JAGS-1.0.3: An Overview

Rebecca Steorts

1. Go to the JAGS webpage (http://www-fis.iarc.fr/~martyn/software/jags/) maintained by Martyn Plummer and download the JAGS manual. The manual describes how to install JAGS on all platforms.

   Note to Mac Os X users: You will need X11 and XCode for installation.
   Note: JAGS is installed on the Linux systems in the Statistics Department.
   You run: /depot/JAGS-1.03/bin/jags

   If you're interested in more examples, try looking at the WinBUGS webpage http://www.mrc-bsu.cam.ac.uk/bugs/ .

2. Create 4 files to be used in JAGS (model.txt, data.txt, initial.txt, script.txt). For the PlA2 data, these are on my webpage.

   The file **model.txt** contains the model statement for the model that you're interested in. The file **data.txt** contains the data that you want to read into JAGS. The data must be stored as c(1,3,5). Furthermore, you set your value of N here from your model statement. For example, in the PlA2 data, we set N <- 12 here. See the data file for an example. The file **initial_1.txt** is where you set the initial values for the parameters in your model. You can also set your seed here if you wish. To set the seed do the following:

   ```
   ".RNG.name" <- "base::Super-Duper"
   ".RNG.seed" <- 12
   ```

   There are three other bases available in JAGS. These are Wichmann-Hill, Marsaglia-Multicarry, and Mersenne-Twister. There is another way to set the seed in JAGS. See page 11 of the manual for details. The last file you need to create, **script.txt**, is a script telling JAGS what commands you would like it to run. To run your script in the terminal type

   ```
   jags script
   ```

If you have any errors, they will print in the terminal screen as they produce so you have to look carefully or you will easily miss seeing them.

The script file looks like

```
model clear
data clear
model in "model.txt"
data in "data.txt"
compile
inits in "initial_1.txt"
initialize
update 10000
monitor mu
monitor psi
monitor gam
update 100000
coda *
```

where update 10000 stands for taking a burn-in of 10000. These values will not be reported in the coda file. The update of 100000 reflects the number of iterations we will run the Gibbs sampler for after taking the burn-in.

After JAGS runs your script, your Gibbs sampler output will produce in two files, CODAchain.txt and CODAindex.txt. The first file, contains a complied vector of all output from the Gibbs sampler of monitored parameter values that can be read into R. The second file lists the parameter values that you monitored so that R will know how to read in the coda file properly.

3. You should now create a fifth file, **pla2.R**, that will read in the coda from JAGS, run diagnostic procedures, find summary statistics, and estimate posterior densities. To do all this in R, you will need to install the coda package and load it.

To read in the data and find the trace, autocorrelation plots, and compute the Geweke diagnostic we do:

```
library(coda)
res = read.coda("CODAchain.txt", "CODAindex.txt")
mu = res[,1]
plot(as.numeric(mu), xlim = c(0,2000), type = 'l',
xlab = "Iteration", ylab = expression(mu)) ##trace plot
acf(mu, lag.max = 40, xlab = "Lag", ylab = "Correlation", main="")
## autocorrelation plot
```

```
geweke.diag(mcmc.list(res1)) ## geweke diagnostic
```

Remember if we want to compute the Gelman-Rubin we need to run two (different) chains with two different seeds. I could not get this to work in JAGS simultaneously running two chains together with two separate seeds for some reason. When I did this, I kept getting the same two chains as output. I'm not sure why this is happening. To avoid this problem, I simply ran two separate chains using two different seeds and then compared the two chains. This is a bit inconvenient, but it works. Recall, that the Gelman-Rubin diagnostic may not be valid anyway, so we could just look at the Geweke diagnostic above and avoid this.

You can reproduce this on your own if you would like using the code on my website. To obtain my results, I ran initial_1 and initial_2 as my two starting values. My outputs from the Gibbs sampler are CODA_iter1.txt, CODA_iter2.txt, and CODAindex_iter1.txt.

For simplicity, I wanted to start with the one chain case above. The commands to run the Gelman-Rubin diagnostic procedures is below

```
library(coda)
res1 = read.coda("CODA_iter1.txt", "CODAindex_iter1.txt")
res2 = read.coda("CODA_iter2.txt", "CODAindex_iter1.txt")
gelman.diag(mcmc.list(res1,res2))
```

4. After running diagnostic procedures, you should go back and thin your chain. In the PlA2 example, I used a thin of 2 and worked with the initial_1.

The new script file looks like

```
model clear
data clear
model in "model.txt"
data in "data.txt"
compile
inits in "initial_1.txt"
initialize
update 10000
monitor mu, thin(2)
monitor psi, thin(2)
monitor gam, thin(2)
update 100000
coda *
```

where now I've just told the script how much to thin each parameter. Now, we can re-read the new coda file into R. The summary function calculates the mean, standard errors, and credible intervals for the posterior of each monitored parameters. We can also estimate the posterior distribution using the density function. For instance, we can do these in R by the following commands

```
res1.thin = read.coda("CODAchain1_thin.txt", "CODAindex_thin.txt")
summary(exp(res1.thin))
summary(mu1.thin)$statistics["Mean"]
summary(mu1.thin)$statistics["SD"]
summary(mu1.thin)$quantiles[c("2.5%","97.5%")]

#posterior of mu
mu1.thin = res1.thin[,1]
mu1.dense = density(mu1.thin)
plot(mu1.dense, main="", xlab = expression(mu), ylab = "Density")


# plot posterior of exp(psi's)
psi1.thin = res1.thin[,2]
psi1.dense = density(exp(psi1.thin))
psi2.thin = res1.thin[,3]
psi2.dense = density(exp(psi2.thin))
psi3.thin = res1.thin[,4]
psi3.dense = density(exp(psi3.thin))
psi4.thin = res1.thin[,5]
psi4.dense = density(exp(psi4.thin))
par(mfrow=c(2,2),mar=c(3,3,1,1))
plot(psi1.dense,xlab = "", ylab = "" , xlim = c(0,3), ylim = c(0,4),
main = "")
legend("topright",expression(exp(psi[1])), cex = 1.25, bty = 'n')
plot((psi2.dense),xlab = "", ylab = "" , xlim = c(0,3), ylim = c(0,4),
main = "")
legend("topright",expression(exp(psi[2])), cex = 1.25, bty = 'n')
plot((psi3.dense),xlab = "", ylab = "" , xlim = c(0,3), ylim = c(0,4),
main = "")
legend("topright",expression(exp(psi[3])), cex = 1.25, bty = 'n')
plot((psi4.dense),xlab = "", ylab = "" , xlim = c(0,3), ylim = c(0,4),
main = "")
legend("topright",expression(exp(psi[4])), cex = 1.25, bty = 'n')
```