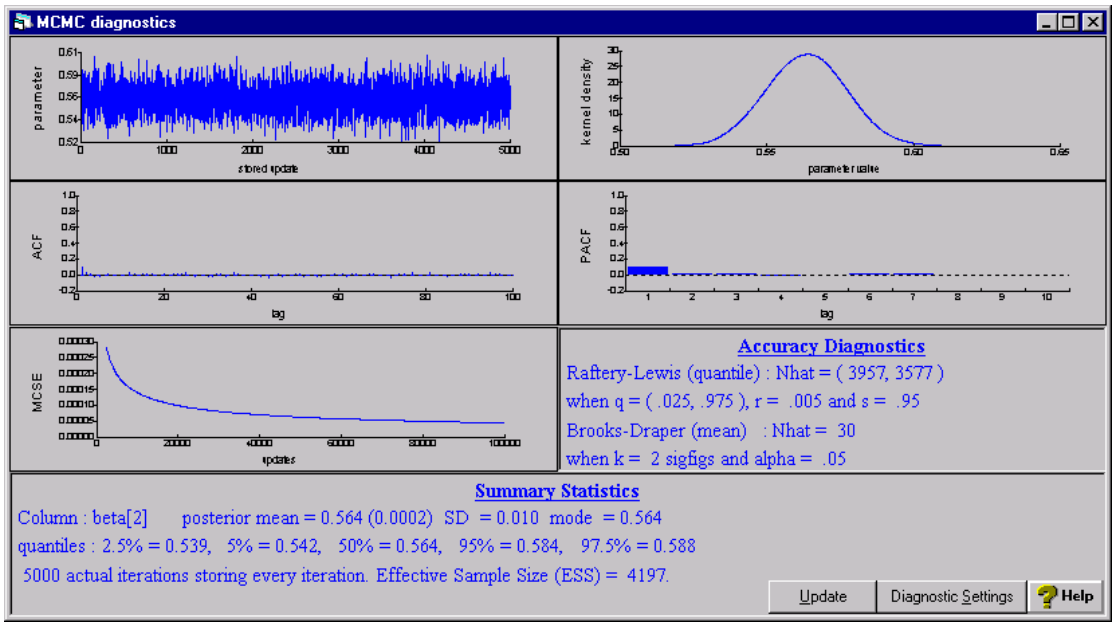
	Name	n	missing	min	max
300	beta[1]	5000	0	-0.1503	0.1601
301	beta[2]	5000	0	0.5235	0.6045
302	c302	0	0	0	0
303	c303	0	0	0	0
304	c304	0	0	0	0
305	c305	0	0	0	0
306	c306	0	0	0	0
307	c307	0	0	0	0
308	c308	0	0	0	0
309	c309	0	0	0	0
310	c310	0	0	0	0
311	c311	0	0	0	0
312	c312	0	0	0	0
313	c313	0	0	0	0
314	c314	0	0	0	0
315	c315	0	0	0	0
316	c316	0	0	0	0
317	c317	0	0	0	0
318	c318	0	0	0	0
319	c319	0	0	0	0
320	c320	0	0	0	0
321	c321	0	0	0	0
322	c322	0	0	0	0
323	c323	0	0	0	0

We can now use the MLwiN MCMC diagnostics on the BUGS output, for example for the slope parameter:

Select the **Column Diagnostics** window from the **Basic Statistics** window. Select the column labelled beta[2]

The window should then look as follows:

Note that this parameter is the fixed effect for the slope that is labelled β_1 in MLwiN. Clicking on **Apply** will give the following diagnostics screen, which is very similar to that given by the MLwiN MCMC sampler in earlier chapters.



We can repeat all of the above procedures for the intercept parameter and the two variances and we will see that we get similar results for all four parameters with both MLwiN and WinBUGS:

	MLwiN	WinBUGS
β_0 (intercept)	0.005(0.042)	0.002 (0.040)
β_1 (slope)	0.563 (0.012)	0.563 (0.013)
σ^2_u (between schools variance)	0.097 (0.021)	0.097 (0.020)
σ^2_e (residual variance)	0.566 (0.013)	0.566 (0.013)
Time for 5,500 iterations	21 seconds	81 seconds

So why have a WinBUGS interface ?

The example we have just gone through will give similar results using both software packages and to use WinBUGS we have to move back and forth between the two packages. Also the estimation engine in WinBUGS is slower, so you may be asking yourself the above question. The interface was written originally as a testing tool to confirm that, when new types of models are programmed into MLwiN we get the same answers as WinBUGS. We recommend that while this version of the MLwiN software is still in development you check that both packages give similar answers.

t distributed school residuals

The main advantage of having a WinBUGS interface, however, is to allow models that have not yet been developed in MLwiN to be fitted using WinBUGS. We will illustrate this by considering alternative distributions for the school level residuals in the tutorial example we considered earlier. In the MLwiN manual (Rasbash et al. 2000) we look at plots of residuals against normal scores to confirm that the normal distributional assumption is a good fit to the data.

The normal distribution is a member of the t distribution family. The t distribution family has an additional degrees of freedom (df) parameter and the normal distribution is the limiting case when this parameter reaches infinity. We will here consider replacing the normal distribution at level 2 with a t distribution where the df parameter has itself got a prior distribution. For this we will use a uniform prior and allow the df parameter to take values in the range 1 to 200. This allows both small values where the distribution has very long tails and large values, which are indistinguishable from the normal distribution.

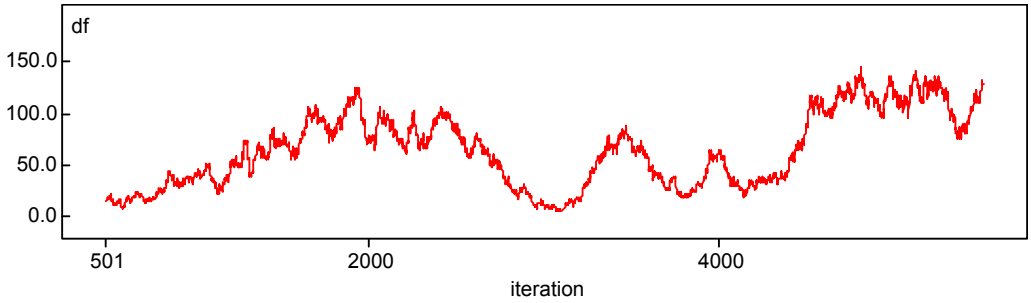
To include this prior we will need to edit the model definition in the file *tutorial.bug*. The new version is as follows (edits in bold font) :

```
#----MODEL Definition----------

model
{
# Level 1 definition
for(i in 1:N) {
normexam[i] ~ dnorm(mu[i],tau)
mu[i]<- beta[1] * cons[i]
+ beta[2] * standlrt[i]
+ u2[school[i]] * cons[i]
}
# Higher level definitions
for (j in 1:n2) {
u2[j] ~ dt(0,tau.u2,df)
}
# Priors for fixed effects
for (k in 1:2) { beta[k] ~ dflat() }
# Priors for random terms
tau ~ dgamma(0.001,0.001)
sigma2 <- 1/tau
tau.u2 ~ dgamma(0.001,0.001)
sigma2.u2 <- 1/tau.u2
df ~ dunif(2,200)
}
```

We will also need to give a starting value for *df* in the initial values file and so we will choose (arbitrarily) *df* = 10. Our initial values file then looks as follows:

```
#----Initial values file----------------------------

list(beta= c(0.002391,0.563371),
u2 = c( 0.373760,0.502043,0.503888,0.018131,0.240431,0.541395,0.379001,-0.026173,-0.135181,-0.337021,
0.179300,-0.061863,-0.149648,-0.165593,-0.182923,-0.409983,-0.172781,-0.084464,-0.011510,0.214462,
0.244016,-0.435732,-0.489244,0.209408,-0.230472,-0.023543,0.023121,-0.610002,0.240626,0.158475,
0.033280,-0.006457,0.029589,-0.137882,0.128634,-0.181341,-0.189077,-0.153068,0.130317,-0.234439,
0.211543,0.092819,-0.089927,-0.247556,-0.109729,-0.352728,-0.042628,-0.045058,0.042845,-0.302413,
-0.051373,0.381929,0.723313,-0.547252,0.503474,0.009972,0.031894,0.138115,-0.658368,0.225656,
-0.039551,-0.054029,0.535641,0.087691,-0.165765),
tau= 1.767625,
tau.u2= 10.854517,
df = 10)
```

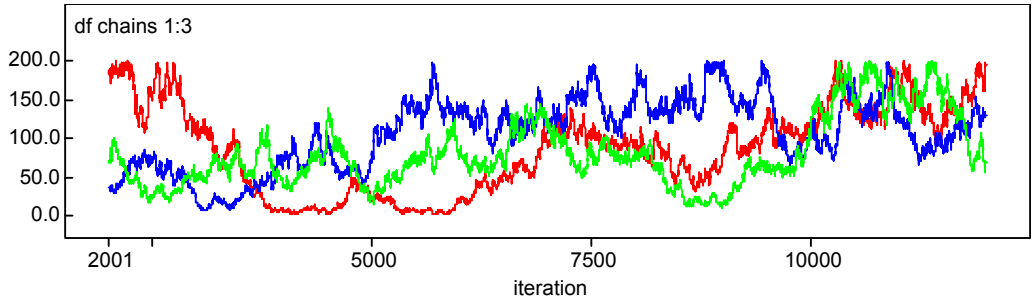
This time we will monitor the same four parameters as before plus the df parameter which we will **set** in the **sample** window. Note that the **adapting** box on the **update** window is ticked because for this model, WinBUGS needs to use Metropolis-Hastings to update the df parameter. Note that the tick disappears when adapting has finished. We again run for 5000 iterations after a burnin of 500 iterations and get the following trace for df



As can be seen this parameter does not mix that well with large autocorrelation in the chain. We can see from the summary statistics below that on this small sample of 5000 iterations we cannot reject the possibility of a heavy-tailed distribution.

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
df	64.29	35.78	4.19	9.757	62.94	129.5	501	5000

In order to investigate the potential of starting value dependence we started three chains with identical parameter starting values except for *df,* which was set to 2,10 and 200 respectively for the 3 runs. To do this in WinBUGS is fairly easy as on the specification window there is a **num of chains** box that we edit to 3 (immediately after checking the model is syntactically correct). Then we have to load the 3 sets of initial values but this simply involves editing the *df=10* line of the initial value file before loading each set. We will increase the burnin to 2000 by changing the **beg** box to 2001 on the **sample** window when we input the parameters we wish to monitor. We will also increase the updates to 12000 so that we have a monitoring run of 10000 iterations after the burnin. Pressing **update**, the three sets of chains will be run concurrently and so this will take longer.



If we look at plots of the 3 sets we see that, although all 3 chains are heavily autocorrelated there is overlap suggesting that sensitivity to the starting value is not a problem. If we look at the summary statistics for the three chains combined we get:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
df	91.9	50.36	2.87	6.241	88.59	189.4	2001	30000

This summary information suggests that a very small degrees of freedom (df) parameter and hence an extremely heavy tailed distribution is not feasible but that a value of df of less than 10 is not out of the question.
As a sensitivity analysis we will instead try fitting a model where the degrees of freedom is assumed known and has value 8 which suggests a slightly long-tailed distribution at level 2. Seltzer (1993) gives Gibbs sampling algorithms for exactly this scenario of a known df parameter.
We will need to simplify our model definition as follows:

```
# WINBUGS code generated from MLwiN program

#----MODEL Definition----------

model
{
# Level 1 definition
for(i in 1:N) {
normexam[i] ~ dnorm(mu[i],tau)
mu[i]<- beta[1] * cons[i]
+ beta[2] * standlrt[i]
+ u2[school[i]] * cons[i]
}
# Higher level definitions
for (j in 1:n2) {
u2[j] ~ dt(0,tau.u2,df)
}
# Priors for fixed effects
for (k in 1:2) { beta[k] ~ dflat() }
# Priors for random terms
tau ~ dgamma(0.001,0.001)
sigma2 <- 1/tau
tau.u2 ~ dgamma(0.001,0.001)
sigma2.u2 <- 1/tau.u2
df <- 8
}
```

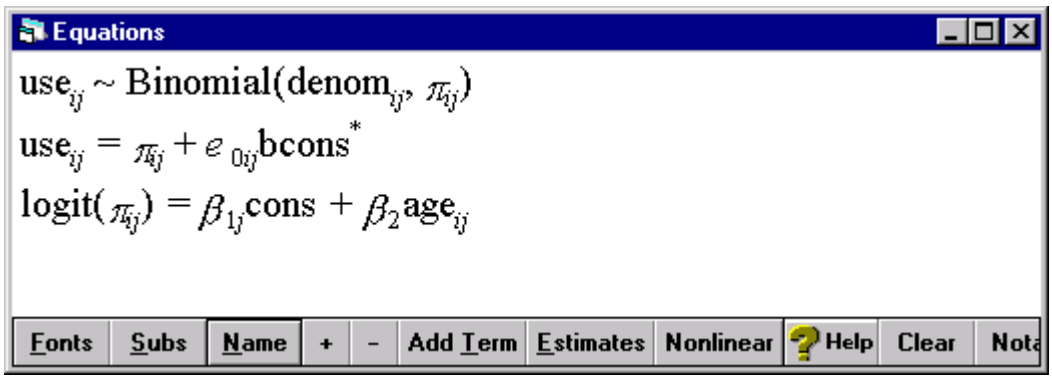
and we will also need to remove the initial values for parameter df as it is now a constant. Again we will monitor *beta, sigma2* and *sigma2.u2*. After running for 5000 iterations after a burnin of 500 we get the following estimates:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
beta[1]	-0.001065	0.04125	0.002364	-0.08322	-7.714E-4	0.07792	501	5000
beta[2]	0.5633	0.0124	1.979E-4	0.5391	0.5636	0.5872	501	5000
sigma2.u2	0.07657	0.01805	4.247E-4	0.04746	0.0745	0.1171	501	5000
sigma2	0.5662	0.01279	1.787E-4	0.5421	0.5659	0.5916	501	5000

Here we see that the fixed effects and level 1 variance are little changed in terms of point estimate and standard errors, suggesting the analysis is robust to different level 2 distributions. The level 2 variance parameter is not directly comparable as the variance of the t distribution is a function of both the *sigma2.u2* and *df* parameters. In fact the variance is 8/6*0.07657 = 0.102, which is slightly higher than for the Normal case.

Using WinBUGS on a Binary response model

We will now consider a second example, this time from demography. We are interested in the use of contraceptives by women in Bangladesh. Our dataset consists of 1934 women who are grouped into 60 districts and we will consider just one predictor the age of the women. Multilevel binary response models are interesting in that we do not have conjugate priors for all parameters and the two packages use different approaches. MLwiN uses random walk Metropolis sampling for these parameters whilst WinBUGS uses adaptive rejection (AR) sampling. Our model is as follows:



We will firstly run IGLS and set up the model for AR sampling in WinBUGS.

Click on the **Start** button
Click on the **Estimation Control** button.
Click on the **MCMC** tab.
Click on the **Advanced MCMC Options** button on the **Estimation Control** window.
Click on the **Bugs Options** button on the **Advanced MCMC Options** window.
Click on the **Save** button
Save the file as 'bang.bug'

We now need to start the WinBUGS program and load the file *bang.bug* as a text file from the directory it has been saved. Note that again we will need to change the **Files of type** box to *'All files (*.*)'* to see the file *bang.bug*. When the file is loaded the model definition will look as follows:

```
# WINBUGS code generated from MLwiN program

#----MODEL Definition----------

model
{
# Level 1 definition
for(i in 1:N) {
use[i] ~ dbin(p[i],denom[i])
logit(p[i]) <- beta[1] * cons[i]
+ beta[2] * age[i]
+ u2[district[i]] * cons[i]
}
# Higher level definitions
for (j in 1:n2) {
u2[j] ~ dnorm(0,tau.u2)
}
# Priors for fixed effects
for (k in 1:2) { beta[k] ~ dflat() }
# Priors for random terms
tau.u2 ~ dgamma(0.001,0.001)
sigma2.u2 <- 1/tau.u2
}
```

Here we see that our response variable *use* is defined as Binomially distributed and we have the logit link function to the predictor variables. We will need to repeat the setting up procedure that we used for the normal response model

Select the **Specification** window from the **Model** menu.
Click on the **Check Model** button.
Highlight the *list* identifier at the start of the data list.
Click on the **Load Data** button.
Click on the **Compile** button.
Highlight the *list* identifier at the start of the initial values list.
Click on the **Load Inits** button.

This will have set up our model and we can now pick our variables to store. Before we do this we will introduce an interesting feature of WinBUGS not mentioned in the first example. If we wish to find out what methods WinBUGS is using to fit the various components of the model we have defined we can use the **Node Info** tool available from the **Info** menu.

If we then type our node name, for example *beta* into the **node** box on this window we can then hit the **method** button and get the following in the **log** window:

```
beta[1] UpdaterDFreeARS.StdUpdater
beta[2] UpdaterDFreeARS.StdUpdater
```

This can be translated to mean that both the fixed effect are being updated by the standard AR sampler. If on the other hand we type the node name *tau.u2* into the **node** box and hit the **method** button we will get:

```
tau.u2 UpdaterGamma.OptUpdater
```

This is because the precision parameter, *tau.u2* is being updated using Gibbs sampling from a Gamma full conditional distribution.

So we now need to tell WinBUGS which parameters we wish to store.

Select the **Samples** window from the **Inference** menu.
Change the **beg** value to 501.
Type *beta* in the **node** box.
Click on the **Set** button.
Type *sigma2.u2* in the **node** box.
Click on the **Set** button.

We have now set up the burnin of 500 iterations and the parameters we wish to store to run the updates we need to use the update window

Select the **Update** window from the **Model** menu.
Change the **updates** value to 5500.
Click on the **update** button.

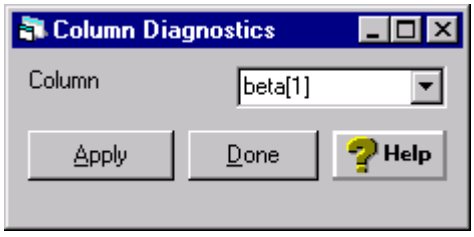
We now need to wait a few minutes for WinBUGS to run. On a Pentium 733Mhz machine the updates take 194 seconds. In order to measure the efficiency of the sampler we will import the chains back into MLwiN and look at the effective sample size (ESS) measure. To do this we need to do the following:

Select the **Samples** window.
Select *beta* from the **node** pull-down list.
Click on the **Coda** button.
Click on the window labeled '*Coda Index'*.
Choose **Save As** from the **File** menu and choose '*plain text (*.txt)'* format.
Save file as '*beta.ind'*.
Click on the window labeled *'Coda for chain 1'*.
Choose **Save As** from the **File** menu and choose '*plain text (*.txt)'* format.
Save file as '*beta.out'*.

Having saved the two files for the fixed effects we can return to MLwiN and use the **BUGS Options** window we used earlier to input the data.

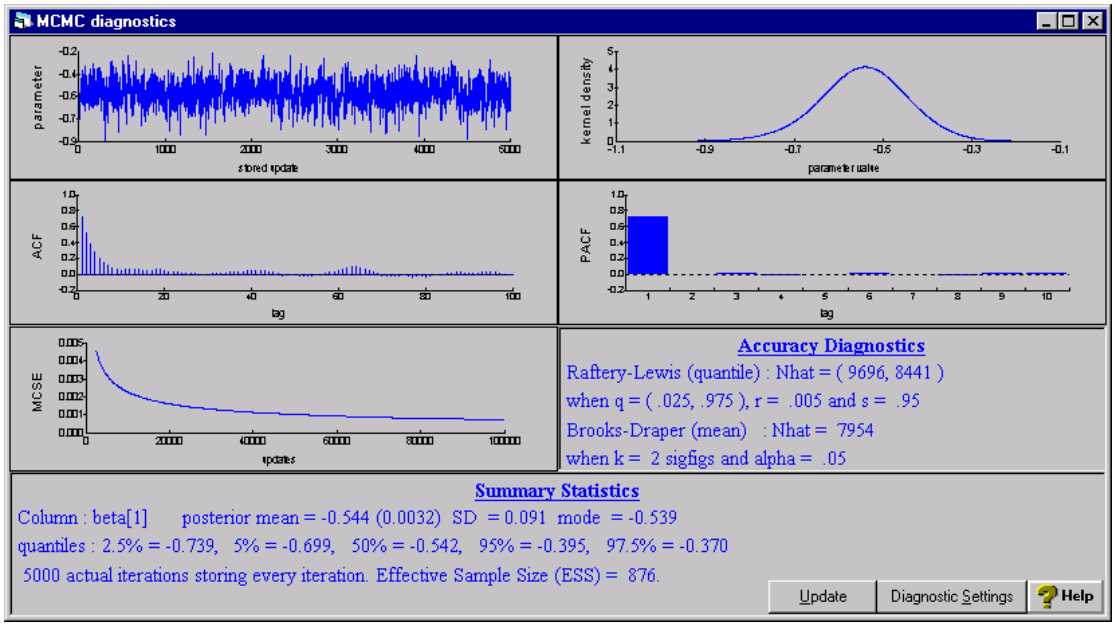
Change **Input .out file** to 'beta.out.txt'
Change **Input .ind file** to 'beta.ind.txt'
Click on **Input Data** button.

This will store *beta[1]* in column c300 and *beta[2]* in column c301 and rename the columns accordingly. We can now look at the diagnostics for the two parameters via the **Column diagnostics** window:



Select the **Column diagnostics** window from the **Basic Statistics** menu.
Select 'beta[1]' from the Column pull-down list.
Click on the **Apply** button.

The diagnostics will then appear as shown below. Here we see that the effective sample size for β_1 is 876 for this sample of size 5,000 due to the autocorrelation in the chain. We can repeat this procedure for β_2 and also we can save the chains for σ^2_u and find its effective sample size. The information for all these parameters will be summarized in the table at the end of this section.



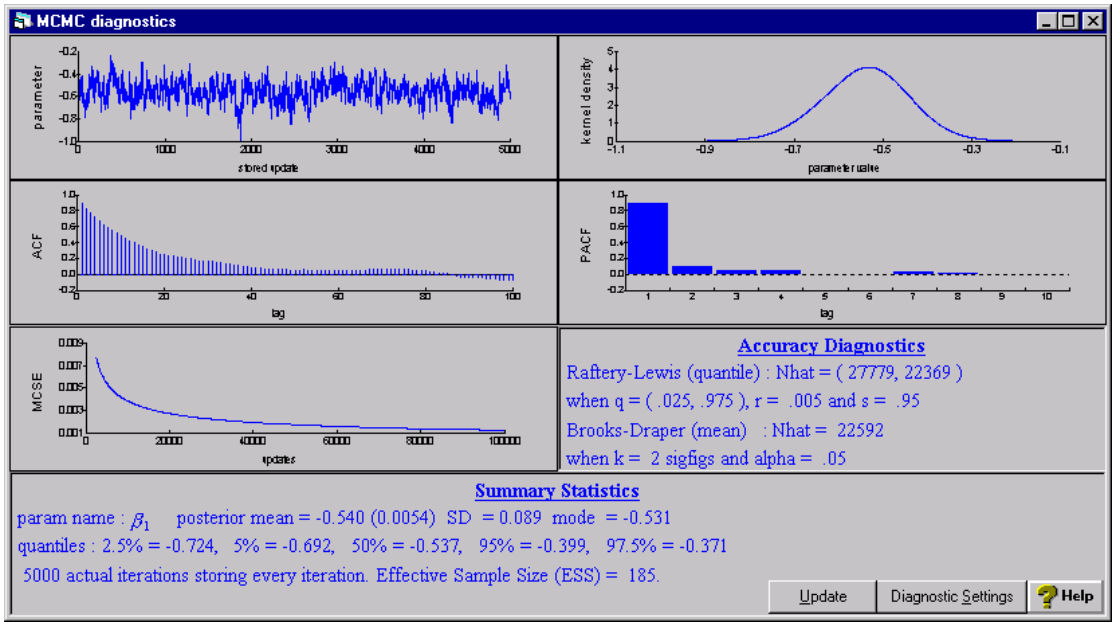
We can now run the same model using Metropolis sampling in MLwiN. The model should currently be set up and MCMC should already be selected and so all we need to do is start the estimation. MLwiN does not by default time estimation but we can set this by enabling the **Smileys** option. The **Smileys** option gives the user a 'smiley face' on the screen that shows when the program is performing an operation or not and as a side effect the timer is enabled.

Select **Close All Windows** from the **Windows** menu.
Select **Smileys** from the **Options** menu.
Click on the **Start** button.

When the estimation finishes a message box will appear saying how long the estimation took (in hours,minutes and seconds). For this example the estimation took 42 seconds on a 733MHz Pentium. Click on the **OK** button to remove this window. We can now get Effective sample sizes and other information from the MCMC diagnostics window.

Select **Trajectories** from the **Model** Menu.
Click in the β_1 chain box.
Select **Yes** to the Calculate MCMC Diagnostics? box.

The diagnostics for β_1 using Metropolis sampling will then be displayed as shown below. Here we have higher autocorrelation and so we see that the effective sample size is only 185.



The results from the two methods can be seen in the following table. The worst mixing parameter is the intercept (β_1) and to get an effective sample size of 5000 will take 18+(5000/876)*176= 17 minutes 3 seconds using AR sampling while Metropolis will take 7+(5000/185)*35 = 15 minutes 53 seconds. This means that even though the AR sampling produces a less correlated chain, Metropolis is sufficiently quicker to give an effective sample of 5,000 in (slightly) less time. Note that Metropolis here has a long burn-in due the adapting method.

Parameter	AR	ESS (AR)	Metropolis	ESS(Metro.)
β_1	-0.544 (0.091)	876	-0.540 (0.089)	185
β_2	0.009 (0.005)	4658	0.009 (0.005)	1209
σ^2_u	0.273 (0.091)	1050	0.273 (0.092)	433
Time	194s(18+176)		42s(7+35)	

So we see that here it appears that Metropolis sampling in MLwiN is (marginally) better for this random effects logistic regression model. Browne and Draper (2000) showed similar results for a couple of random effects logistic regression models. It is however not guaranteed that this will be true for all models. Multilevel Poisson models, which can also be fitted using either method, seem to give highly correlated chains using Metropolis and they may be models that it makes more sense to fit using AR sampling in WinBUGS.

References

Best, N.G., Cowles, M.K. and Vines, S.K. (1995) *CODA: Convergence Diagnosis and Output Analysis software for Gibbs Sampler output: Version 0.3*. Cambridge: Medical Research Council Biostatistics Unit.

Browne, W.J. (2002). *MCMC Estimation in MLwiN*. (in preparation).

Browne, W.J. and Draper, D. (2000) Implementation and performance issues in Bayesian and likelihood fitting of multilevel models. *Computational Statistics* **15**:391-420.

Goldstein, H. (1986) Multilevel mixed response model analysis using iterative generalized least squares. *Biometrika* **73**, 43-56.

Rasbash, J., Browne, W.J., Goldstein, H., Yang, M., Plewis, I., Healy, M., Woodhouse, G., Draper, D., Langford, I. and Lewis, T. (2000) *A User's Guide to MLwiN Version 2.1*, London: Institute of Education, University of London.

Seltzer, M. H. (1993) Sensitivity Analysis for Fixed Effects in the Hierarchical Model: A Gibbs Sampling Approach. *Journal of Educational Statistics* **18**, 207-235.

Spiegelhalter, D.J., Thomas, A. and Best, N.G. (2000) *WinBUGS version 1.3: user manual*. Cambridge: Medical Research Council Biostatistics Unit.