An Introduction to **iBUGS**

Yihui Xie *

November 11, 2012

iBUGS is an R package which aims to make it easier to call WinBUGS and OpenBUGS in R. The computation is done by **R2WinBUGS** (Sturtz *et al.*, 2005) and **BRugs** (Thomas *et al.*, 2006), and the GUI is created by **gWidgetsRGtk2** (Lawrence and Verzani, 2010). To start the GUI, simply type `library(iBUGS)` in R. In case you close the window carelessly, you may start it again by `iBUGS()`.

```
## the GUI will show up once the package is loaded
library(iBUGS)
## or call iBUGS() to generate another GUI
iBUGS()
```

1 Motivation

I was thinking about writing this package when I saw the default value for the argument `bugs.directory` was "`c:/Program Files/WinBUGS14/`" in the main function *bugs()* of **R2WinBUGS**, as I believe it is not a too difficult task to use R to find the installation directories of WinBUGS and OpenBUGS for most Windows users. I tried the Windows registry approach[1] but found it was not general enough, so I switched to another "brute-force" way: searching for WinBUGS and OpenBUGS in the directory defined by the environment variable "`ProgramFiles`". Most Windows users will install programs in this directory, so we no longer need to manually specify "`bugs.directory = c:/Program Files/WinBUGS14/`".

Based on this trivial motivation, I think we can also make other arguments easier to specify. For instance, we can analyze the BUGS model code and guess the parameter names using regular expressions; then put them in a list and select the ones we are interested in. In this case, we do not need to type the names of parameters one by one, which will otherwise be tedious in R or WinBUGS/OpenBUGS. When I was new to WinBUGS[2], I often forgot to specify the parameters (node) to monitor in the "Sampler Monitor Tool" panel, and the consequence was I got nothing after a long long waiting and began to regret clicking the "Update" menu too fast.

2 The GUI

The main interface of **iBUGS** looks quite simple as shown in Figure 1. The button "Open" and "Save" can be used to open and save a BUGS model respectively. There will be a tooltip hanging around the text box and showing the path of the current file when you move the mouse over it (the tooltip will be empty if you didn't open a model and have not saved the current one). The text box is the place to write the model; it comes with a default sketch of the model.

When you finished writing the model, you need to make sure the data objects you mentioned in the model are already in the current R session. Then open the "Preferences" panel (Figure 2): all the arguments

*Department of Statistics, Iowa State University. Email: xie@yihui.name (or xie@iastate.edu). Homepage: `http://yihui.name`

[1] `http://yihui.name/en/2010/03/looking-for-software-paths-in-windows-registry/`

[2] In fact, I'm still new to WinBUGS, because now I mainly use R and **iBUGS**...

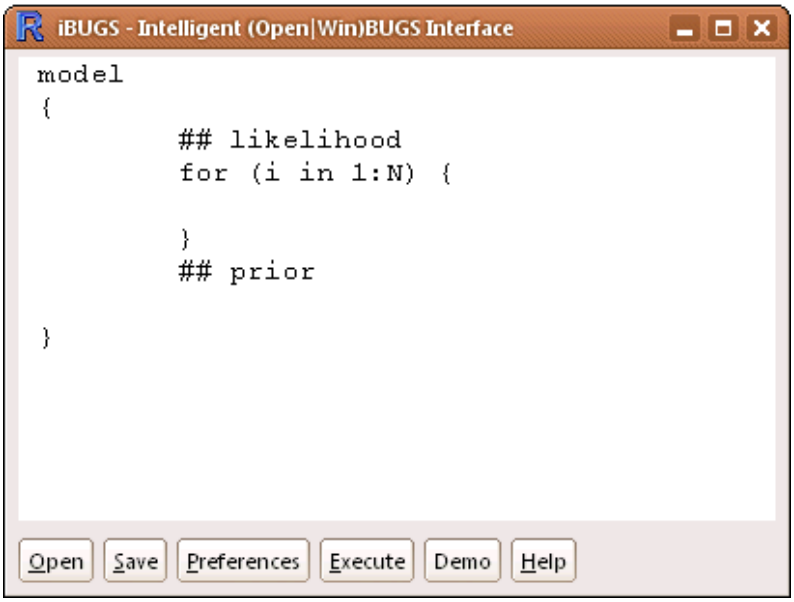

Figure 1: The main interface of **iBUGS**

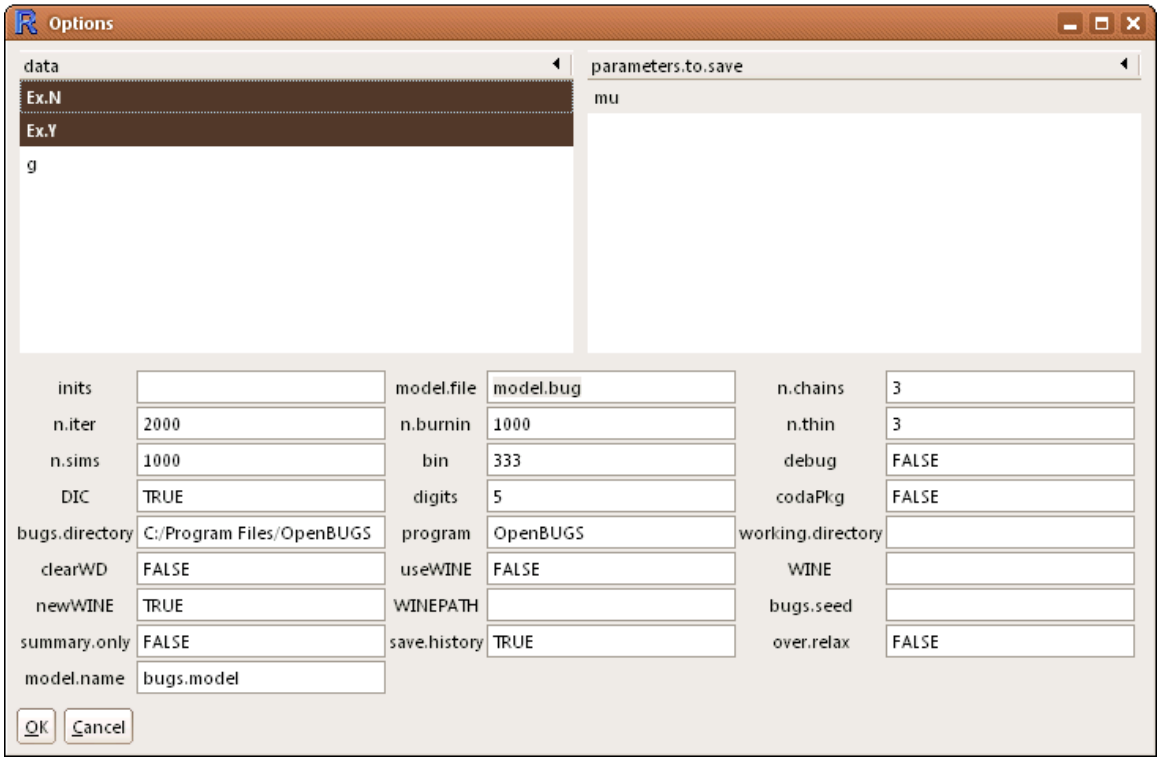

Figure 2: The preference panel

for the function *bugs()* are listed there. If you are familiar with the function *bugs()*, I will not need to explain anything here. The `data` list is read from R's workspace; note it even includes the names from the objects that are *attach()*ed to the R session. For example, the code below can make the two objects `x` and `y` in `dat` visible to **iBUGS**:

```
dat = list(x = 1:3, y = rnorm(5))
## personally I don't recommend attaching R objects that is a
## bad habit
attach(dat)
```

The parameter names, as introduced before, can be found out automatically from the BUGS model. In case it failed to identify certain parameter names, you can also double click on the list box and manually add the names to the list. Other options are from the parameters of the *bugs()* function.

Note that in the preference panel, there is an additional option "`model.name`" (default to be "`bugs.model`"); the results will be saved to an R object with this name, so that you can do further analysis with this object.

Click "Execute" to run the model. There is also a simple demo and you can test if **iBUGS** works for you.

3 Technical Details

3.1 WinBUGS / OpenBUGS directory

Most Windows system comes with an environment variable "`ProgramFiles`", which records the default directory to install new software packages. We can search in this directory for WinBUGS or OpenBUGS. Using file manipulation functions such as *list.files()* as well as regular expressions, we can test if any BUGS package has been installed; e.g.

```
if (nzchar(prog <- Sys.getenv("ProgramFiles")) && length(bugs.dir <- list.files(prog,
    "^(Open|Win)BUGS.*")) && length(bugs.exe <- dirname(list.files(file.path(prog,
    bugs.dir), pattern = "(Open|Win)BUGS.*\\.exe$", full.names = TRUE,
    recursive = TRUE)))) {
    ## if I can find OpenBUGS, use it prior to WinBUGS
    program = ifelse(length(grep("OpenBUGS", bugs.exe)), "OpenBUGS",
        "WinBUGS")
    ## ignore multiple directories if (several versions of) BUGS
    ## installed in multiple places
    bugs.directory = bugs.exe[grep(program, bugs.exe)][1]
}
program
bugs.directory
```

Note if both WinBUGS and OpenBUGS are detected, **iBUGS** will prefer OpenBUGS, as we know the development of WinBUGS has stopped.

3.2 GUI construction

Building a GUI in R with the **gWidgets** (Verzani, 2010) package is quite easy. What's more, it is even *dynamic*! That means you can generate GUI components dynamically whenever you need them. The **gWidgets** package comes with 4 types of interface, namely GTK+ (**gWidgetsRGtk2**), tcltk (**gWidgetstcltk**), Java (**gWidgetsrJava**) and WWW (**gWidgetsWWW**). You don't need to deal with these four specific packages – just play with **gWidgets** and specify your GUI type. Personally I like the GTK+ interface most, so I made it the default one for **iBUGS**. Here is a short demo:

```
library(gWidgets)
options(guiToolkit = "RGtk2")
## create a window and add a button to it
gw = gwindow("GUI Demo")
gb = gbutton("Click me!", container = gw, handler = function(h,
    ...) {
    svalue(h$obj) = paste(svalue(h$obj), "haha!")
})
```

In **gWidgets**, there are several widgets available (text boxes, buttons, drop-down list, ...), and you just need to think about how to arrange them. You can attach an event (handler) to a widget so that this event will be called when users take an action to the widget.

4 Future Work

A list of things in my mind:

1. Dynamically read the help page of *bugs()* and add the explanations of all arguments to the widgets, so users do not need to turn to `?bugs` each time they use it;
2. Add support for Linux – possibly with JAGS[3] (R packages like **rjags**, **R2jags** or **runjags**); I know little about Mac, but I guess with proper Wine emulation specifications, iBUGS can also work (but not so intelligent); I'll appreciate help from Mac users;
3. I have not made up my mind yet: is it worth providing a menu for diagnostics? (e.g. Plot ▷ Gelman-Rubin-Brooks) Sounds like I'm re-inventing the wheel...
4. Intelligent enough to automatically judge convergence and adaptively select parameters?...

Acknowledgment

I'd like to thank Prof Di Cook for her support and ideas, Ted Peterson for the IT support, and Dr Alyson Wilson for introducing me to the Bayesian world.

References

Lawrence M, Verzani J (2010). *gWidgetsRGtk2: Toolkit implementation of gWidgets for RGtk2*. R package version 0.0-64, URL `http://CRAN.R-project.org/package=gWidgetsRGtk2`.

Sturtz S, Ligges U, Gelman A (2005). "R2WinBUGS: A Package for Running WinBUGS from R." *Journal of Statistical Software*, **12**(3), 1–16. URL `http://www.jstatsoft.org`.

Thomas A, O'Hara B, Ligges U, Sturtz S (2006). "Making BUGS Open." *R News*, **6**(1), 12–17. URL `http://cran.r-project.org/doc/Rnews/`.

Verzani J (2010). *gWidgets: gWidgets API for building toolkit-independent, interactive GUIs*. R package version 0.0-40, URL `http://CRAN.R-project.org/package=gWidgets`.

[3]Just Another Gibbs Sampler: `http://www-fis.iarc.fr/~martyn/software/jags/`