10

Introduction to Bayesian Analysis using WinBUGS

In this chapter we introduce the software package WinBUGS for implementing the use of the Gibbs Sampler to estimate parameters in Bayesian Models. The BUGS software uses Markov Chain Monte Carlo methods to simulate random values from the posterior distributions of the parameters in the model being estimated.

There are two main versions of BUGS for Microsoft Windows. One is the WinBUGS program that can be downloaded from the original WinBUGS website

`http://www.mrc-bsu.cam.ac.uk/bugs/welcome.shtml`

see the WinBUGS link on the left

`http://www.mrc-bsu.cam.ac.ukbugs/winbugs/contents.shtml`

The WinBUGS project is now being further developed as OpenBUGS. The current version of OpenBUGS can be downloaded from the OpenBUGS website. The current and future development plans for WinBUGS are discuss on the Software webpage.

`http://mathstat.helsinki.fi/openbugs`

In this Chapter we use of OpenBUGS 3.0.3 to show some examples of the use of Gibbs Sampling. We also demonstrate the use of BRugs, the R library for running BUGS programs from within R.

In this chapter we refer to WinBUGS as the software packages since when OpenBUGS is installed the software is still entitled WinBUGS.

10.1 What is BUGS?

The BUGS program is designed for Bayesian modeling. The term BUGS stands for Bayesian Inference Using Gibbs Sampling. At the core of the software program is the Gibbs Sampler which is used to sample from the conditional posterior distributions of the parameters, calculated by the program,

from a listing of the model and the prior distributions. The posterior analysis is performed using the simulated Monte Carlo Markov Chain output produced by the program. Posterior statistics and posterior densities can be calculated to produce posterior estimates of the parameters in the model. Trace plots, mean plots, and autocorrelation plots can be made to examine the convergence of the Markov Chain simulations.

BUGS also uses other algorithms to simulate from the conditional posterior distributions. The Metropolis-Hastings Algorithm and Adaptive Rejection methods are also implemented when necessary.

A WinBUGS program is written in code that uses some of the same conventions as R. The program includes a statement of the model, after the word **model** with a semicolon after it, with the model following in curly braces. The data used in the model is specified using a **list**, as defined in R. And finally there is a list of initial values, or **inits**. Examples of the code will be given in the Chapter.

The Warning from the creators of WinBUGS:

Potential users are reminded to be extremely careful if using this program for serious statistical analysis. We have tested the program on quite a wide set of examples, but be particularly careful with types of model that are currently not featured. If there is a problem, WinBUGS might just crash, which is not very good, but it might well carry on and produce answers that are wrong, which is even worse. Please let us know of any successes or failures.

Beware: MCMC sampling can be dangerous!

10.2 Election Poll Example in WinBUGS

In Chapter 8 the Election polling Example 8.1 is the first model we will fit using WinBUGS. Recall the model used for the number of voters in favor of Proposition A, x, was the $\mathsf{BINOM}(n, p)$ with a $\mathsf{BETA}(\alpha, \beta)$ prior on the parameter p. The prior was determined to have parameters $\alpha_0 = 330$ and $\beta_0 = 270$, which centered the prior at 55% with high probability that the parameter p would be between 51% and 59%. The data used in the Example 8.5 was $x = 620$ yes votes out of $n = 1000$ voters sampled. Because this model and prior are conjugate the posterior distribution can be derived, so WinBUGS implements Monte Carlo simulation directly from the posterior and does not implement the Gibbs Sampler. For this example we give the steps necessary to run the model in WinBUGS directly. We use only 2000 interations for this first example so we can see the simulated history plots more clearly. Longer runs of the program would produce more accurate results and would match the exact answers from Chapter 8 to at least 2 decimal places.

The following program is opened in WinBUGS. Note that there are three parts to the program, the **Model** and two lists, one for the **Data** and one for the initial values or **Inits**.

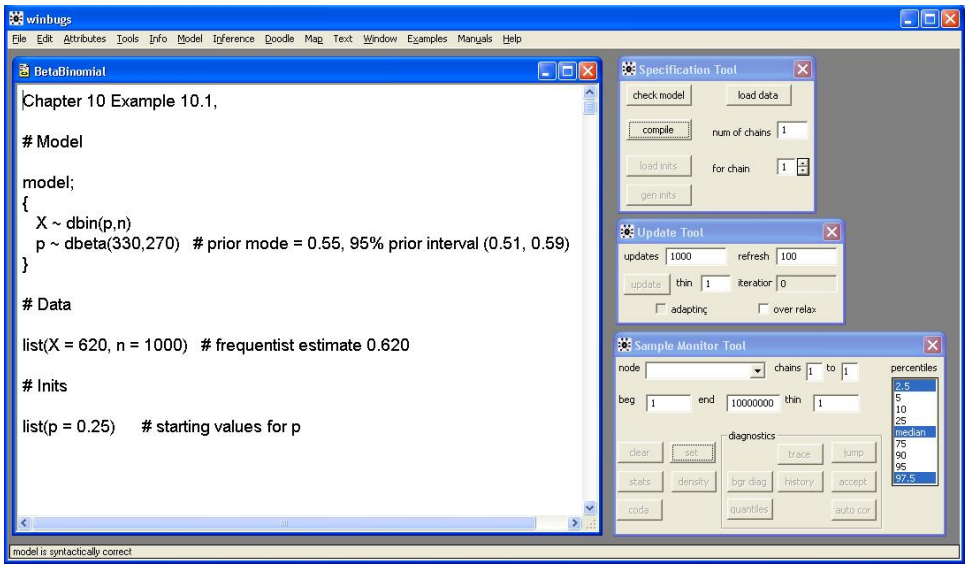

Figure 10.1. Polling Example WinBUGS screenshot

```
# Model
model;
{
   X ~ dbin(p,n)
   p ~ dbeta(330,270)   # prior mode = 0.55, 95% prior interval (0.51, 0.59)
}

# Data
list(X = 620, n = 1000) # observed data

# Inits
list(p = 0.25) # starting values for p
```

To run the program we begin by opening the Specification Tool, from the Model pull down menu, see Figure 10.1 for the view of the the software program and the pull down menus and see Figure 10.2 for the view of the Specification Tool.

To specify the model, highlight the word **model;**, under **# Model** and then click the **check model** button in the Specification Tool. The response in the lower left corner should be "model is syntactically correct." To load the data, highlight the word **list** under **# Data**, and then click the **load data** button in the Specification Tool. The response in the lower left corner should be "data loaded." To compile the model, click the **compile** button in the Specification Tool. The response in the lower left corner should be "model compiled." To load the initial values, highlight the word **list** under **# Inits**,

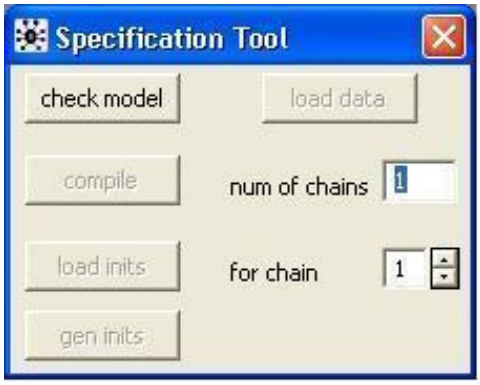

Figure 10.2. Polling Example Specification Tool

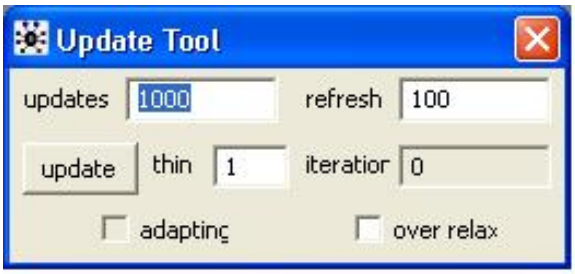

Figure 10.3. Polling Example Update Tool

and then click the **load inits** button in the Specification Tool. The response in the lower left corner should be "model is initialized."

Next we open the Update Tool from the Model pull down menu, see Figure 10.3, and the Sample Monitor Tool from the Inference pull down menu, see Figure 10.4. The WinBUGS program should look like Figure 10.1.

Once the Sample Monitor Tool is open, we can enter the "node" in the model. Since there is one model parameter, we need to enter one node. In the Sample Monitor Tool type the node **p** and click the **set** button. To save the sampled values and analyze the values, enter the star symbol in the node box, *. You should see all of the button appear.

Now we update the model 2000 iterations. In the Update Tool click the **update** button twice. To view the sampled values, in the Sample Monitor Tool, click the **history** button. To view the sampled values of **p**, estimated parameter, the estimated posterior density, and the ACF of the simulated

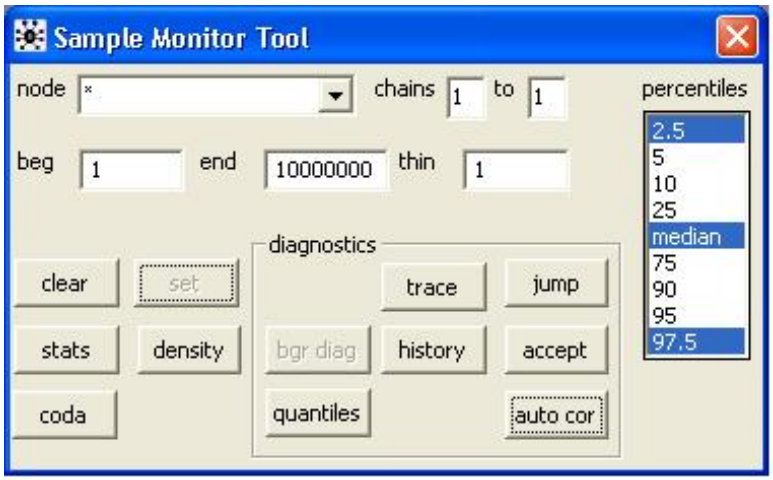

Figure 10.4. Polling Example Sample Monitor Tool

values, in the Sample Monitor Tool, click the **history**, **stats**, **density**, and **auto cor** buttons. The WinBUGS program should look like Figure 10.5.

The results from WinBUGS are close to the computed values from Chapter 8 the Election polling Example 8.6. The posterior mean is 59.39% in favor of Proposition A. And the posterior probability interval for the proportion in favor is (56.9%, 61.79%). These results are very close to the exact values computed earlier, 59.4% and (57.0%, 61.8%). With a longer run of the WinBUGS program we could come closer to the actual values, but with 2000 values we come very close.

The summary results from the simulated values from the posterior distribution of **p** are as follows:

```
   mean    sd      MC_error val2.5pc median val97.5pc start sample
p  0.5939  0.01222 2.389E-4 0.569    0.5939 0.6179     1     2000
```

Note that we could have used a burn-in period by changing the starting value, **beg** from 1 to 1001 in the Sample Monitor Tool. The summary results would change to:

```
    mean    sd    MC_error val2.5pc median val97.5pc start sample
p 0.5935  0.01221 3.51E-4  0.5684   0.5935 0.6178    1001  1000
```

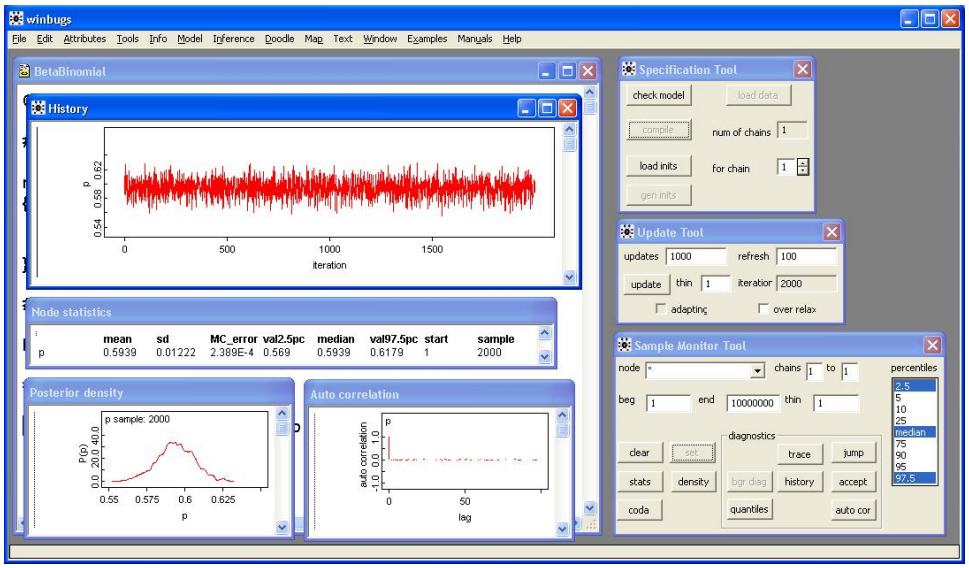

Figure 10.5. Polling Example WinBUGS screenshot

10.3 Reaction Time Example, Pooled Two Sample T-test

One way to study how the human brain performs various tasks is to measure the lengths of time required to complete the tasks. If one task takes longer than another, then the brain must be going through more steps or more complicated steps to process the task that takes longer.

As part of one experiment in a series, a researcher tested 48 students. Each subject was given three nonsense words to memorize: one, two, and three syllables long, and beginning with three different letters of the alphabet. For example, one subject might be given the words ga, duco, and tekabi. As is often the case in a controlled experiment of this kind, the 48 subjects were divided at random into two groups of 24 each.

"Choice" group. Subjects in the first group were seated in front of a computer screen and taught to respond to the following sequence: A message appears on the screen to announce the beginning of a trial, Get Ready. Then a random length of time later a letter appears on the screen, corresponding to one of the three nonsense words. For example, G is the signal for ga. The subject is supposed to say the appropriate word aloud as soon as possible after this letter cue appears. Then another one of the subjects three words is selected for the next trial and the process is repeated, and so on.

The time from when the letter cue appears to when the subject begins to say the word, called the reaction time, is measured in milliseconds. After a number of practice trials for the subject to become familiar with the procedure, 90 trials are measured, including 30 for each word with a random order

of presentation. Simple group. The 24 subjects in the second group were asked to learn a slight variation of the procedure just described. For these subjects, the message that starts each trial reveals the first letter of the nonsense word randomly selected for that trial. For example, Get ready for G is the message for the word ga. Then a random length of time later a star appears as the signal to say the word as soon as possible. Except for the difference in the way the cues are given, both groups are treated exactly the same.

Comparison. In the measured length of time, subjects in the Simple group only have to say a word they have already had time to recall while waiting for the star to appear. The time measured for the Choice group includes both recalling the correct word out of three and saying it. Does the additional mental activity of choosing which word to say take a detectable amount of extra time? If so, how much? In statistical language, we assume that the two groups are samples from theoretical populations of humans performing the Choice and Simple tasks. We wonder whether the mean reaction times of these two populations are different.

The data. The dataset REACTIME contains summarized reaction times in milliseconds (msec) for 24 subjects in the Simple group and for 24 subjects in the Choice group. Results are presented in two separate columns.

c1 Choice Summarized reaction times (in msec) of 24 subjects, letter cue given at end

c2 Simple Summarized reaction times (in msec) of 24 subjects, letter cue given to start

Additional columns in this dataset are introduced in Section 4. Recall that the 48 subjects in this experiment were randomized into the Simple and Choice groups. In particular, the first subject listed in c1 is different from the first in c2.

... end of quote from Trumbo.

The Reaction Time data can be analyzed using a comparison of two normal models. In Classical Statistics a Pooled Independent Two Sample T-test could be performed. Here we develop a Bayesian model for the two groups assuming normal data for each group and assuming a common variance for both groups (Choice $x_1 \sim N(\mu_1, \tau)$ and Simple $x_2 \sim N(\mu_2, \tau)$, where $\sigma^2 = 1/\tau$). The

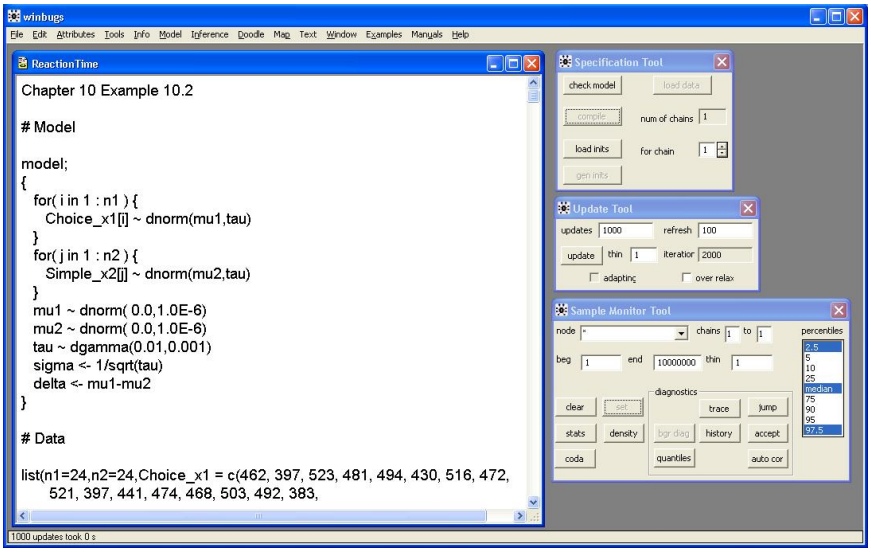

Figure 10.6. ReactionTime Example WinBUGS screenshot

model assumes flat conjugate normal priors for the means μ_1 and μ_2 and assumes a flat conjugate gamma prior for the precision τ. And two other model parameters are defined, $\sigma = 1/\sqrt{\tau}$, the standard deviation, and $\delta = \mu_1 - \mu_2$, the difference between the two means.

The following program is opened in WinBUGS, see Figure 10.6. Note that the code for the Normal distribution is in terms of the mean μ and the precision τ.

```
# Model

model;
{
   for( i in 1 : n1 ) {
      Choice_x1[i] ~ dnorm(mu1,tau)
   }
   for( j in 1 : n2 ) {
      Simple_x2[j] ~ dnorm(mu2,tau)
   }
   mu1 ~ dnorm( 0.0,1.0E-6)
   mu2 ~ dnorm( 0.0,1.0E-6)
   tau ~ dgamma(0.01,0.001)
   sigma <- 1/sqrt(tau)
   delta <- mu1-mu2
}
```

We begin by opening the Specification Tool. From the Model pull-down menu select Specification. To specify the model, highlight the word **model;** under **# Model** and then click the **check model** button in the Specification Tool. To load the data, highlight the word **list** under **# Data**, and then click the **load data** button in the Specification Tool. To compile the model, click the **compile** button in the Specification Tool. To load the initial values, highlight the word **list** under **# Inits**, and then click the **load inits** button in the Specification Tool. The response in the lower right corner should be "model is initialized."

Next we open the Update Tool. From the Model pull-down menu select Update. We also open the Sample Monitor Tool. From the Inference pull-down menu select Samples. Once the Sample Monitor Tool is open, we can enter the nodes in the model. Since there are five model parameters, we need to enter five nodes.

1. Type mu1, click **set**.
2. Type mu2, click **set**.
3. Type tau, click **set**.
4. Type sigma, click **set**.
5. Type delta, click **set**.

To save all the simulated values of the model nodes, enter a $*$ in the node box. All of the buttons on the Sample Monitor Tool should appear.

Now update the model 2000 iterations for a burn-in period. In the Update Tool click the **update** button twice. Now select the **trace** button in the Sample Monitor Tool to watch the sample values as they are simulated. Run the simulation 2000 more iterations by clicking the **update** button in the Update Tool two more times, to produce the simulated values to analyze as sampled values from the posterior distributions. Note that in the Dynamic trace window the simulated values are automatically updated, see Figure 10.7.

To view the sampled values, in the Sample Monitor Tool, click the **history** button. To see the posterior analysis after burn-in, change the **beg** value to 2001, and click the **stats** and **density** buttons, see Figure 10.8. The results are:

```
mean      sd      MC_error val2.5pc median    val97.5pc start sample
delta 168.9     13.76 0.2404    141.3      169.3    197.0     2001 2000
mu1     470.4    9.69 0.2021    451.7      470.6    489.4     2001 2000
mu2     301.6     10.01 0.2161    281.5      301.8    321.0     2001
2000
sigma 48.54     5.168 0.1007    39.5          48.09    59.62     2001
2000
tau     4.387E-4 9.195E-51.75E-6    2.826E-4      4.325E-4  6.413E-4
2001 2000
```

There is a clear difference in reachtion time for the Simple group. To see this we can examine the model parameter $\delta = \mu_1 - \mu_2$ which is estimated

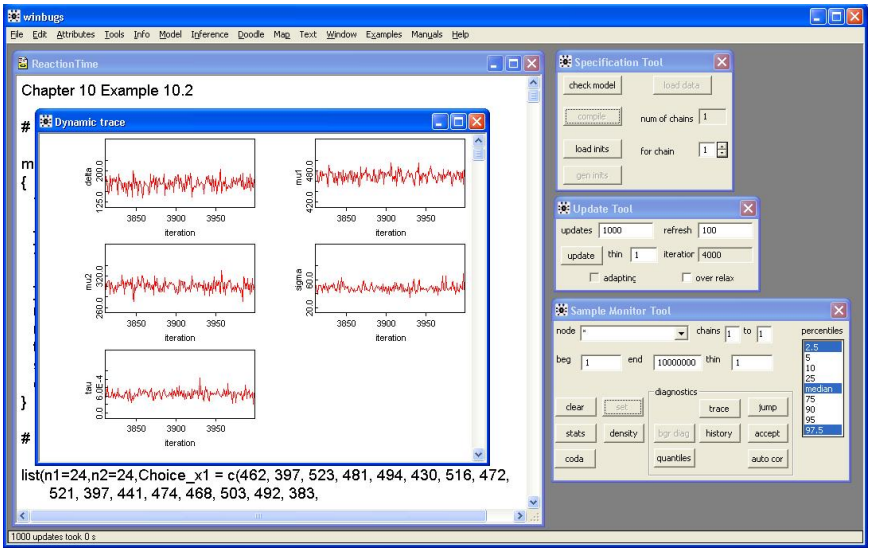

Figure 10.7. ReactionTime Example Dynamic Trace

to have a posterior mean of 168.8 with a posterior 95% interval of (141.1, 196.7). Indicating that there is a 95% chance that the difference $\mu_1 - \mu_2$ is in the interval, and note that the interval does not include 0, this indicates that there is a significant difference between the two group means. The Simple group has a reaction time that is 168.8 less that the Choice group.

There is another way to run the WinBUGS program, it is through writing a script. Here is an example of the script needed to run the model for the Reaction Time data.

```
modelCheck('C:/ReactionTime/ReactionTimeModel.odc')
modelData('C:/ReactionTime/ReactionTimeData.odc')
modelCompile(1)
modelInits('C:/ReactionTimeInit.odc')
modelUpdate(1000)
samplesSet(mu1)
samplesSet(mu2)
samplesSet(tau)
samplesSet(sigma)
samplesSet(delta)
modelUpdate(1000)
samplesStats('*')
samplesDensity('*')
samplesHistory('*')
samplesAutoC('*')
```

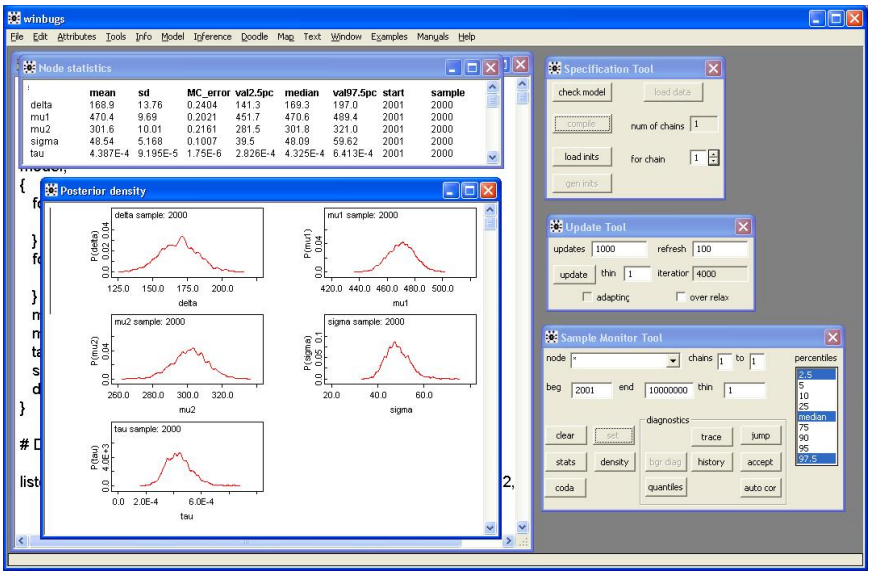

Figure 10.8. ReactionTime Example Node statistics and Posterior density

Three files need to be created, a model file, a data file, and an initial values file. The files contain the three part of the Reaction Time WinBUGS program from above.

To run the script, open the Log to see the running of the Script. From the Info pull-down menu select Open Log. Then click on the window in WinBUGS containing the Script, from the Model pull-down menu select Script. The model should run. In Log you should see the calls of each command in the script and windows should appear with the history and other results, see Figure 10.9.

10.4 Die Data Example, One-way random effects ANOVA

Compare the results here with Exercise 9.20.

```
# Model

model;
{
for( i in 1 : batches ) {
mu[i] ~ dnorm(theta, tau.btw)
for( j in 1 : samples ) {
y[i , j] ~ dnorm(mu[i], tau.with)
```

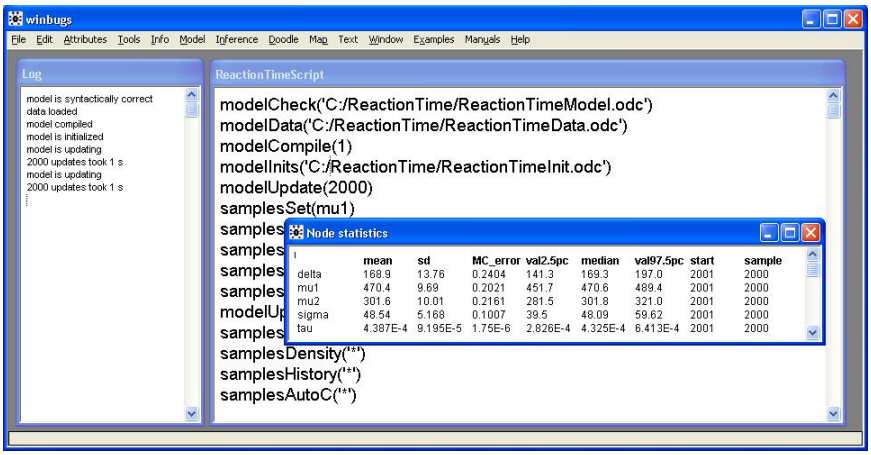

Figure 10.9. ReactionTime Example Script

```
}
}
theta ~ dnorm(0.0, 1.0E-10)

# prior for within-variation
sigma2.with <- 1/tau.with
tau.with ~ dgamma(0.001, 0.001)
sigma.with <- sqrt(sigma2.with)
sigma.btw <- sqrt(sigma2.btw)

# Prior for ICC
ICC ~ dunif(0,1)
sigma2.btw <- sigma2.with *ICC/(1-ICC)
tau.btw <- 1/sigma2.btw
}

# Data

list(batches = 30, samples = 5,
y = structure(.Data = c(959, 976, 1015, 1003, 971,
977, 990, 972, 977, 1004,
987, 988, 1008, 973, 963,
974, 1004, 995, 991, 977,
1011, 1009, 984, 1033, 1014,
1058, 1063, 1057, 1062, 1072,
972, 965, 968, 973, 974,
1028, 1047, 1044, 1039, 1047,
973, 1007, 998, 1030, 1001,
1042, 991, 1016, 1014, 1018,
```

```
1013, 989, 1016, 992, 1010,
991, 1001, 1024, 977, 999,
978, 939, 960, 990, 990,
1026, 1032, 1017, 1018, 998,
1030, 1024, 1054, 1046, 1061,
1038, 1024, 1005, 990, 1018,
1020, 997, 1019, 1008, 1016,
1039, 1024, 1053, 1038, 1019,
1023, 1009, 1037, 993, 1007,
1006, 994, 1002, 1011, 1007,
993, 1003, 979, 990, 983,
1011, 975, 988, 1017, 999,
982, 1006, 1003, 975, 991,
970, 935, 951, 960, 972,
1008, 987, 977, 981, 1004,
994, 980, 1016, 1023, 1009,
971, 969, 973, 964, 951,
1070, 1050, 1041, 1055, 1047,
985, 995, 998, 995, 989,
1000, 987, 979, 1013, 1008),
.Dim = c(30, 5)))

# Inits

list(theta=1500, tau.with=1,ICC=0.5)
```

10.5 Old Faithful Data, Linear Regression

Plot the fitted Bayesian estimated line and show the idea of a prediction interval for a new value. Also, it is reasonable to show this a a BRugs program in R.

Note there is serial correlation in the sampled values. See Figure 6.2

```
# Model

model;
{
   for( i in 1 : N ) {
      mu[i] <- alpha + beta * (x[i] - xbar)
   }
   for( i in 1 : N ) {
      Y[i] ~ dnorm(mu[i],tau)
   }
   alpha ~ dnorm( 0.0,1.0E-6)
   beta ~ dnorm( 0.0,1.0E-6)
   tau ~ dgamma(0.001,0.001)
```

```
   sigma <- sqrt(1 / tau)
}

# Data     WaitNext = alpha + beta DurLast

list(x = c(4.4, 3.9, 4.0, 4.0, 3.5, 4.1, 2.3, 4.7, 1.7, 4.9,
     1.7, 4.6, 3.4, 4.3, 1.7, 3.9, 3.7, 3.1, 4.0, 1.8,
     4.1, 1.8, 3.2, 1.9, 4.6, 2.0, 4.5, 3.9, 4.3, 2.3,
     3.8, 1.9, 4.6, 1.8, 4.7, 1.8, 4.6, 1.9, 3.5, 4.0,
     3.7, 3.7, 4.3, 3.6, 3.8, 3.8, 3.8, 2.5, 4.5, 4.1,
     3.7, 3.8, 3.4, 4.0, 2.3, 4.4, 4.1, 4.3, 3.3, 2.0,
     4.3, 2.9, 4.6, 1.9, 3.6, 3.7, 3.7, 1.8, 4.6, 3.5,
     4.0, 3.7, 1.7, 4.6, 1.7, 4.0, 1.8, 4.4, 1.9, 4.6,
     2.9, 3.5, 2.0, 4.3, 1.8, 4.1, 1.8, 4.7, 4.2, 3.9,
     4.3, 1.8, 4.5, 2.0, 4.2, 4.4, 4.1, 4.1, 4.0, 4.1,
     2.7, 4.6, 1.9, 4.5, 2.0, 4.8, 4.1),
     Y = c(78,  74,  68,  76,  80,  84,  50,  93,  55,  76,
     58,  74,  75,  80,  56,  80,  69,  57,  90,  42,
     91,  51,  79,  53,  82,  51,  76,  82,  84,  53,
     86,  51,  85,  45,  88,  51,  80,  49,  82,  75,
     73,  67,  68,  86,  72,  75,  75,  66,  84,  70,
     79,  60,  86,  71,  67,  81,  76,  83,  76,  55,
     73,  56,  83,  57,  71,  72,  77,  55,  75,  73,
     70,  83,  50,  95,  51,  82,  54,  83,  51,  80,
     78,  81,  53,  89,  44,  78,  61,  73,  75,  73,
     76,  55,  86,  48,  77,  73,  70,  88,  75,  83,
     61,  78,  61,  81,  51,  80,  79), xbar = 3.461, N = 107)

# Inits

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

10.6 Problems

1. For the Election polling example, run the program for 10,000 iterations and see if the posterior estimate of p and the posterior 95% probability interval is closer to the values computed earlier in Chapter 8.
2. Suppose only 100 people were polled in the Election polling example presented at the beginning of this Chapter. And suppose the same proportion of voters in favor of Proposition A responded in favor. So there were $x = 62$ yes votes out of the $n = 100$ voters polled.
   a) Modify the program given at the beginning of this Chapter to reflect the change in the data. Run the program and report the estimated posterior proportion of voters in favor of Proposition A.
   b) Because of the reduction in the amount of data collected, is there an increase in the influence of the prior on the posterior?