

Markov Chain Monte Carlo Exercise

In this exercise, we will study an implementation of the Metropolis-Hastings algorithm for the Ising model, and work with **BRugs** on selected machines on which it has been downloaded. I would encourage you to study **BRugs** first, and then work on the other exercise at your leisure.

BRugs

The **BRugs** library contains an **R** implementation of the popular **WinBugs** package. It can be downloaded from the **R** website for your personal use, but is not currently available on the department and college computers with a few exceptions.

BRugs requires the creation of three text files—a model file, a data file, and an initialization file. I have examples on the website for the one-way random effects model. Visit Brian Habing’s Spring 2006 STAT 778 website for an IRT example with some more flexibility built into it.

The model file is probably the most straightforward in structure. Note however that for any normal density, the second argument in the **dnorm** command is actually the precision (the inverse of the variance) rather than the variance. While atypical, it is actually more convenient when creating models for the precision, since the conjugate in this case is a Gamma density for the precision τ , rather than an Inverse Gamma distribution for the variance σ^2 . Note that the model file can contain the definition of seemingly extraneous variables; this occurs because **BRugs** would otherwise monitor only the chains of the parameters themselves, while we are often interested in other values as well. In the default example in **BRugs** (a study of weight gain in young rats), a linear predictor is actually the main focus of the study.

The data file should include not only the data set, but also any constants used in the model file, such as the number of observations, the number of items, the number of factor levels, factor level means, etc. All of these variables are saved in a simple list format. The only peculiarity is the format of the data matrix, which must be declared using the rather user-unfriendly format to be found in the data file on the website.

Brian includes an example that builds the data matrix from a comma-delimited file. This requires extensive use of the **paste** command to add commas, parentheses, and the necessary text. Though not necessary for a simple demonstration, it would be useful for repetitive analysis of the same data set, which is exactly what he had in mind for his IRT application.

Brian's approach has been superceded by the `BRugs` command `bugsData` which converts a named list in **R** to the correct data file format. Suppose you had already input or created the relevant data sets and variables in **R**. To use `bugsData`, you need to create a `list` object and then save it in a file. Though we have already constructed a data file, the commands below would create a data file named `BeerInit.txt` in the working directory. Run them (after having loaded `y` using the commands from the simple Gibbs sampler exercise) and verify that `BeerInit.txt` matches the syntax of the data file on the website.

```
setwd("z:/Stat 740/BRugs/")
BeerInit=list(J=6,K=8,y=y)
bugsData(BeerInit,file="BeerInit.txt")
```

The initialization file can simply be a list of start values for the parameters being modelled, as I have chosen for my demonstration. In practice, though, you should probably write an initialization function that computes good start values for both the parameters of interest and hyperparameters, where applicable. I chose to compute those values externally in **R**, and then provided answers as constants in the initialization file; some of the commands I used appear in the code I constructed in **R** as a simple example of a Gibbs sampler. Brian Habing uses an example of an initialization function whose last line creates the appropriate `list` command. Alternatively, `bugsInits` can assist in creating the file in the appropriate format.

To conduct a simulation, you could start with the following commands (after moving the model, data, and initialization files to the appropriate directory):

```
setwd("z:/stat 740/BRugs/")
modelCheck("modellwayre.txt")
modelData("data1wayre.txt")
modelCompile(numChains=3)
modelInits(rep("inits1wayre.txt",3))
```

These commands check the files for proper syntax and declare the number of chains (each chain needs to be initialized, hence we read the initialization file 3 times). Any error messages are cryptic and unhelpful. At this point, we are ready to start simulating. Usually, we burn-in the chains, and then start monitoring selected variables/parameters of interest. Once complete, there are several different approaches to examining results—most of these are easy

enough to re-create by hand, but the `BRugs` commands are very convenient. You may have to clear the graphing window in order for some of the graphics to appear.

```
modelUpdate(10) #We only need a small number of burn-ins for this chain
samplesSet(c("theta","mu","tst","tse"))
modelUpdate(50)
samplesStats("*")
samplesHistory("*",mfrow=c(3,3))
samplesDensity("theta")
```

Once finished with the one-way random effects example, you could look at other `BRugs` examples—use `help(BRugs)` to try the default exercise, for example.

Metropolis-Hastings exercise

I have included an implementation of the Metropolis-Hastings algorithm for an Ising model over a 50 x 50 grid. The code saves the output grid and the original grid, which is randomly generated with each cell set to 1 or -1 with equal probability .5. The algorithm replaces cells one at a time and each cell is replaced an average of 100 times by default. You will find that the default options take a while to run.

You can use the `levelplot` command after loading the `lattice` library. To implement this, you would generate the following commands:

```
mh.out=ising.mh()
x=1:50
y=1:50
grid=expand.grid(x=x,y=y)
r=as.vector(mh.out[[1]])
grid$z=r
levelplot(z~x*y,grid,main="Final grid for Ising model")
win.graph()
r=as.vector(mh.out[[2]])
grid$z=r
levelplot(z~x*y,grid,main="Initial grid Ising model",colorkey=FALSE)
```

Note the clustering in the Ising grid. You could also use a logical vector that compares the original file to the output file as your `z` vector in the above plot. The default value for β is .1; try .2 and compare your results.