Introduction to Applied Bayesian Modeling
A brief JAGS and R2jags tutorial

Johannes Karreth*
University of Georgia

jkarreth@uga.edu

ICPSR Summer Program 2011
Last updated on July 7, 2011

1 What are JAGS, R2jags, ...?

JAGS is **J**ust **A**nother **G**ibbs **S**ampler that was mainly written by Martyn Plummer in order to provide a BUGS engine for Unix. More information can be found in the excellent JAGS manual at `http://sourceforge.net/projects/mcmc-jags/` .

R2jags is an R package that allows running JAGS models from within R. It was written by Andrew Gelman et al. with the purpose of providing identical functionality to the R2WinBUGS package on Windows machines. Almost all examples in Gelman and Hill's *Data Analysis Using Regression and Multilevel/Hierarchical Models* can thus be run equivalently in JAGS, using R2jags.

*R code and other material for using JAGS in this course are available at `http://jkarreth.myweb.uga.edu/bayes2011.htm`

rjags is another R package that allows running JAGS models from within R.

2 Installing JAGS and R2jags on Mac OS X

1. Install R version 2.13.0 from the CRAN website: `http://cran.r-project.org/bin/macosx`. If you are already running a different version of R, you can (but need not) uninstall it by typing
`rm -rf /Library/Frameworks/R.framework /Applications/R.app`
in the Terminal.

2. Install the Tcl/Tk libraries (`tcltk-8.5.5-x11.dmg`) and GNU Fortran (`gfortran-4.2.3.dmg`) from the CRAN tools directory: `http://cran.r-project.org/bin/macosx/tools`.

3. Install JAGS version 1.0.4u.dmg from Martyn Plummer's archive: `http://www-fis.iarc.fr/~martyn/software/jags/old/mac`. This step appears (used?) to be necessary because only 1. versions of JAGS create the files that rjags (see next step) will require during its installation. Start the Terminal and type
`jags`
to see if JAGS 1.0.4 is installed.

4. Now install JAGS version 2.2.0 (`JAGSdist-2.2.0.dmg`) from Martyn Plummer's repository: `http://sourceforge.net/projects/mcmc-jags/files/JAGS/2.x/Mac%20OS%20X/`. Start the Terminal and type
`jags`
to see if JAGS 2.2.0 is installed.

5. Install the packages R2jags, coda, R2WinBUGS, lattice, and (lastly) rjags from within R, via the Package Installer.

6. Consider using a scientific text editor for writing R and JAGS code. A list of good editors for Mac OS X is here: `http://www.pure-mac.com/textword.html`. RStudio is a very neat integrated environment for running R on a Mac (and other platforms): `http://www.rstudio.org`.

3 Fitting Bayesian models in JAGS

3.1 Using R2jags

Just like R2WinBUGS[1], the purpose of R2jags is to allow running JAGS models from within R, and to analyze convergence and perform other diagnostics right within R. A typical sequence of using R2jags could look like this:

- `> library(R2jags)`

- Read the data in from the car package:

```
> library(car)
> data(Angell)
> angell.1 <- Angell[, -4]
```

- Save the model as "angell.model.jags" in your working directory. (Do not run this model from within R.) One convenient way to do this is to write your model in a text editor (see above) and save it in your WD; make sure that if your text editor adds a file extension (e.g., `.txt`), that you specify the *complete* file name, including this extension, in the `jags()` command below. You can set your working directory in the R preferences, or via:

[1] See Appendix C in Gelman and Hill (2007) or their online appendix `http://www.stat.columbia.edu/~gelman/bugsR/runningbugs.html` for more info on how to run R2WinBUGS and R.

```
> setwd("/Users/johanneskarreth/R/Bayes/angell.demo")
```

The model looks just like the BUGS models shown in class:

```
model {
for(i in 1:N){
moral[i]~dnorm(mu[i], tau)
mu[i]<-alpha + beta1*hetero[i] + beta2*mobility[i]
}

alpha~dnorm(0, .01)
beta1~dunif(-100000,100000)
beta2~dunif(-100000,100000)
tau~dgamma(.01,.01)
}
```

- Now define the vectors of the data matrix for JAGS:

```
> moral <- angell.1$moral
> hetero <- angell.1$hetero
> mobility <- angell.1$mobility
> N <- length(angell.1$moral)
```

- Read in the Angell data for JAGS

```
> angell.data <- list("moral", "hetero", "mobility", "N")
```

- Define the parameters you're interested in:

```
> angell.params <- c("alpha", "beta1", "beta2")
```

- Define the starting values for JAGS

```
> angell.inits <- function() {
+     list(alpha = c(20), beta1 = c(-0.1), beta2 = c(-0.02))
+ }
```

- If you want to specify different starting values for each chain:

```
> inits1 <- list(alpha=0, beta1=0, beta2=0)
> inits2 <- list(alpha=1, beta1=1, beta2=1)
> angell.inits <- list(inits1, inits2)
```

- Fit the model in JAGS. Make sure that if your text editor adds a file extension (e.g., `.txt`), that you specify the *complete* file name, including this extension, in this command:

```
> angellfit <- jags(data = angell.data, inits = angell.inits, angell.params,
+     n.chains = 2, n.iter = 9000, n.burnin = 1000, model.file = "angell.model.jags")
```

- Update your model if necessary - e.g. if there is no/little convergence:

```
> angellfit.upd <- update(angellfit, n.iter = 1000)
> angellfit.upd <- autojags(angellfit)
```

The latter command will auto-update until convergence - I believe it uses the Gelman-Rubin statistic to assess convergence.

- View your results:

```
> print(angellfit)

Inference for Bugs model at "angell.model.jags", fit using jags,
 2 chains, each with 9000 iterations (first 1000 discarded), n.thin = 8
 n.sims = 2000 iterations saved
         mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
alpha     19.654   1.234  17.256  18.819  19.632  20.487  22.045 1.003   630
beta1     -0.107   0.018  -0.142  -0.119  -0.107  -0.095  -0.071 1.003   660
beta2     -0.186   0.036  -0.257  -0.211  -0.185  -0.162  -0.113 1.002  1100
deviance 192.721   3.016 188.941 190.492 192.149 194.198 200.037 1.001  2000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 4.6 and DIC = 197.3
DIC is an estimate of expected predictive error (lower deviance is better).

> plot(angellfit)
```

- Diagnostics:

```
> traceplot(angellfit)
```

- If you want to print and save the plot, you can use the following set of commands:

```
> pdf("angell.trace.pdf")
```

... defines that the plot will be saved as a PDF file with the name "angell.trace.pdf" in your working directory.

```
> traceplot(angellfit)
```

creates the plot in the background (you will not see it).

```
> dev.off()
```

finishes the printing process and creates the PDF file of the plot. If successful, R will display the message `"null device 1"`.

- Generate MCMC object for analysis

```
> angellfit.mcmc <- as.mcmc(angellfit)
> summary(angellfit.mcmc)
```

- Plot:

```
> xyplot(angellfit.mcmc)
```

- Density plot:

```
> densityplot(angellfit.mcmc)
```

- Trace- and density in one plot, print directly to your working directory:

```
> pdf("angellfit.mcmc.plot.pdf")
> plot(angellfit.mcmc)
> dev.off()
```

- Autocorrelation plot, print directly to your working directory:

```
> pdf("angellfit.mcmc.autocorr.pdf")
> autocorr.plot(angellfit.mcmc)
> dev.off()
```

- Other diagnostics using CODA:

```
> gelman.plot(angellfit.mcmc)
> geweke.diag(angellfit.mcmc)
> geweke.plot(angellfit.mcmc)
> raftery.diag(angellfit.mcmc)
> heidel.diag(angellfit.mcmc)
```

- If you want to examine the Coda files via Boa or Coda, you could use the function jags2. Note that we specify `clearWD=FALSE` to keep the coda files in the working directory:

```
> angell.params <- c("alpha", "beta1", "beta2")
> angellfit <- jags2(data = angell.data, inits = angell.inits, angell.params,
+     n.iter = 5000, model.file = "angell.model.jags", clearWD=FALSE)
```

- This will create the following files in your working directory:

```
/Users/johanneskarreth/R/CODAchain1.txt
/Users/johanneskarreth/R/CODAchain2.txt
/Users/johanneskarreth/R/CODAindex.txt
/Users/johanneskarreth/R/jagsdata.txt
/Users/johanneskarreth/R/jagsinits1.txt
/Users/johanneskarreth/R/jagsinits2.txt
/Users/johanneskarreth/R/jagsscript.txt
```

- ... which can then be analyzed using BOA

```
> library(boa)
> boa.menu()
```

- or CODA (see below).

- In both cases (using boa or coda), remember to rename the index file's extension to `.ind`, and the chain files to `.out`.

Good resources to find more information about R2jags:

- Yu-Sung Su (Columbia) co-wrote the R2jags package:
  `http://yusung.blogspot.com/2008/05/r2jags-package-for-running-jags-from-r.html`
- Gelman and Hill's book website with code:
  `http://www.stat.columbia.edu/~gelman/arm/software/`
- Rebecca Steorts' presentation on JAGS:
  `http://www.stat.ufl.edu/~rsteorts/jags_present.pdf`

3.2 Using JAGS via Terminal

JAGS can also be run straight from the command line - on Windows and Unix systems alike. Probably the most feasible way to do this is to write a script file with the following parts, and save it as `angell.jags`:

```
model in angell.mod
data in angell.dat
compile, nchains(2)
parameters in angell.inits
initialize
update 10000
monitor alpha
monitor beta1
monitor beta2
update 2500
coda *, stem(angell_out)
```

You can run this script file by opening a Terminal window, changing to the working directory (WD) in which all the above files are located -

```
cd /Users/johanneskarreth/R/Bayes/angell
```

and then simply telling JAGS to run the script:

```
jags angell.jags
```

In more detail:

- `model in angell.mod`

  Use the model angell.mod, which is saved in your WD, and looks like a regular BUGS model:

  ```
  model {
  for(i in 1:N){
  moral[i]~dnorm(mu[i], tau)
  mu[i]<-alpha + beta1*hetero[i] + beta2*mobility[i]
  }

  alpha~dnorm(0, .01)
  beta1~dunif(-100000,100000)
  beta2~dunif(-100000,100000)
  tau~dgamma(.01,.01)
  }
  ```

- `data in angell.dat`

  Use the data angell.dat. These data can be saved as vectors in one file that you name `angell.dat`:

  ```
  "moral" <- c( 19, 17, ...)
  "hetero" <- c( 20.6, 15.6, ...)
  "mobility" <- c( 15, 20.2, ...)
  "N" <- 43
  ```

- `compile, nchains(2)`

  Compile the models and run two Markov chains.

- `parameters in angell.inits`

  Use the starting values you provide in angell.inits, which could look like this:

  ```
  "alpha" <- c(0)
  "beta1" <- c(0)
  "beta2" <- c(0)
  ```

- `initialize`

  Initialize and run the model.

- `update 10000`

  Specify 10000 updates.

- `monitor alpha`
  `monitor beta1`
  `monitor beta2`

  Monitor these values.

- `update 2500`

- `coda *, stem(angell_out)`

  Tell JAGS to produce coda files that all begin with the stem `angell_out` and are put in your WD.

4 Using the coda and boa packages in R

You can use either of the two packages boa or coda in R to analyze your BUGS/JAGS output, regardless of the BUGS software you used to fit your model (i.e. WinBUGS/OpenBUGS/JAGS). These are the key steps to get your data into R to use boa or coda:

- Fit your model in WinBUGS/OpenBUGS/JAGS, and identify where your software saved the chains and index files – most likely in the working directory where the other components of your model (data, model, inits) are.
- Give these files a proper name, for instance `angell_out_chain1.txt` and `angell_out_chain2.txt`.
- In R, load the boa or coda package, whichever you prefer:

  ```
  > library(boa)
  > library(coda)
  ```

- Now, read your chain and index files into R, via the commands below.
- If you prefer the usual R command-line behavior to the boa/coda menu option, both packages also can be used via the command line.

4.1 Diagnostics with the coda package

- Set your working directory:

  ```
  > setwd("/Users/johanneskarreth/R/Bayes/angell.demo")
  ```

- Read in your BUGS/JAGS output. This requires that the chains and index files (see above) are in your working directory.

```
> chain1 <- read.coda("angell_out_chain1.txt", "angell_out_index.txt")
> chain2 <- read.coda("angell_out_chain2.txt", "angell_out_index.txt")
> angell.chains <- as.mcmc.list(list(chain1, chain2))
```

- Now you can analyze using some of the commands listed in

  ```
  > help(package = coda)
  ```

  for instance:

  ```
  > summary(angell.chains)
  > traceplot(angell.chains)
  ```

  You should be able to play around with graphical parameters and different printing devices this way.

4.2 Diagnostics with the boa package

- Start the boa session:

  ```
  > boa.init()
  ```

- Read in your BUGS/JAGS output. This requires that the chains and index files (see above) are in your working directory.

  ```
  > my.model.chains <- boa.chain.import(angell_out_, path = boa.par("~/Bayes/angell.demo/",
  +     type = "BUGS"))
  ```

  where `angell_out` is the stem of your chains and index files; and `"~/Bayes/angell.demo/"` is the folder where your chains and index files are saved.

- Now you can analyze using some of the commands listed in

  ```
  > help(package=boa)
  ```

  for instance:

  ```
  > summary(angell.chains)
  > boa.plot(angell.chains)
  ```

  You should be able to play around with graphical parameters and different printing devices this way.

5 Differences between JAGS and WinBUGS

5.1 Logit models and the p.bound workaround:

Andrew Gelman's blog entry[2] on specifying p.bound:

> *Aurélien Madouasse writes:*
>
> > *I am currently fitting a multilevel logistic model using WinBUGS. I have adapted the example you provide in 'Data analysis using Regression and Multilevel/Hierarchical Models' p. 381-2:*
> >
> > ```
> > y[i] ~ dbin(p.bound[i], 1)
> > p.bound[i] <- max(0, min(1, p[i]))
> > logit(p[i]) <- Xbeta[i] ...
> > ```

[2] http://www.stat.columbia.edu/~cook/movabletype/archives/2009/04/pbound_in_multi.html

> *I was wondering what is the aim of the p.bound variable. When I use p[i] instead WinBUGS crashes. I thought that the inverse logit was bounded between 0 and 1 so I don't see the point of constraining it to be in this interval. What do I miss?*

My reply: Yes, inverse logit is bounded, but I think Bugs sometimes messes up and gets it outside of the bound. The other thing is that when you change the specification (in this case, using p.bound) it changes the sampler that Bugs uses. Maybe it's switching from the buggy adaptive rejection sampler to the foolproof Metropolis or slice sampler.

5.2 Ordered logit models in JAGS

In ordered logit (probit) models, you need to ensure that the estimated cut points are sorted in order. From the JAGS manual (p. 36):

> Prior ordering of top-level parameters in the model can be achieved using the sort function, which sorts a vector in ascending order. Symmetric truncation relations like this
>
> ```
> alpha[1] ~ dnorm(0, 1.0E-3) I(,alpha[2])
> alpha[2] ~ dnorm(0, 1.0E-3) I(alpha[1],alpha[3])
> alpha[3] ~ dnorm(0, 1.0E-3) I(alpha[2],)
> ```
>
> should be replaced by this
>
> ```
> for (i in 1:3) {
> alpha0[i] ~ dnorm(0, 1.0E-3)
> }
> alpha[1:3] <- sort(alpha0)
> ```

Accordingly, we need to adjust the code for a simple ordered logit model like displayed below.

5.2.1 BUGS

```
model{
for(i in 1:N){
for(j in 1:2){
logit(gamma[i,j]) <- theta[j] - mu[i]
}
quality[i] ~ dcat(p[i,1:3])
p[i,1]<- gamma[i,1]
p[i,2] <- gamma[i,2] - gamma[i,1]
p[i,3] <- 1-gamma[i,2]
mu[i] <-  b[1]*price[i] + b[2]*sodium[i] + b[3]*alcohol[i]+b[4]*calories[i]
}

}
for(m in 1:4){
b[m] ~ dnorm(0, .0001)
}
theta[1] ~ dnorm(0,.1)I(0, theta[2])
theta[2] ~ dnorm(0,.1)I(theta[1], )
}
```

5.2.2 JAGS

```
model{
for (i in 1:N){
for (j in 1:2){
logit(gamma[i,j]) <- theta1[j] - mu[i]
}
quality[i] ~ dcat(p[i,1:3])
p[i,1]<- gamma[i,1]
p[i,2] <- gamma[i,2] - gamma[i,1]
p[i,3] <- 1-gamma[i,2]
mu[i] <-  b1*price[i] + b2*sodium[i] + b3*alcohol[i]+b4*calories[i]
}
for (i in 1:2) {
theta[i] ~ dnorm(0, 1.0E-3)
}
theta1[1:2] <- sort(theta)
b1 ~ dnorm(0, .1)
b2 ~ dnorm(0, .1)
b3 ~ dnorm(0, .1)
b4 ~ dnorm(0, .1)
}
```

5.3 Truncation (constraining variables)

In some contexts, you might want to constrain a variable to assume a limited range of values a priori, e.g., specifying for a variable to have positive values only.

5.3.1 BUGS

In BUGS, you would *also* use the `I( , )` fragment to constrain variables, for instance:

```
theta ~ dnorm(0, 0.001) I(0, )
```

5.3.2 JAGS

In JAGS, this constraint (truncation) of the variable is handled differently:

```
theta ~ dnorm(0, 0.001) T(0, )
```

For more information on how JAGS separates between the concepts of censoring, truncation, and ordering, see pp. 35-36 of the JAGS manual at `http://softlayer.dl.sourceforge.net/project/mcmc-jags/Manuals/2.x/jags_user_manual.pdf`.