Introduction to R and BUGS

Richard F MacLehose, PhD
National Institute of Environmental Health Sciences

June 18, 2007

1 What is R

R is the freeware version of S-Plus. It is an exceptional statistical computing environment and offers tremendous graphical flexibility (see http://addictedtor.free.fr/graphiques/ if you don't believe me). User-written packages that can be downloaded from the web make R very flexible and it will cover most of the needs of most epidemiologists. In fact, one of the packages is *epitools* which includes functions for 2x2 tables, standardization, kaplan-meier curves, etc.

2 What is BUGS

BUGS (**B**ayesian inference **U**sing **G**ibbs **S**ampling) is free software that allows users to specify fairly complex hierarchical models and then figures out a Markov Chain Monte Carlo (MCMC) algorithm to estimate the model. BUGS allows you to specify your outcome model (the likelihood) and your priors, which are relatively easy tasks, and then it does the hard part: running an MCMC algorithm. WinBUGS is the Windows version of BUGS and openBUGS is the new open source version of BUGS. Other BUGS programs exist: geoBUGS, for spatial data, and pkBUGS for pharmacokinetic data. The examples from our shortcourse use winBUGS and openBUGS. You should get the same answer from either software package, although they may use different flavors of MCMC methods to get to that answer. OpenBUGS is nice to use in conjunction with R simply because it runs completely in the background, whereas when R calls winBUGS it has to open a new window.

2.1 Getting BUGS and R

Follow the directions on Andrew Gelman's website http://www.stat.columbia.edu/ gelman/bugsR/ to install R and BUGS, as well as the R packages that call BUGS from R. The directions are very explicit and easy to follow.

The directions at his website are for his book with Jennifer Hill titled "Data Analysis using Regression and Multilevel/Hierarchical Models", which is an excellent resource for anyone who plans on doing any of this type of analysis.

Download the file "Rprofile.site" from our website and copy it to the directory "etc" where R was saved, probably C:\Program Files\R\R-2.5.0\etc. You'll save this over an existing file of the same name. Rprofile.site tells R what packages to load when it starts up and where your working directory is. As a default, I'm setting c:\r as the working directory, so you should create that file on your harddrive and save the R programs and datasets on our website there.

3 Getting Started in R

It really is worth becoming proficient in R, as it offers virtually every statistical and graphical technique you could ever want. As with any software program, the easiest way to learn it is to use it for a couple of analyses. You'll pick it up quickly. Online tutorials are everywhere. Here are a few good ones

- http://www.math.csi.cuny.edu/Statistics/R/simpleR/index.html
- http://cran.r-project.org/doc/contrib/usingR.pdf
- cran.r-project.org/doc/manuals/R-intro.pdf
- http://cran.r-project.org/doc/contrib/refcard.pdf

One of the things you'll find strange about R at first is that while you *can* interact with it on a line-by-line basis (sort of like Stata's command line), it's much easier to write a program and then run the program (as in SAS). However, unlike SAS, R doesn't have a built in text editor where you can write and run the program. You can use your computers standard text editor, like winpad, but its better to download (free) programs that have been adapted to suit R programming. I use WinEdt (www.winedt.com), and it works quite well. You'll want to type

install.packages("RWinEdt")

at the R command prompt (>) so that WinEdt can interact with R. Go ahead and install the epitools package while you're at it

install.packages("epitools").

Now you've got everything you need to get up and running. As I said, the easiest way to learn R is to use it. In this tutorial, I'll walk you through the analysis of the county-specific breast cancer rates in North Carolina in 2004. Download "counties.txt" and "counties.R" and save them in c:\r.

When you start R, WinEdt will start up automatically (because we told it to in Rprofile.site). In WinEdt, open the files counties.txt and counties.bug. I usually find if I'm working on a desktop or a laptop with a large enough screen that its easiest to have R open on the left half of the screen and WinEdt on the right half of the screen. Even on my laptop, this work pretty well. The file counties.txt that we'll be using is well annotated so you should be able to follow along with what each line of code does.

In WinEdt, there are a few ways to interact with R. The two that I generally choose are to 1) put my curser on the line of code in WinEdt that I want to run in R and press alt-l 2)highlight a block of code that I want to run and press alt-p. We'll go line by line right now. First, notice a couple of R-peculiarities: the pound sign (#) is used for comments. R will ignore anything after a #. Second, R uses ← like other programs use =. This is the more traditional usage, however a few versions ago R changed and now lets you use = sign too. I generally use the arrow notation because thats how i learned it. Feel free not to adopt this notation if you find it confusing. But if you type **a ← 2** or **a=2**, in either case a is set equal to 2.

Lets load the county data. Put your curser on the line of code that says

county ← read.table('c:/r/counties.txt',header=TRUE)

and press alt-l. This will automatically run the line of code and bring the R window up so you can see what's happening. We've read in the dataset counties.txt and saved in the matrix county. If you type

county, you'll see the dataset. If you type **county[1,]** you'll see the first row of the dataset. If you type **county[,1]** you'll see the first column of the dataset. If you type **county[1:10,1]**, you'll see the first 10 rows of the first column of the dataset, etc.

Now we generate the exact MLE results by placing the cursor on the line of code that says

mle.exact←pois.exact(county$cases,county$pop)

and pressing alt-l. This line of code calls the function pios.exact and stores the results in mle.exact. The dollar signs tell R that you want the column of the dataset that has the heading cases or pop. If you type **mle.exact** you'll see the results: the MLE of the rate for each county with upper and lower confidence intervals.

We have our frequentist results, now lets get the Bayes results. First, take a look at the file counties.bug (open it in WinEdt). You'll notice that the way this file looks is pretty similar to how I wrote things in my slides. It starts with a model statement which has to be there: it tells BUGS everything between the curly brackets is the model. The next line is a for loop. This is a little different from what you're probably used to in most software packages. When defining models into BUGS, its typically easiest to write out the model for each of the observations in your dataset. In this case we have 100 observations (100 counties) so we have a loop that goes from 1 to 100. BUGS understands that everything in the FOR loop's curly brackets is repeated for each county. The first line in the for loop says that the number of BrCa cases in each county follows a poisson distribution with parameter mu[i]. We define mu[i] in the next line as the product of the size of the population in county i, N[i], and the rate of BrCa in that county, lambda[i]. We want to place a prior distribution on the log of lambda[i] (this is convenient to do, since we can place a Normally distributed prior on log(lambda[i]) and not have to worry about negative values, since exponentiating a negative number leads to a positive number which a rate must be) so we define theta[i] as the log of lambda[i] in the next line of code. Finally, we place a Normal prior on theta[i] in the last line. That's it. Four lines of code (excluding the brackets, model statement and for statement).

Now that we know what our BUGS program says, we just need to call it from R. Doing this is easy. You define your data set, initialize parameters, then call BUGS. First, we'll define the data that we pass to BUGS "data." There are three lines of code that define this dataset as the cases/county and the population/county. Highlight those lines of code and hit alt-p. Next, define where you want your MCMC algorithm to begin:

inits ← function() list (theta=rep(0,100))

Remember, we have 100 values of theta so we need to specify 100 initial values. Here we use the rep (repeat) command, which creates a vector of zeros one hundred units long. Try altering this to start at locations other than zero and see what happens.

Now you need to tell BUGS which parameters you want to keep track of. You'll store the names of these parameters in the variable "parameters":

parameters←c("theta","lambda")

Finally, call BUGS with the command that begins
county.sim ←bugs(. . .)
I have this set up to call openBUGS, but you can change it to winBUGS if you want. Try it and you'll see why I like calling openBUGS a bit more. When you're troubleshooting programs, its sometimes useful to call one and than the other. They tend to give different types of error messages, so one may be

more helpful in locating your error.

It will take a few seconds for BUGS to run your model. When its done, your results will be saved in ”county.sim”. To examine the results you need to ”attach” the results, so run that line of code. Now if you type ”print(county.sim)” you’ll get a summary of your results. You can monitor convergence by typing plot(theta[,1]) to get the trace plot for theta[1].

I included the graphics commands I used at the bottom of the program to give you a head start on graphics. In general the easiest way to program in BUGS and R is just like in SAS: borrow as much code as you can from other sources. Use what we provide on this website as much as you can. Andrew Gelman has code for all the examples in his book on his website. The winBUGS program includes two volumes of example code that are very helpful too. For R, when I don’t know how to do something I do a quick google search and usually find an answer without much difficulty.