APPENDIX 1

A List of WinBUGS TRICKS

The appendix provides a list of tips that hopefully allow you to love WinBUGS more unconditionally. It is based on an appendix in the book *Introduction to WinBUGS for Ecologists* by Marc Kéry (2010), but also includes new stuff. We would suggest you skim over the list when you start working with WinBUGS and then refer back to it later as necessary.

1. Do read the manual: WinBUGS may not have the best documentation available for a software, but its manual is nevertheless very useful. Be sure to at least skim over most of it once when you start getting into WinBUGS, so you may remember that the manual has something to say about a particular topic when you need it. Do not forget the sections entitled “Tricks: Advanced Use of the BUGS Language” and “Tips and Troubleshooting”.
2. Always begin from a template: When starting a new analysis, *always* start from a template of a similar analysis. Only ever try to write an analysis from scratch if you want to test yourself.
3. Make a clear distinction between BUGS and R code: We use the R function `sink()` to write into the R working directory (which you can set yourself using `setwd()`) a text file containing the model description in the BUGS language. We find it practical to have all our codes in a single document. Here is a sketch example of how this looks like (you see this kind of code for each example where we use WinBUGS in the book).

```
# Define the model in the BUGS language
sink("modelname.txt")
cat("
   model {          # BUGS model starts on this line

   # Priors
   some priors
```

```
    # Likelihood
    for (i in 1:n){
        some description of the likelihood
        }
    }                   # BUGS model ends with this line
",fill = TRUE)
sink()
```

Here, R code is left-aligned and BUGS code is indented one level and in bold. We neither show this indentation nor the bold type in the book, but show it here to clarify how everything inside the two quotes after "`cat`" and before "`,fill`" is BUGS code and everything outside these two quotes is R code. You have to be totally clear about which part of the code is in the BUGS language and which is in the R language. This may be a little confusing at first, especially because the two languages are quite similar: R is a dialect of S and BUGS is strongly inspired by S. Moreover, this way of writing the BUGS model sometimes seems to cause problems when using the R editor Tinn-R (see trick 17).

4. Give initial values: The wise choice of initial values can be the key to success or failure of an analysis in WinBUGS. With complex models, WinBUGS needs to start the Markov chains not too far away from their stationary distribution or it will crash or not even start to update. Of course, the requirement to start the chains close to the solution goes counter the requirement to start them at dispersed places in order to assess convergence, so some reasonable intermediate choice is important.

5. Do not select initial values that contradict the priors: Initial values must not be outside of the possible range of a parameter. For instance, negative initial values for a parameter that has prior mass only for positive values (such as a variance) will cause WinBUGS to crash and so do initial values outside of the range of a uniform prior.

6. Only provide initial values for quantities that appear in the model: Otherwise the "incompatible copy" error may appear.

7. Do not provide initial values for fixed elements of a vector-valued parameter: Sometimes some elements of a vector-valued parameter are known or fixed at a certain value. Then, they are no longer a parameter that is estimated and initial values must not be given for them. An example is a two-way fixed-effects ANOVA, where you must set to zero the effect of one level (e.g., the first) of one factor to avoid overparameterisation. In this case, no initial value is required for that effect. For a factor `beta` with four levels, the first of which is set to zero, you can do this as follows: `beta = c(NA, rnorm(3))`.

8. In your prior choice, be ignorant, but not too ignorant: When you want your Bayesian inference to be dominated by your data and choose priors intended to be vague, do not specify too much ignorance, otherwise traps may result or convergence may not be achieved. For instance, do not specify the limits of a uniform prior or the variance of a normal prior to be too wide.

9. How to deal with missing values (NAs): In WinBUGS, NAs are dealt with less automatically than in conventional stats programs with which you are likely familiar; hence, it is important to know how to deal with them: briefly, missing responses (i.e., missing y s) are not a problem, but NAs in the explanatory variables (the x s) need attention. A missing response is simply estimated, and indeed, adding missing responses for selected covariate values is one of the simplest ways to form predictions for desired values of explanatory variables (see Sections 5.4 and 11.5). On the other hand, a missing explanatory variable must either be replaced with some number, for example, the mean observed value for that variable, or else given some prior distribution. In general, the former is easier and should not pose a problem unless the number of missing x s is large.

10. NAs and NaNs: When dealing with data in multidimensional arrays, a very useful R package is "reshape". The newer versions of the reshape package in R 2.9 use an NaN to fill in NAs. This makes WinBUGS very unhappy—you must have NA, not NaN. In general, this is probably good to know about BUGS, and newer versions of other packages may be doing the same thing. So, if you use the `melt/cast` functions in reshape to organize data, then you will need to update your code in the newer R versions by adding `"fill=NA_real_"`. Example: `Ymat=cast(data.melt, SppCode~JulianDate~GridCellID, fun.aggregate=mean, fill=NA_real_)` (Beth Gardner, personal communication).

11. Data in arrays; think in a box (and know your box): When coding an analysis in WinBUGS, you often will have to deal with data that come in arrays, and these may have more than one dimension. For instance, when analyzing animal counts from different sites, over several years and taken at various months in a year, it may be useful to format them into a three-dimensional array. Some covariates of such an analysis will then have two or even three dimensions, too. You must then be absolutely clear about the dimensions of theses "boxes" in which your data are and not get confused by the indexing of the data. In our experience, knowing how to format data into such arrays and then not getting lost is one of the most difficult things to learn about the routine use of WinBUGS.

12. Loop order in arrays: In "serious" analyses, your modeling will often require the data to be formatted in some multidimensional array. For instance, for a multispecies version of a site-occupancy model (Dorazio and Royle, 2005), you will have at least three dimensions corresponding to species (*i*), site (*j*), and replicate survey (*k*). It appears that how you build your array and, especially, how you loop over that array in the definition of the likelihood can make a huge difference in terms of the speed with which your Markov chains in WinBUGS evolve. You should loop over the longest dimension first and over the shortest last. For instance, if you have data from 450 sites, 100 species, and for two surveys each, then it appears best to format the data as *y*[*j*, *i*, *k*] and then loop over sites (*j*) first, then over species (*i*), and finally over replicate surveys (*k*) (Beth Gardner and Elise Zipkin, personal communication).

13. Do not define things twice: Every parameter in WinBUGS can only be defined once. For instance, writing `y ~ dnorm(mu, tau)` and then adding `y[3] <- 5` will cause an error. There is a single exception to this rule, and that is the transformation of the response by some function such as the `log()` or `abs()`. So in order to conduct an analysis of a log-transformed response, you may write `log.y <- log(y)` and then `log.y ~ dnorm(mu, tau)`. Beware of inadvertently defining quantities multiple times when erroneously putting them within a loop that they do not belong.

14. Problems with WinBUGS' own logit function: We have sometimes experienced problems when using WinBUGS' own logit function, for instance, with achieving convergence. Therefore, it is often better to specify that transformation explicitly by `logit.p[i] <- log(p[i] / (1 – p[i]))`, `p[i] <- exp(logit.p[i]) / (1 + exp(logit.p[i]))` or `p[i] <- 1 / (1 + exp(-logit.p[i]))`.

15. "Stabilizing" the logit: To avoid numerical over- or underflow, you may "stabilize" the logit function by excluding extreme values (Brendan Wintle, personal communication). Here's a sketch of how to do that. The Gibbs sampling will typically get slower, but at least WinBUGS will be less likely to crash:

```
logit(psi.lim[i]) <- lpsi.lim[i]
lpsi.lim[i] <- min(999, max(-999, lpsi[i]))
lpsi[i] <- alpha.occ + beta.occ * something[i]
```

16. Truncated priors for normal random effects: Similarly, in log- or logit-normal mixtures (which we see when introducing a normal random effect into the linear predictor), you can truncate the zero-mean normal distribution, e.g., at ± 20 (Kéry and Royle, 2009): `e[i] ~ dnorm(0, tau) I(–20,20)`). This can greatly help convergence of the Markov chains.

17. Problems with Tinn-R: Users of the popular R editor Tinn-R 2.0 (or newer) may have problems writing the text file containing the BUGS model description with the `sink()` function; Tinn-R adds to that file some gibberish that will cause WinBUGS to crash. You must then use an alternative way of writing the model file. As an example, here is a workaround that should be compatible with Tinn-R (Wesley Hochachka, personal communication):

```
modelFilename = 'model.txt'
cat("
model {
# Here is the model in BUGS language
}
",fill=TRUE, file=modelFilename)
```

An alternative solution due to Jérôme Guélat is this: The "R send" functions available in Tinn-R allow sending commands into R. However, the "(echo=TRUE)" versions of these functions should not be used when sending the `sink()` function and its content into R. For example: one should use "R send: selection" instead of "R send: selection (echo=TRUE)".

18. Run trial analyses first: Run very short chains first, for example, of length 12 with a burnin of 2, just to confirm that there are no coding or other errors. Only when you are satisfied that your code works and your model does what it should, increase the chain length to get a production run.

19. How to choose the burnin length: We have chosen Markov chain lengths so that convergence appears to be achieved. You may ask yourself how we decided on adequate chain lengths. The answer is simple: we always conduct trial runs first and based on that decide on the chain length for a production run of the analysis.

20. Avoid long Windows addresses: WinBUGS does not seem to like very long Windows addresses (C:\My harddisk\Important stuff\Less important stuff\ ...) for its working directory. Hence, you should not bury your WinBUGS analyses too far down in a tree hierarchy.

21. Use of native WinBUGS (1): A feature of both Kéry (2010) and this book is that WinBUGS is run exclusively from within program R. We believe that this is much more efficient than running native WinBUGS. However, with some complex models and/or large data sets, WinBUGS will be extremely slow. This may be the one exception where it is perhaps more efficient to run WinBUGS natively. You may still prepare the analysis in R as shown in this book, but only request WinBUGS to run very short Markov chains. When you set the option DEBUG = TRUE in the function `bugs()`, then WinBUGS will stay open after the requested number of iterations have been conducted. Then, you can request more iterations to be executed directly in WinBUGS (i.e., using the Update Tool; see chapter 4 in Kéry, 2010). You can then incrementally increase the total chain length and monitor convergence as you go. Once convergence has been achieved, do the required additional number of iterations and save them into coda files. You must do this latter, since when exiting WinBUGS, the `bugs()` function will only import back into R the (small) number of iterations that you originally requested. When you have your valuable samples of your complex model's posterior distribution in coda files, use facilities provided by R packages `boa` or `coda` to import them into R and process them (e.g., compute Brooks–Gelman–Rubin convergence tests or posterior summaries for inference about the parameters).

22. Use of native WinBUGS (2): Use of native WinBUGS can also be helpful to diagnose why a model does not run properly or produces unexpected results. If WinBUGS has been successfully called from R, there will be three (or more) text files in your current directory. If you have specified `working.directory = getwd()` in "bugs", the files will be stored in the working directory (type `getwd()` if you are not sure which one this is). Otherwise, the files will be in a temporary directory (type `tempdir()` to see the path). The file stored first is the BUGS model and has the name that you have specified just after the `sink` command. The second file with the name `data.txt` contains the data in WinBUGS format. Finally, the third file contains the initial values in WinBUGS format and is named `inits1.txt`. If you have specified more than one chain, there will be more files of this type. To make use of native WinBUGS, you open WinBUGS, and select "file" –> "new". Then you copy the model text file, the data text file, and the text file(s) with the initial values into the empty window. You have then all information to start with a native WinBUGS analysis. See the WinBUGS documentation and chapter 4 in Kéry (2010) if you do not know how to proceed within WinBUGS.

23. Be flexible in your modeling: Try out different priors, for example, for parameters representing probabilities try a uniform(0,1), a flat normal for the logit transform, or a beta(1,1). Sometimes, one may work while another does not. Similarly, WinBUGS is very sensitive to changes in the parameterization of a model (see Gelman and Hill, 2007, for some good examples). Sometimes, one way of writing the model may work and the other does not, or one works much faster than the other.

24. Don't know how to specify the linear model? If you have trouble seeing how to specify a linear model in WinBUGS, use of the very handy R function `model.matrix()` may help. For instance, if you want to fit a model with four factors, A, B, C, and D, with all main effects and interactions A.B and C.D, do this to see how the linear model looks like: `model.matrix(~ A + B + C + D + A:B + C:D)`.

25. Scale continuous covariates: Scaling continuous covariates so that their range does not extend too far away on either side of zero, can greatly improve mixing of the chains and often makes convergence possible (see, e.g., Section 11.4. in Kéry, 2010, and Section 3.3.1 in this book).

26. What if WinBUGS hangs after compiling? Try a restart and if that does not work, find better starting values (this tip is from the manual).

27. How to debug a WinBUGS analysis (1): If something went wrong, you need to attentively read through the entire WinBUGS log file from the top to identify the first thing that went wrong. Other errors may follow, but they may not be the actual cause of the failure.

28. How to debug a WinBUGS analysis (2): When something does not work, the simplest and best advice (see also Gelman and Hill, 2007) is to go back to a simpler version of the same model, or to a similar model, that did work, and then incrementally increase the complexity of that model until you arrive at the desired model. That is, from less complex models *sneak up* on the model you want. Indeed, when using WinBUGS, you learn to always start from the simplest version of a problem and gradually build in more complexity until you are at the level of complexity that you require. We think that this is actually a very good approach to learning in general.

29. DIC problems: Surprisingly, sometimes when getting a trap (including one with the very informative title "NIL dereference (read)"), setting the argument `DIC = FALSE` in the `bugs()` function has helped.

30. What if R chokes on too much results from WinBUGS? Sometimes the R object created by R2WinBUGS is too big for one's computer. Then, use `boa` or `coda` to read in the coda files directly and use their facilities to produce your inference in this way (e.g., convergence diagnostics, posterior summaries). Also see trick 37.

31. Check of identifiability/estimability of parameters: To see whether two or more parameters are difficult to estimate separately, you can plot the values of their Markov chains against each other.

32. Check of model adequacy: Do residual analysis, posterior predictive checks, and cross validation to see whether your model appears to be an adequate representation of the main features in the data.

33. Predictions: The estimation of unobserved or future data is a very important part of inference. One particularly useful way to examine predictions is to estimate what a response would look like for a chosen combination of values of the explanatory variables. The generation and examination of such predicted values is an important method to understand complex models (for instance, to see what a particular interaction means) and also needed to illustrate the results of an analysis, for example, as a figure in a paper.

34. Sensitivity analysis for priors: Consider assessing prior sensitivity, that is, repeat your analyses, or those for key models, with different prior specifications and see whether your inference is robust in this respect.

If it is not, then not all is lost, but you must report on that in the methods section of your paper.

35. VISTA problems: Windows VISTA has caused all sorts of "challenges" in workshops taught—be prepared! One problem was that the default BUGS directory is not the same as that stated in the preface.

36. Windows 7 problems: We experienced problems when WinBUGS is installed in a folder like "C:\Program Files\...". WinBUGS runs fine when directly installed on "C:\".

37. Use of the coda package: Some prefer to work with coda objects than with the results returned by `bugs()` as we do throughout the book. Here is some R sample code to summarize the posterior samples and to check convergence and sample autocorrelation (from Richard B. Chandler):

```
outmc <- as.mcmc.list(out)
summary(outmc)
plot(outmc, ask=TRUE)
outmc[,"alpha"]
autocorr.plot(outmc, ask=TRUE)
gelman.plot(outmc, ask=TRUE)
gelman.diag(outmc)
window(outmc, thin=5)
```

38. Free choice among the sisters: When a model does not run in WinBUGS, you may try (and succeed) in OpenBUGS or JAGS, or vice versa (Mike Meredith, personal communication).

39. Beware of the dreaded "tiefschutz and probefahrt error" (Scott Sillett, Andy Royle, personal communication): It means that your computer has been invaded by space aliens. Stay calm, shut the door, leave the building, and set it on fire.

40. Last but not least, you must have a healthy distrust in your solutions: Always inspect your inference to see whether the WinBUGS solution makes sense with respect to what you know about the modeled system. For instance, look at tables of estimates and plot predictions against observed values for quantities that can be observed. Also watch out for unexplained differences in parameter estimates between neighboring models, for example, those that differ by only one covariate or some other rather minor model characteristic. This can be an indication that something went wrong (e.g. convergence was not reached or you made a coding error) or that there are estimability problems with the model for your data set.