Week 4: Some Computing Notes

by Michael Escobar

February 1, 2016

1 Introduction

In this handout, several of the computer programs that are used in the class are given. Sections 2 and 3 discuss some simple MCMC algorithm with code in R. Section 2 is for the simple normal models, and section 3 is for the simple ANOVA model. The three sections after that contain descriptions for files to run a simple linear regression with OpenBUGS. Section 4 discusses the files to run OpenBUGS manually. That is, when you open OpenBUGS and use a lot of "mousing around" to run OpenBUGS. This is useful to understand what OpenBUGS does, but mousing around quickly gets tiring. Section 5 shows some details of using the script command. This allows OpenBUGS to run some parts automatically. Still, it is a bother to have all those different files to run things. There are ways to run OpenBUGS through R. Commands for this are shown in Section 6.

2 Simple normal MCMC algorithm

The following code generates the simple MCMC sampler. This is the sampler which assumes:

- We have the following model for heights:

$$\begin{aligned} X_i|\mu,\tau &\sim N(X_i|\mu,\tau) \\ \mu|\mu_0,\tau,\theta &\sim N(\mu|\mu_0,\tau\theta) \\ \tau|\alpha,\beta &\sim \text{Gam}(\tau|\alpha,\beta) \end{aligned}$$

- Priors: $\mu_0 = 66$, $\theta = 4$, $\alpha = 1$, $\beta = 25$.
- Data: $X = (64, 73, 64, 63, 69, 71)$, $n = 6$, $\bar{X} = 67.333$, and $\sum(X_i - \bar{X})^2 = 89.33$.

To do this, please note the use of the following R functions:

- `for(i in 2:nbig){....}`: This program is a "loop" function in R. It will loop over the values in the `(...)` brackets and execute the statements in the `{..}` brackets each time. Here, the command will create a variable `i` and it will have values: $\{2, 3, \ldots, nbig\}$.

- `X[...]`: The brackets: `[...]` is the indexing function for a vector. If `X` is a vector, then an expression like `X[1]` is the first element of the vector and `X[i]` is the i-th element of the vector.

```
#
#  SimplMCMC.ssc
#  July 21, 2003
#
#   This is the code for the simple normal model
#

# prior
mu0<-66; theta<-4; alp<-1; bet<-25

# data
x<-c(64, 73, 64, 63, 69, 71)
xbar<-mean(x)
n<-length(x)

# starting values
muStart<-20;tauStart<-0.0025;

#  MCMC parameters

nbig<-50000


# Initialize chain

mu<-rep(0,nbig)
tau<-rep(0,nbig)

mu[1]<-muStart; tau[1]<-tauStart

mu0p<- (theta*mu0+n*xbar)/(theta+n)  # does not change in loop
alpp<-alp+(0.5)*(n+1)                # does not change in loop

tau0p<-rep(0,nbig)
betp<-rep(0,nbig)

for(i in 2:nbig){
# sample mu
tau0p[i]<-tau[i-1]*(n+theta)
mu[i]<-rnorm(1,mean=mu0p,sd=1/sqrt(tau0p[i]))

# sample tau
betp[i]<- bet+ 0.5*( sum( (x-mu[i])^2) + theta*(mu[i]-mu0p)^2)
tau[i]<-rgamma(1,alpp,betp[i])
}
 cbind(mu,tau)[1:20,]

# ---------------------------
```

3 Bugs code for line example

Below is the file which is used to run the OpenBugs example. (Please note that I did not list all 62 data points and used a series of dots to represent the missing data points.)

```
model{
for(i in 1:N){
lgbrain[i]<-log(brain[i])
lgbody[i]<-log(body[i])

lgbrain[i]~dnorm(mu[i],tau)
mu[i]<-alpha +beta*lgbody[i]
}
alpha~dnorm(0,.0001)
beta~dnorm(0,.0001)
tau~dgamma(.0001,.0001)
}

Initialization:

list(alpha=1,beta=1,tau=1)


Data:

list(N=62)

body[] brain[]
3.385 44.5
.480 15.5
```

```
1.35 8.1
465 423
36.33 119.5
.
.
.      {{{ note, these dots represent the missing 62 points}}}
.
.9 2.6
1.62 11.4
.104 2.5
4.235 50.4
END
```

4 Running OpenBugs Through R

The basic idea is that R creates all the files to run OpenBUGS in batch mode and then launches OpenBUGS to execute these commands. Then, it collects the output in R so that more can be done.

To run OpenBUGS through R, one can use the R package R2OpenBUGS. One needs to get this package from the same place that you got R. There are also similar packages like brugs and rbugs which do similar things.

In order to run the example of the linear regression that I did in sections 3 through R, I will use the same "Model" file that I used to run the script command. Also, I need to enter the data in R. I did this by creating a ".csv" file (by using Excel). It is not necessary to input it via a csv file, one needs to use one of the many ways of input data in R.

Note: the commands run here are in the file: `brnbdyR2Openbugs.txt`.

The following lines are then input into R. First, some initial booking which loads the working directory and loads the R package.

```
WorkDir<-"c:/mike workstation/BayesCourse13/Documents/Temp"
# Note: change this to the file where everything is
setwd(WorkDir)

library(R2OpenBUGS)
```

Then, create the data file by copying the data into a csv file. (Note, obviously don't need to do this if it already exist. This step is included here to show you how you might do it.)

```
#  data a bit, long... code after setting up datafile.
```

```
cat("body,brain
3.385,44.5
.
.   << note: all the numbers are not copied here... see the actual file>>
.

4.235,50.4
",file="brnbdy.csv")

brnbdy.dat<- read.table(file="brnbdy.csv",header=TRUE,sep=",")
attach(brnbdy.dat);
N<-62
data<-list("N","body", "brain")
```

The following code does the following: a) creates a function which will initialize a chain, b) creates a file where the model is defined. Note, using `cat` command is one way to create this file. An advantage of doing this in this command file is that it keeps everything in one file.

```
inits<-function(){ list(alpha=rnorm(1),beta=rnorm(1),tau=runif(1,.5,10))}
parameters<-c("alpha", "beta", "tau")

cat("model{
for(i in 1:N){
lgbrain[i]<-log(brain[i])
lgbody[i]<-log(body[i])

lgbrain[i]~dnorm(mu[i],tau)
mu[i]<-alpha +beta*lgbody[i]
}
alpha~dnorm(0,.0001)
beta~dnorm(0,.0001)
tau~dgamma(.0001,.0001)
}
", file="NonCenterMod.txt")
```

Now, use `bugs` function to launch OpenBugs. Let `debug=TRUE` to allow you to go into OpenBugs and check for errors or to get into OpenBugs and maybe do more inside OpenBugs.

```
### run Openbugs through R:
brnbdy.sim<-bugs(data,inits, parameters,model.file="NonCenterMod.txt",
  n.chains=3, n.iter=10000, n.burnin=700,
  n.thin=1,  #,debug=TRUE
  )
```

Now, you can do some postprocessing. This includes printing the results and getting some plots. Note that the sampled values from the MCMC are in the array `sims.array` which is in the output list from the `bug` command.

```
# post process the MCMC values.
print(brnbdy.sim)
str(brnbdy.sim)

out.sim<-brnbdy.sim$sims.array

plot(out.sim[,1,"alpha"],out.sim[,1,"beta"],xlab="alpha",ylab="beta")
junk=locator(1)
#### locator() stops the plot and waits for you to click on the plot

#  simple trace plot:
plot(out.sim[,1,"alpha"],type="l",ylab="alpha")
junk=locator(1)
#  all three chains on same plot:
ts.plot(out.sim[,,"alpha"],col=c(1,2,3))
junk=locator(1)
# density plot:
plot(density(out.sim[,,"alpha"]))
junk=locator(1)
# put all three densities on same plot (need to guess ranges)
plot(c(1.5,3.0), c(0, 5),type="n",
  ylab="", xlab="alpha",main="posterior density of alpha", sub="all three chains")
for(i in 1:3){lines(density(out.sim[,i,"alpha"]))}
junk=locator(1)
#
#  put multiplots on one plot
#
par(mfrow=c(2,2))
for(i in 1:3){ plot(density(out.sim[,,parameters[i]]),main=parameters[i])}
plot(density(log(out.sim[,,"tau"])),main="log(tau)")
par(mfrow=c(1,1))
junk=locator(1)
```

```
### this will calculate the autocorrelation:
# with the cross corr for the different chains:
acf(out.sim[,,"alpha"],lag.max=40)
junk=locator(1)
# For one chain:
acf(out.sim[,1,"alpha"],lag.max=40)
```

These series of commands to the following:

1. The `library` commands loads the R2OpenBUGS package. This assumes that you have already downloaded the package first.

2. `setwd` sets the working directory to where the various files that you will be using are located.

3. The `read.table` command loads the data from a file into an R database. This database is then "attached" with the `attach` command. Also, for the program, I will want to set `N` to the value 62.

4. The `bugs` is the function which sets up the necessary files, runs OpenBUGS, and then sends the output to the object `brnbdy.sim`.

5. Note that the first four arguments to the `bugs` are related to the data, the initiation values, the monitored parameters, and the model file. Check the documentation on what is accepted here. For example there is a way to specify the model as an R function as opposed to putting the model in a file.

6. The next series of arguments to the command `bugs` are related to how long to run the MCMC chain, issues of burn-in and thinning and also to then number of chains to run.

7. There are also other arguments that are useful. In the above command, I have commented out the use of the `debug` option. This option is useful if you want to not immediately close OpenBUGS after the various commands have been run. This is useful if there are errors in what you are doing and you want to debug the submitted statements.

Some comments on these commands used in post processing the MCMC chain.

- When one prints the bugs object, one receives the summary statistics from the MCMC program.

- The values that are sampled from the output are contained in `brnbdy.sim$sims.array`. This is a 3-dimensional array. The first dimension is the iteration number (after the burn in). The second dimension gives the chain number. In this run there were three chains so the middle dimension has 3 values. The third dimension has 4 different values corresponding to the three monitored parameters and the monitored deviance value.

- The command `ts.plot` gives a time series plot.

- The command `acf` will plot of the autocorrelation versus the value of the lag. (Here, I restricted the lag to a maximum of 40.)

- Commands like `plot(density(....))` estimate the density of a parameter.

- The command `junk=locator(1)` stops the R program so that you can look at the plots one at a time. To continue the plotting of new plots, click on the plot. (What this command does is stop R and wait for you to identify a point on the plot. When you click on the plot, the function returns the point on the plot that you clicked on.)

- The `par` command controls higher level plotting values. The command `par(mfrow=c(2,2))` tells R to create the next four plots as subplots in a 2x2 grid of plots.

5 Running OpenBugs with a script file

Instead of always using the GUI to run OpenBugs, it is often easier to run a script file to run OpenBugs. This is especially true when one is trying to run many models. With the R2OpenBugs package, you will almost always want to run the batch jobs through R. So, you might want to ignore this section. (Originally, R2xxxxBugs where written to basically create these script files and then open up the xxxBugs program. I don't know if that is still the way that these functions are run.)

To run Openbugs this way you need a script file and some other supplementary files. Here, there is one script file, two data files, and three files to initiate three different parallel chaings. The following file is the OpenBugs script file.

```
#
# this file analyses the brain/body data
#     this is the script file for OpenBugs

modelDisplay("log")

modelCheck(
 'c:/mike workstation/BayesCourse10/Week 4 files/ScriptBrnBdy/brnbdyNCMod.txt')
```

```
modelData(
 'c:/mike workstation/BayesCourse10/Week 4 files/ScriptBrnBdy/brnbodyData1.txt')
modelData(
 'c:/mike workstation/BayesCourse10/Week 4 files/ScriptBrnBdy/brnbodyData2.txt')
modelCompile(3)
modelInits(
 'c:/mike workstation/BayesCourse10/Week 4 files/ScriptBrnBdy/brnbdyNCinit1.txt',1)
modelInits(
 'c:/mike workstation/BayesCourse10/Week 4 files/ScriptBrnBdy/brnbdyNCinit2.txt',2)
modelInits(
  'c:/mike workstation/BayesCourse10/Week 4 files/ScriptBrnBdy/brnbdyNCinit3.txt',3)

samplesSet(alpha)
samplesSet(beta)
samplesSet(tau)


modelUpdate(10000)
samplesStats("*")
samplesDensity("*")
samplesTrace("*")
samplesStats("*")
```

The model file which is called is:

```
model{
for(i in 1:N){
lgbrain[i]<-log(brain[i])
lgbody[i]<-log(body[i])

lgbrain[i]~dnorm(mu[i],tau)
mu[i]<-alpha +beta*lgbody[i]
}
alpha~dnorm(0,.0001)
beta~dnorm(0,.0001)
tau~dgamma(.0001,.0001)
}
```

The two data files are files which contain the information on the data. The initialization files follow.

First data file:

```
list(N=62)
```

Second data file. (Note, again, not all the data points are listed):

```
body[] brain[]
3.385 44.5
.480 15.5
1.35 8.1
465 423
36.33 119.5
.
.
.      {{{ note, these dots represent the missing 62 points}}}
.
.9 2.6
1.62 11.4
.104 2.5
4.235 50.4
END
```

The initiation files all look similar to the following:

```
list(alpha=1,beta=1,tau=1)
```