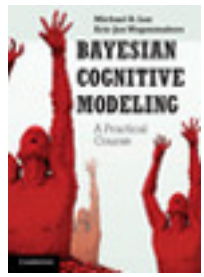

Bayesian Cognitive Modeling

A Practical Course

Michael D. Lee, Eric-Jan Wagenmakers

Book DOI: http://dx.doi.org/10.1017/CBO9781139087759

Online ISBN: 9781139087759

Hardback ISBN: 9781107018457

Paperback ISBN: 9781107603578

Chapter

2 - Getting started with WinBUGS pp. 16-34

Chapter DOI: http://dx.doi.org/10.1017/CBO9781139087759.004

Cambridge University Press

2 Getting started with WinBUGS

with Dora Matzke

Throughout this book, you will use the WinBUGS (Lunn et al., 2000, 2009) software to work your way through the exercises. Although it is possible to do the exercises using the graphical user interface provided by the WinBUGS package, you can also use the Matlab or R programs to interact with WinBUGS.

In this chapter, we start by working through a concrete example using just WinBUGS. This provides an introduction to the WinBUGS interface, and the basic theoretical and practical components involved in Bayesian graphical model analysis. Completing the example will also quickly convince you that you do *not* want to rely on WinBUGS as your primary means for handling and analyzing data. It is not especially easy to use as a graphical user interface, and does not have all of the data management and visualization features needed for research.

Instead, we encourage you to choose either Matlab or R as your primary research computing environment, and use WinBUGS as an "add-on" that does the computational sampling part of analyses. Some WinBUGS interface capabilities will remain useful, especially in the exploratory stages of research. But either Matlab or R will be primary. Matlab and R code for every example in this book, as well as the scripts that implement the models in WinBUGS, are all available at `www.bayesmodels.com`.

This chapter first does a concrete example in WinBUGS, then re-works it in both Matlab and R. You should pay particular attention to the section that features your preferred research software. You will then be ready for the following chapters, which assume you are working in either Matlab or R, but understand the basics on the WinBUGS interface.

2.1 Installing WinBUGS, Matbugs, R, and R2WinBugs

2.1.1 Installing WinBUGS

WinBUGS is currently free software, and is available at `http://www.mrc-bsu.cam.ac.uk/bugs/`. Download the most recent version, including any patches, and make sure you download and apply the registration key. Some of the exercises in this book might work without the registration key, but some of them will not. You can download WinBUGS and the registration key directly from `http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml`. A note to Windows 7 users: when you

install the patch and the registration key, make sure that you have first opened WinBUGS using the "Run as administrator" option (right-click on the WinBUGS icon to make this option available); next, go to `File` → `New`, copy-paste the code (i.e., patches or key), and then select `Tools` → `Decode` → `Decode All`.

2.1.2 Installing Matlab and Matbugs

Matlab is a commercial software package, and is available at `http://www.mathworks.com/`. As far as we know, any reasonably recent version of Matlab should let you do the exercises in this book. Also, as faras we know, no toolboxes are required. To give Matlab the ability to interact with WinBUGS, download the freely available `matbugs.m` function and put it in your Matlab working directory. You can download `matbugs.m` directly from `https://code.google.com/p/matbugs`.

2.1.3 Installing R and R2WinBUGS

R is a free software package, and is available at `http://www.r-project.org/`: click "download R," choose your download location, and proceed from there. Alternatively, you can download the Windows version of R directly from `http://cran.xl-mirror.nl/`. To give R the ability to interact with WinBUGS, you have to install the `R2WinBUGS package`. To install the `R2WinBUGS` package, start R and select the `Install Package(s)` option in the `Packages` menu. Once you choose your preferred CRAN mirror, select `R2WinBUGS` in the `Packages` window and click on `OK`.

2.2 Using the applications

2.2.1 An example with the binomial distribution

We will illustrate the use of WinBUGS, Matbugs, and R2WinBUGS by means of the same simple example from Chapter 1, which involved inferring the rate of success for a binary process. A binary process is anything where there are only two possible outcomes. An inference that is often important for these sorts of processes is the underlying rate at which the process takes one value rather than the other. Inferences about the rate can be made by observing how many times the process takes each value over a number of trials.

Suppose that one of the outcomes (e.g., the number of successes) happens on k out of n trials. These are known, or observed, data. The unknown variable of interest is the rate θ at which the outcomes are produced. Assuming that what happened on one trial does not influence the other trials, the number of successes k follows a binomial distribution, $k \sim \text{Binomial}(\theta, n)$. This relationship means that by observing the k successes out of n trials, it is possible to update our knowledge

Box 2.1 Our graphical model notation

There is no completely agreed standard notation for representing graphical models visually. It is always the case that nodes represent variables, and the graph structure connecting them represents dependencies. And it is almost always the case that plates are used to indicate replication. Beyond that core, there are regularities and family resemblances in the approaches used by numbers of authors and fields, but not adherence to a single standard. In this book, we make distinctions between: *continuous* versus *discrete* valued variables, using circular and square nodes; *observed* and *unobserved* variables, using shaded and unshaded nodes; and *stochastic* versus *deterministic* variables, using single- and double-bordered nodes. Alternative or additional conventions are possible, and could be useful. For example, our notation does not distinguish between observed variables that are data (e.g., the decision a subject makes in an experiment) and observed variables that are known properties of an experimental design (e.g., the number of trials a subject completes). It is also possible to argue that, for deterministic variables, it is the functions that are deterministic, and so the arrows in the graph, rather than the nodes, should be double-bordered.

about the rate θ. The basic idea of Bayesian analysis is that what we know, and what we do not know, about the variables of interest is always represented by probability distributions. Data like k and n allow us to update prior distributions for the unknown variables into posterior distributions that incorporate the new information.

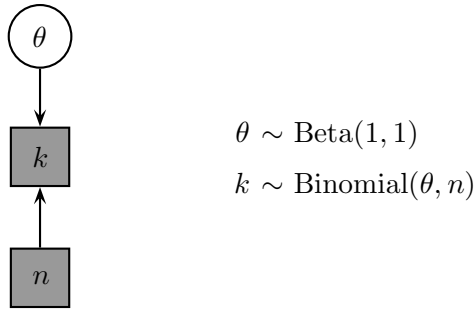

Fig. 2.1 Graphical model for inferring the rate of a binary process.

The graphical model representation of our binomial example is shown in Figure 2.1. The nodes represent all the variables that are relevant to the problem. The graph structure is used to indicate dependencies between the variables, with children depending on their parents. We use the conventions of representing unobserved

variables without shading and observed variables with shading, and continuous variables with circular nodes and discrete variables with square nodes.

Thus, the observed discrete numbers of successes k and number of trials n are represented by shaded and square nodes, and the unknown continuous rate θ is represented by an unshaded and circular node. Because the number of successes k depends on the number of trials n and on the rate of success θ, the nodes representing n and θ are directed towards the node representing k. We will start with the prior assumption that all possible rates between 0 and 1 are equally likely. We will thus assume a uniform prior $\theta \sim \text{Uniform}(0,1)$, which can equivalently be written in terms of a beta distribution as $\theta \sim \text{Beta}(1,1)$.

One advantage of using the language of graphical models is that it gives a complete and interpretable representation of a Bayesian probabilistic model. Another advantage is that WinBUGS can easily implement graphical models, and its various built-in MCMC algorithms are then able to do all of the inferences automatically.

2.2.2 Using WinBUGS

WinBUGS requires the user to construct three text files: one that contains the data, one that contains the starting values for the model parameters, and one that contains the model specification. The WinBUGS model code associated with our binomial example is available at `www.bayesmodels.com`, and is shown below:

```
# Inferring a Rate
model{
   # Prior Distribution for Rate Theta
   theta ~ dbeta(1,1)
   # Observed Counts
   k ~ dbin(theta,n)
}
```

Note that the uniform prior on θ is implemented here as $\theta \sim \text{Beta}(1,1)$. An alternative specification may seem more direct, namely $\theta \sim \text{Uniform}(0,1)$, denoted `dunif(0,1)` in WinBUGS. These two distributions are mathematically equivalent, but in our experience WinBUGS has fewer computational problems with the beta distribution implementation.

Implementing the model shown in Figure 2.1, and obtaining samples from the posterior distribution of θ, can be done by following the sequence of steps outlined below. At the present stage, do not worry about some of the finer details, as these will be clarified in the remainder of this book. Right now, the best you can do is simply to follow the instructions below and start clicking away.

1. Copy the model specification text above and paste it in a text file. Save the file, for example as `Rate_1.txt`.
2. Start WinBUGS. Open your newly created model specification file by selecting the `Open` option in the `File` menu, choosing the appropriate directory, and double-clicking on the model specification file. Do not forget to select files of type "txt," or you might be searching for a long time. Now check the syntax

of the model specification code by selecting the `Specification` option in the `Model` menu. Once the `Specification Tool` window is opened, as shown in Figure 2.2, highlight the word "model" at the beginning of the code and click on `check model`. If the model is syntactically correct and all parameters are given priors, the message "model is syntactically correct" will appear in the status bar all the way in the bottom left corner of the WinBUGS window. (But beware: the letters are very small and difficult to see.)

3. Create a text file that contains the data. The content of the file should look like this:

```
list(
k = 5,
n = 10
)
```

   Save the file, for example as `Data.Rate_1.txt`.

4. Open the data file and load the data. To open the data file, select the `Open` option in the `File` menu, select the appropriate directory, and double-click on the data file. To load the data, highlight the word "list" at the beginning of the data file and click on `load data` in the `Specification Tool` window, as shown in Figure 2.2. If the data are successfully loaded, the message "data loaded" will appear in the status bar.
5. Set the number of chains. Each chain is an independent run of the same model with the same data, although you can set different starting values for each chain.[1] Considering multiple chains provides a key test of convergence. In our binomial example, we will run two chains. To set the number of chains, type "2" in the field labelled `num of chains` in the `Specification Tool` window, shown in Figure 2.2.
6. Compile the model. To compile the model, click on `compile` in the `Specification Tool` window, shown in Figure 2.2. If the model is successfully compiled, the message "model compiled" will appear in the status bar.
7. Create a text file that contains the starting values of the unobserved variables (i.e., just the parameter θ for this model).[2] The content of the file should look like this:

```
list(
theta = 0.1
)
list(
theta = 0.9
)
```

[1] Running multiple chains is the best and easiest way to ensure WinBUGS uses different random number sequences in sampling. Doing a single-chain analysis multiple times can produce the same results because the random number sequence is identical.

[2] If you do not specify starting values yourself, WinBUGS will create them for you automatically. These automatic starting values are based on the prior and may occasionally result in numerical instability and program crashes. It is therefore safer to assign a starting value for all unobserved variables, and especially for variables at nodes "at the top" of the graphical model, which have no parents.

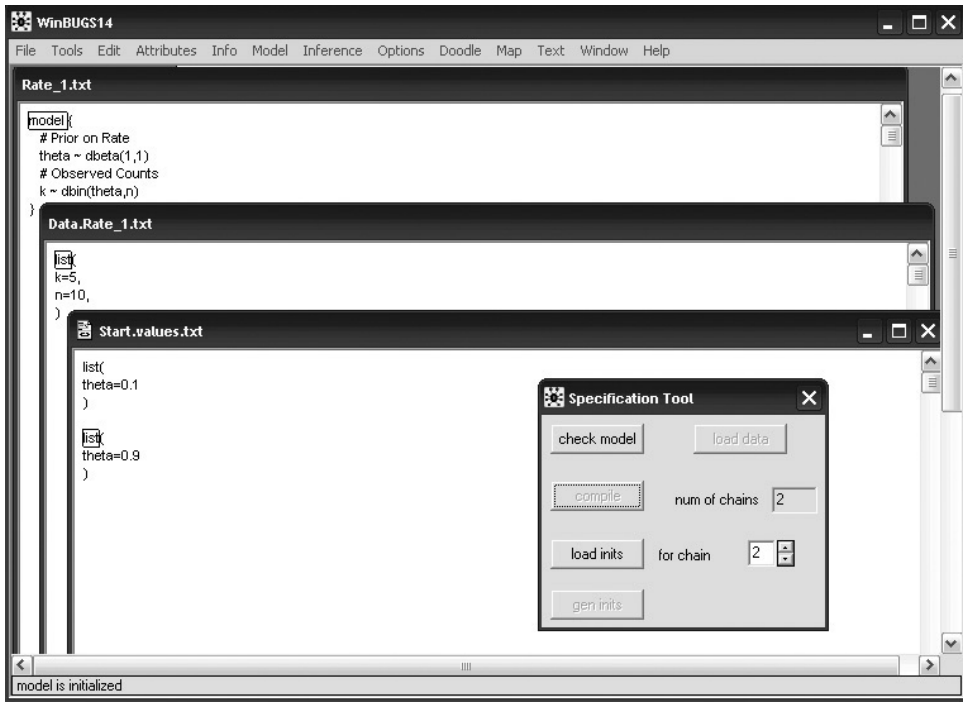

Fig. 2.2 The WinBUGS model specification tool.

Note that there are two initial values, one for each chain. Save the file, for example as `Start.values.txt`.

8. Open the file that contains the starting values by selecting the `Open` option in the `File` menu, selecting the appropriate directory, and double-clicking on the file. To load the starting value of θ for the first chain, highlight the word "list" at the beginning of the file and click on `load inits` in the `Specification Tool` window, shown in Figure 2.2. The status bar will now display the message "chain initialized but other chain(s) contain uninitialized variables." To load the starting value for the second chain, highlight the second "list" command and click on `load inits` once again. If all starting values are successfully loaded, the message "model is initialized" will appear in the status bar.
9. Set monitors to store the sampled values of the parameters of interest. To set a monitor for θ, select the `Samples` option from the `Inference` menu. Once the `Sample Monitor Tool` window, shown in Figure 2.3, is opened, type "theta" in the field labeled `node` and click on `set`.
10. Specify the number of samples you want to record. To do this, you first have to specify the total number of samples you want to draw from the posterior of θ, and the number of burn-in samples that you want to discard at the beginning of a sampling run. The number of recorded samples equals the total number of samples minus the number of burn-in samples. In our binomial example, we will

not discard any of the samples and will set out to obtain 20,000 samples from the posterior of θ. To specify the number of recorded samples, type "1" in the field labeled `beg` (i.e., WinBUGS will start recording from the first sample) and type "20000" in the field labeled `end` in the `Sample Monitor Tool` window, shown in Figure 2.3.

Fig. 2.3 The WinBUGS sample monitor tool.

11. Set "live" trace plots of the unobserved parameters of interest. WinBUGS allows you to monitor the sampling run in real-time. This can be useful on long sampling runs, for debugging, and for diagnosing whether the chains have converged. To set a "live" trace plot of θ, click on `trace` in the `Sample Monitor Tool` window, shown in Figure 2.3, and wait for an empty plot to appear on the screen. Once WinBUGS starts to sample from the posterior, the trace plot of θ will appear live on the screen.
12. Specify the total number of samples that you want to draw from the posterior. This is done by selecting the `Update` option from the `Model` menu. Once the `Update Tool` window, as in Figure 2.4, is opened, type "20000" in the field labeled `updates`. Typically, the number you enter in the `Update Tool` window will correspond to the number you entered in the `end` field of the `Sample Monitor Tool`.
13. Specify how many samples should be drawn between the recorded samples. You can, for example, specify that only every second drawn sample should be recorded. This ability to "thin" a chain is important when successive samples are not independent but autocorrelated. In our binomial example, we will record every sample that is drawn from the posterior of θ. To specify this, type "1" in the field labeled `thin` in the `Update Tool` window, shown in Figure 2.4, or in the `Sample Monitor Tool` window, shown in Figure 2.3. To record only every 10th sample, the `thin` field needs to be set to 10.
14. Specify the number of samples after which WinBUGS should refresh its display. To this end, type "100" in the field labeled `refresh` in the `Update Tool` window, shown in Figure 2.4.
15. Sample from the posterior. To sample from the posterior of θ, click on `update` in the `Update Tool` window, shown in Figure 2.4. During sampling, the message

"model is updating" will appear in the status bar. Once the sampling is finished, the message "updates took x s" will appear in the status bar.

Fig. 2.4 Update Tool.

16. Specify the output format. WinBUGS can produce two types of output; it can open a new window for each new piece of output, or it can paste all output into a single log file. To specify the output format for our binomial example, select `Output options` from the `Options` menu, and click on `log` in the `Output options` window.
17. Obtain summary statistics of the posterior distribution. To request summary statistics based on the sampled values of θ, select the `Samples` option in the `Inference` menu, and click on `stats` in the `Sample Monitor Tool` window, shown in Figure 2.3. WinBUGS will paste a table reporting various summary statistics for θ in the log file.
18. Plot the posterior distribution. To plot the posterior distribution of θ, click on `density` in the `Sample Monitor Tool` window, shown in Figure 2.3. WinBUGS will paste the "kernel density" of the posterior distribution of θ in the log file.[3]

Figure 2.5 shows the log file that contains the results for our binomial example. The first five lines of the log file document the steps taken to specify and initialize the model. The first output item is the `Dynamic trace` plot that allows the θ variable to be monitored during sampling, and is useful for diagnosing whether the chains have reached convergence. In this case, we can be reasonably confident that convergence has been achieved because the two chains, shown in different colors, are overlapping one another.[4] The second output item is the `Node statistics` table that presents the summary statistics for θ. Among other things, the table shows the mean, the standard deviation, and the median of the sampled values of θ. The last output item is the `Kernel density` plot that shows the posterior distribution of θ.

How did WinBUGS produce the results in Figure 2.5? The model specification file implemented the graphical model from Figure 2.1, saying that there is a rate θ with a uniform prior, that generates k successes out of n observations. The data file supplied the observed data, setting $k = 5$ and $n = 10$. WinBUGS then sampled

[3] A kernel density is a fancy smoothed histogram. Here, it is a smoothed histogram for the sampled values of θ.

[4] Note that the `Dynamic trace` plot only shows 200 samples. To have the entire time series of sampled values plotted in the log file, click on `history` in the `Sample Monitor Tool` window.

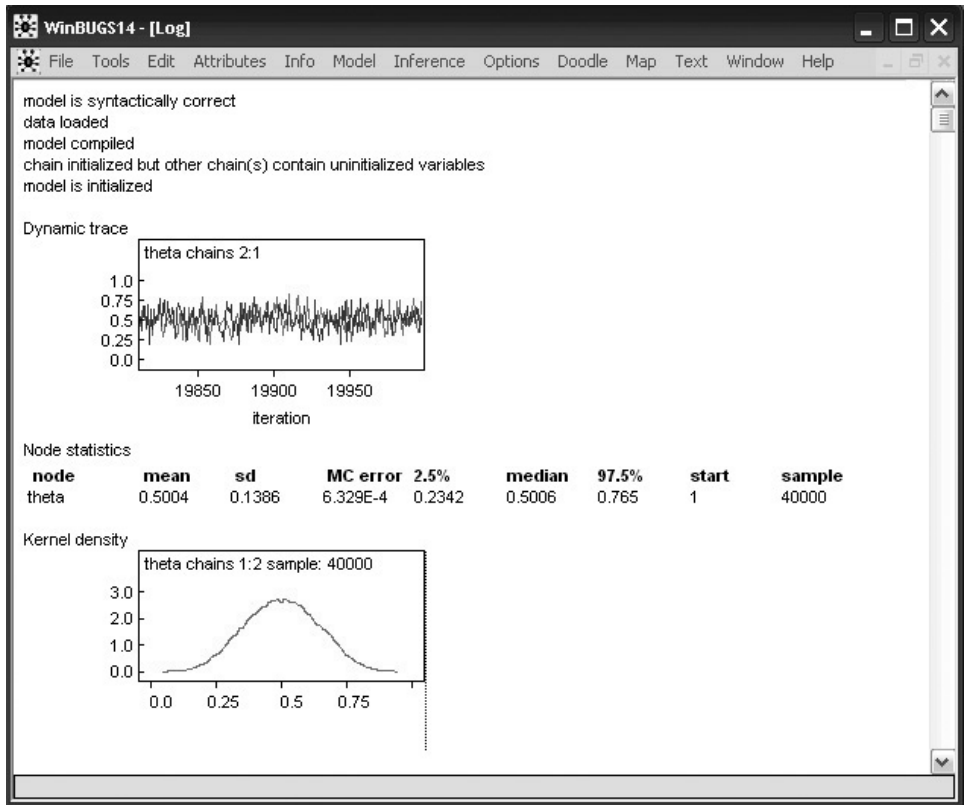

Fig. 2.5 Example of an output log file.

from the posterior of the unobserved variable θ. "Sampling" means drawing a set of values, so that the relative probability that any particular value will be sampled is proportional to the density of the posterior distribution at that value. For this example, the posterior samples for θ are a sequence of numbers like 0.5006, 0.7678, 0.3283, 0.3775, 0.4126, ... A histogram of these values is an approximation to the posterior distribution of θ.

Error messages

If the syntax of your model file is incorrect or the data and starting values are incompatible with your model specification, WinBUGS will balk and produce an error message. Error messages can provide useful information for debugging your WinBUGS code.[5] The error messages are displayed in the bottom left corner of the status bar, in very small letters.

Suppose, for example, that you mistakenly use the "assign" operator (`<-`) to specify the distribution of the prior on the rate parameter θ and the distribution of the observed data `k`:

[5] Although nobody ever accused WinBUGS of being user-friendly in this regard. Many error messages seem to have been written by the same people who did the Dead Sea Scrolls.

Box 2.2 Do I need or want to understand computational sampling?

Some people find the idea that WinBUGS looks after sampling, and that there is no need to understand the computational routines involved in detail, to be a relief. Others find it deeply disturbing. For the disturbed, there are many Bayesian texts that give detailed accounts of Bayesian inference using computational sampling. Start with the summary for cognitive scientists presented in Chapter 7 from Kruschke (2010a). Continue with the tutorial-style overview in Andrieu et al. (2003) or the relevant chapters in the excellent book by MacKay (2003), which is freely available on the Web, and move on to the more technical references such as Gilks et al. (1996), Ntzoufras (2009), and Gamerman and Lopes (2006). You can also browse the internet for more information; for example, there is an instructive applet at `http://www.lbreyer.com/classic.html`, and an excellent YouTube tutorial at `http://www.youtube.com/watch?v=4gNpgSPal_8`.

```
model{
  #Prior Distribution for Rate Theta
  theta <- dbeta(1,1)
  #Observed Counts
  k <- dbin(theta,n)
}
```

As WinBUGS requires you to use the tilde symbol "∼" to denote the distributions of the prior and the data, it will produce the following error message: `unknown type of logical function`, as shown in Figure 2.6. As another example, suppose that you mistype the distribution of the observed counts `k`, and you mistakenly specify the distribution of `k` as follows:

```
k ~ dbon(theta,n)
```

WinBUGS will not recognize `dbon` as an existing probability distribution, and will produce the following error message: `unknown type of probability density`, as shown in Figure 2.7.[6]

With respect to errors in the data file, suppose that your data file contains the following data: `k` = -5 and `n` = 10. Note, however, that `k` is the number of successes in the 10 trials and it is specified to be binomially distributed. WinBUGS therefore expects the value of `k` to lie between 0 and `n` and it will produce the following error message: `value of binomial k must be between zero and order of k`.

[6] On Windows machines, the error message is accompanied by a penetrating "system beep." After experiencing a few such system beeps you will want to turn them off. Browse the web for information on how to do this, or go straight to `http://www.howtogeek.com/howto/windows/turn-off-the-annoying-windows-xp-system-beeps/`. The people sitting next to you will be grateful too.

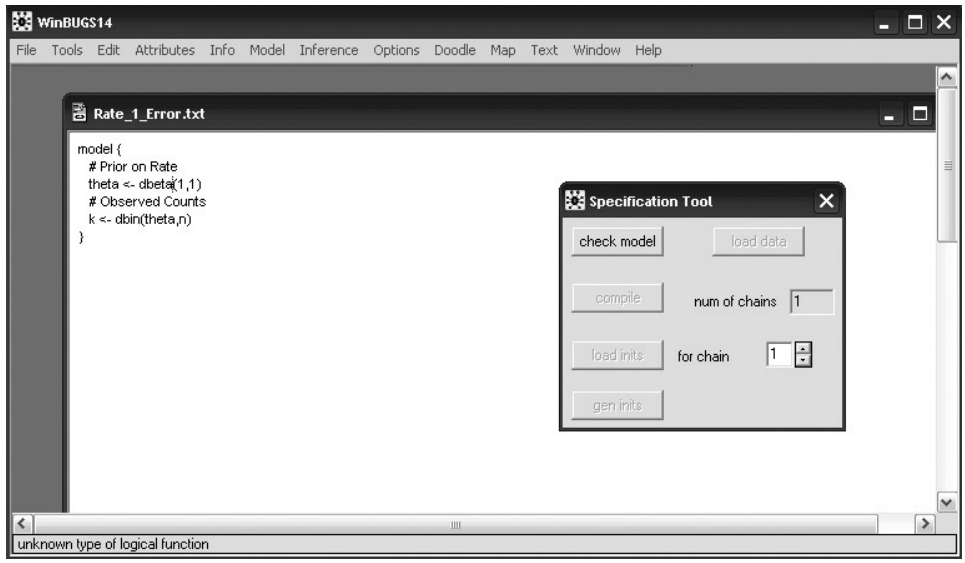

Fig. 2.6 WinBUGS error message as a result of incorrect logical operators. Note the small letters in the bottom left corner of the status bar.

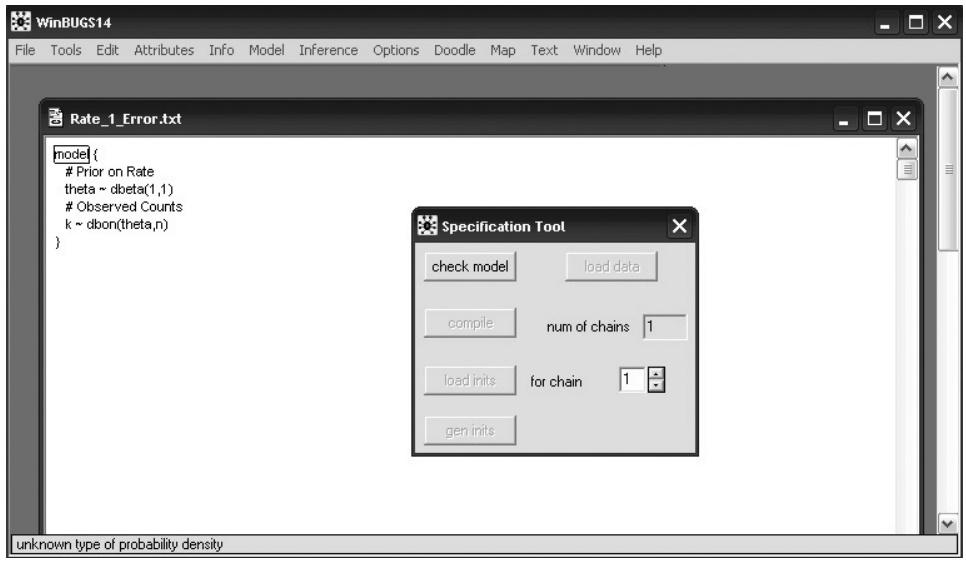

Fig. 2.7 WinBUGS error message as a result of a mis-specified probability density. Note the small letters in the bottom left corner of the status bar.

Finally, with respect to erroneous starting values, suppose that you chose 1.5 as the starting value of θ for the second chain. Because θ is the *probability* of getting 5 successes in 10 trials, WinBUGS expects the starting value for θ to lie between 0 and

Box 2.3 Changing the sampler

WinBUGS uses a suite of samplers, each fine-tuned to a particular class of statistical problems. Occasionally it may be worth the effort to change the default settings and edit the Updater/Rsrc/Methods.odc file. Any such editing should be done with care, and only after you have made a copy of the original Methods.odc file that contains the default settings. The advantage of changing the sampler is that it may circumvent traps or crashes. For example, the WinBUGS manual mentions that problems with the adaptive rejection sampler DFreeARS can sometimes be solved by replacing, for the log concave class, the method UpdaterDFreeARS by UpdaterSlice. Note for Windows 7 users: you may not be able to save any changes to files in the Updater/Rsrc directory. Work-around: copy the file to your desktop, edit it, save it, and copy it back to the Updater/Rsrc directory.

1. Therefore, specifying a value such as 1.5 produces the following error message: `value of proportion of binomial k must be between zero and one`.

2.2.3 Using Matbugs

We will use the `matbugs` function to call the WinBUGS software from within Matlab, and to return the results of the WinBUGS sampling to a Matlab variable for further analysis. The code we are using to do this is shown below:

```
% Data
k = 5;
n = 10;

% WinBUGS Parameters
nchains = 2; % How Many Chains?
nburnin = 0; % How Many Burn in Samples?
nsamples = 2e4;  % How Many Recorded Samples?
nthin = 1; % How Often is a Sample Recorded?

% Assign Matlab Variables to the Observed WinBUGS Nodes
datastruct = struct('k',k,'n',n);

% Initialize Unobserved Variables
start.theta= [0.1 0.9];

for i=1:nchains
    S.theta = start.theta(i); % An Intial Value for the Success Rate
    init0(i) = S;
end

% Use WinBUGS to Sample
[samples, stats] = matbugs(datastruct, ...
    fullfile(pwd, 'Rate_1.txt'),
```

```
'init', init0, 'view', 1, ...
'nChains', nchains, 'nburnin', nburnin, ...
'nsamples', nsamples, 'thin', nthin, ...
'DICstatus', 0, 'refreshrate',100, ...
'monitorParams', {'theta'}, ...
'Bugdir', 'C:/Program Files/WinBUGS14');
```

Some of the options in the Matbugs function control software input and output:

- `datastruct` contains the data that you want to pass from Matlab to WinBUGS.
- `fullfile` gives the name of the text file that contains the WinBUGS scripting of your graphical model (i.e., the model specification file).
- `view` controls the termination of WinBUGS. If `view` is set to 0, WinBUGS is closed automatically at the end of the sampling. If `view` is set to 1, WinBUGS remains open and it pastes the results of the sampling run in a log output file. To be able to inspect the results in WinBUGS, maximize the log output file and scroll up to the top of the page. Note that if you subsequently want WinBUGS to return the results to Matlab, you first have to close WinBUGS.
- `refreshrate` gives the number of samples after which WinBUGS should refresh its display.
- `monitorParams` gives the list of variables that will be monitored and returned to Matlab in the `samples` variable.
- `Bugdir` gives the location of the WinBUGS software.

Other options define the values for the computational sampling parameters:

- `init` gives the starting values for the unobserved variables.
- `nChains` gives the number of chains.
- `nburnin` gives the number of burn-in samples.
- `nsamples` gives the number of recorded samples that will be drawn from the posterior.
- `thin` gives the number of drawn samples between those that are recorded.
- `DICstatus` gives an option to calculate the Deviance Information Criterion (DIC) statistic (Spiegelhalter, Best, Carlin, & Van Der Linde, 2002). The DIC statistic is intended to be used for model selection, but is not universally accepted theoretically among Bayesian statisticians. If `DICstatus` is set to 0, the DIC statistic will not be calculated. If it is set to 1, WinBUGS will calculate the DIC statistic.

How did the WinBUGS script and Matlab work together to produce the posterior samples of θ? The WinBUGS model specification script defined the graphical model from Figure 2.1. The Matlab code supplied the observed data and the starting values for θ, and called WinBUGS. WinBUGS then sampled from the posterior of θ and returned the sampled values in the Matlab variable `samples.theta`. This flow of events is illustrated in Figure 2.9. You can plot the histogram of these sampled values using Matlab, in the way demonstrated in the script `Rate_1.m`. It should look something like the jagged line in Figure 2.8. Because the probability of any value

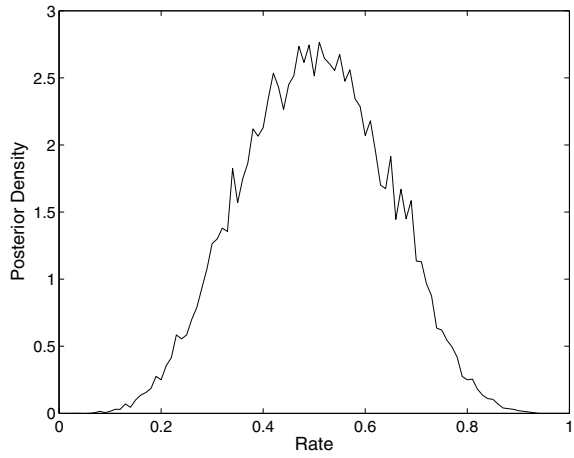

Fig. 2.8 Approximate posterior distribution of rate θ for $k = 5$ successes out of $n = 10$ trials, based on 20,000 posterior samples.

appearing in the sequence of posterior samples is decided by its relative posterior probability, the histogram is an approximation to the posterior distribution of θ.

Besides the sequence of posterior samples, WinBUGS also returns some useful summary statistics to Matlab. The variable `stats.mean` gives the mean of the posterior samples for each unobserved variable, which approximates its posterior expectation. This can often (but not always, as later exercises explore) be a useful point-estimate summary of all the information in the full posterior distribution. Similarly, `stats.std` gives the standard deviation of the posterior samples for each unobserved variable.

Finally, WinBUGS also returns the so-called $\hat{R}$ statistic in the `stats.Rhat` variable. This is a statistic about the sampling procedure itself, not about the posterior distribution. The $\hat{R}$ statistic is proposed by Brooks and Gelman (1998) and it gives information about convergence. The basic idea is to run two or more chains and measure the ratio of within-to-between-chain variance. If this ratio is close to 1, the independent sampling sequences are probably giving the same answer, and there is reason to trust the results.

Exercise

Exercise 2.2.1 Re-read the section on `view`. The Matlab code above specifies `view=1`. What does this do? Change the code to `view=0`. What has changed?

2.2.4 Using R2WinBUGS

We will use the `bugs()` function in the R2WinBUGS package to call the WinBUGS software from within R, and to return the results of the WinBUGS sampling to an

R variable for further analysis. Note for Windows 7 users: in order for the samples to be returned to R successfully, you may need to run R "as administrator" (right-click on the R icon to reveal this option). The R code we are using to obtain the WinBUGS samples is as follows:

```
setwd("D:/WinBUGS_Book/R_codes") #Set working directory, adjust as needed
library(R2WinBUGS) #Load the R2WinBUGS package
bugsdir <- "C:/Program Files/WinBUGS14" #Set WinBUGS directory, adjust as needed

k <- 5
n <- 10

data <- list("k", "n")
myinits <- list(
   list(theta = 0.1), #chain 1 starting value
   list(theta = 0.9)) #chain 2 starting value

parameters <- c("theta")

samples <- bugs(data, inits=myinits, parameters,
               model.file ="Rate_1.txt",
               n.chains=2, n.iter=20000, n.burnin=1, n.thin=1,
               DIC=T, bugs.directory=bugsdir,
               codaPkg=F, debug=F)
```

Note that lines 1 and 3 (i.e., the `setwd` line and the `bugsdir` line) specify the working directory and the WinBUGS directory, but only for the computer that runs the code. If you want to run the code on your own computer you need to modify these lines to match your setup.[7]

Some of the above options control software input and output:

- `data` contains the data that you want to pass from R to WinBUGS.
- `parameters` gives the list of variables that will be monitored and returned to R in the `samples` variable.
- `model.file` gives the name of the text file that contains the WinBUGS scripting of your graphical model (i.e., the model specification file). Avoid using non-alphanumeric characters (e.g., "&" and "*") in the directory and file names. Also, make sure that the name of the directory that contains the model file is not too long, otherwise WinBUGS will generate the following error message: `incompatible copy`. If WinBUGS fails to locate a correctly specified model file, try to include the entire path in the `model.file` argument.
- `bugs.directory` gives the location of the WinBUGS software.
- `codaPkg` controls the content of the variable that is returned from WinBUGS. If `codaPkg=F` (i.e., `codaPkg` is set to FALSE), WinBUGS returns a variable that contains the results of the sampling run. If `codaPkg=T` (i.e., `codaPkg` is set to TRUE), WinBUGS returns a variable that contains the file names of the

[7] When the code does not work immediately, check whether you have changed the directories correctly. The working directory should contain the model file, in this case `Rate_1.txt`, and the `bugsdir` variable should refer to the directory that contains the `WinBUGS14.exe` file.

WinBUGS outputs and the corresponding paths. You can access these output files by means of the R function `read.bugs()`.

- `debug` controls the termination of WinBUGS. If `debug` is set to FALSE, WinBUGS is closed automatically at the end of the sampling. If `debug` is set to TRUE, WinBUGS remains open and it pastes the results of the sampling run in a log output file. To be able to inspect the results in WinBUGS, maximize the log output file and scroll up to the top of the page. Note that if you subsequently want WinBUGS to return the results in the R `samples` variable, you first have to close WinBUGS. In general, you will not be able to use R again until after you terminate WinBUGS.

The other options define the values for the computational sampling parameters:

- `inits` assigns starting values to the unobserved variables. If you want WinBUGS to choose these starting values for you, replace `inits=myinits` in the call to bugs with `inits=NULL`.
- `n.chains` gives the number of chains.
- `n.iter` gives the number of samples that will be drawn from the posterior.
- `n.burnin` gives the number of burn-in samples.
- `n.thin` gives the number of drawn samples between those that are recorded.
- `DIC` gives an option to calculate the Deviance Information Criterion (DIC) statistic (Spiegelhalter et al., 2002). The DIC statistic is intended to be used for model selection, but is not universally accepted theoretically among Bayesian statisticians. If `DIC` is set to FALSE, the DIC statistic will not be calculated. If it is set to TRUE, WinBUGS will calculate the DIC statistic.[8]

WinBUGS returns the sampled values of θ in the R variable `samples`. You can access these values by typing `samples$sims.array` or `samples$sims.list`. The flow of events is illustrated in Figure 2.9.

You can use R to plot the histogram of sampled values of θ, as is demonstrated in the script `Rate_1.R`. In addition to the sequence of posterior samples, WinBUGS also returns to R some useful summary statistics. These summary statistics can be obtained by typing `samples` at the R prompt. When you run two or more chains, the `samples` command also provides the $\hat{R}$ statistic, introduced by Brooks and Gelman (1998). The $\hat{R}$ statistic provides information about the convergence of the sampling procedure, not about the posterior distribution. The basic idea is to run two or more chains and measure the ratio of within-to-between-chain variance. If this ratio is close to 1, the independent sampling sequences are probably giving the same answer, and there is reason to trust the results.

[8] For some reason, setting DIC equal to FALSE can lead to problems in the communication between R and WinBUGS. It is safest to set DIC equal to TRUE, even when you are not interested in the DIC.

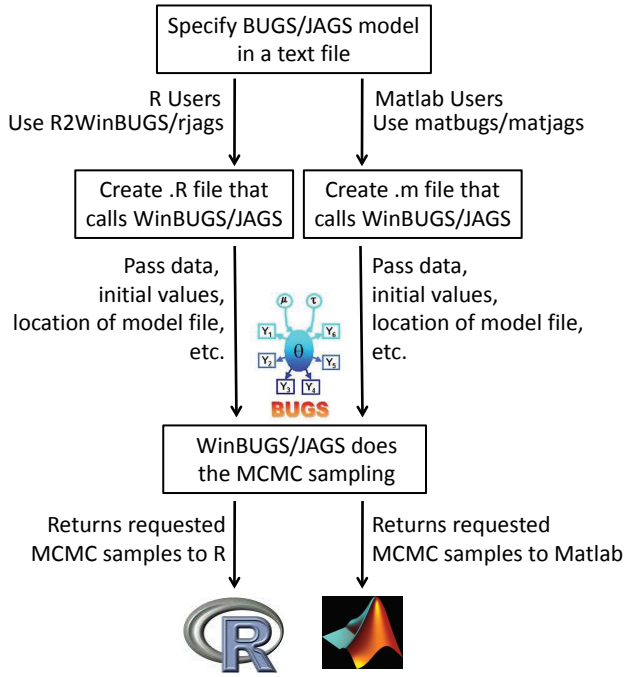

Fig. 2.9 Flowchart that illustrates the interaction between WinBUGS and R (left stream) or Matlab (right stream). JAGS is a program that is very similar to WinBUGS, described in the section on OpenBUGS and JAGS.

Exercise

Exercise 2.2.2 Re-read the section on `debug`. The R code above specifies `debug = F`; what does this do? Change the code to `debug = T`; what has changed?

2.3 Online help, other software, and useful URLs

2.3.1 Online help for WinBUGS

- The BUGS Project webpage `http://www.mrc-bsu.cam.ac.uk/bugs/weblinks/webresource.shtml` provides useful links to various articles, tutorial materials, and lecture notes about Bayesian modeling and the WinBUGS software.
- The BUGS discussion list `https://www.jiscmail.ac.uk/cgi-bin/webadmin?A0=bugs` is an online forum where WinBUGS users can exchange tips, ask questions, and share worked examples.

2.3.2 For Mac users

You can run WinBUGS on Macs using emulators, such as Darwine. As far as we know, you need a Dual Core Intel-based Mac and the latest stable version of Darwine to be able to use R2WinBUGS. Nevertheless, running WinBUGS on a Mac is not ideal. Mac users are encouraged to use JAGS instead. At the time of writing, the information below was useful for running WinBUGS on a Mac:

- The Darwine emulator is available at `www.kronenberg.org/darwine/`.
- The R2WinBUGS reference manual on the R-project webpage `cran.r-project.org/web/packages/R2WinBUGS/index.html` provides instructions on how to run R2winBUGS on Macs.
- Further information for running R2WinBUGS on Macs is available at `ggorjan.blogspot.com/2008/10/runnning-r2winbugs-on-mac.html` and `idiom.ucsd.edu/~rlevy/winbugsonmacosx.pdf`.
- Further information for running WinBUGS on Macs using a Matlab or R interface is available at `http://www.helensteingroever.com` and `www.ruudwetzels.com/macbugs`.

2.3.3 For Linux users

You can run WinBUGS under Linux using emulators, such as Wine and CrossOver.

- The BUGS Project webpage provides useful links to various examples on how to run WinBUGS under Linux `www.mrc-bsu.cam.ac.uk/bugs/faqs/contents.shtml` and how to run WinBUGS using a Matlab interface `www.mrc-bsu.cam.ac.uk/bugs/winbugs/remote14.shtml`.
- The R2WinBUGS reference manual on the R-project webpage `cran.r-project.org/web/packages/R2WinBUGS/index.html` provides instructions on how to run R2WinBUGS under Linux.

2.3.4 OpenBUGS, Stan, and JAGS

This book is designed primarily to work with WinBUGS. There are, however, alternative programs for generating MCMC samples from graphical models. Both OpenBUGS, Stan (Stan Development Team, 2013), and JAGS (Plummer, 2003) may be particularly attractive for Mac and Linux users, since they raise fewer issues than WinBUGS to install and run. The model code for OpenBUGS, Stan, and JAGS is very similar to WinBUGS, so that the transition from one program to the other is generally easy. An effort has been made to make most of the examples in this book compatible with JAGS. Often, in our experience, sampling is much faster in JAGS than it is in WinBUGS.

- OpenBUGS is available from `http://www.openbugs.info/w/`.
- Stan is available from `http://mc-stan.org/`.

- JAGS is available from `http://mcmc-jags.sourceforge.net/`.
- To give R the ability to interact with JAGS, you have to install the `rjags` package, and, optionally, the `R2jags` package. To ensure that you install the latest version of the `rjags` package, the safest procedure is to first Google the terms `rjags CRAN`, go to a website such as `http://cran.r-project.org/web/packages/rjags/index.html`, and—when using Windows—download the package zip file. Then start R, go to the `Packages` menu, choose `Install package(s) from local zip file...`, and select the package zip file you just downloaded. To check whether the installation was successful, type `library(rjags)` at the R prompt. To install the `R2jags` package, you can use the standard installation procedure: start R and select the `Install Package(s)` option in the `Packages` menu. After choosing your preferred CRAN mirror, select `R2jags` in the `Packages` window and click on `OK`.
- To give Matlab the ability to interact with JAGS, download the freely available `matjags.m` function and put it in your Matlab working directory. You can download `matjags.m` directly from `http://psiexp.ss.uci.edu/research/programs_data/jags/`.