

A simple jags demo

December 14, 2009

1 An introduction to BUGs/WinBUGs/JAGs

BUGs/WinBUGs/JAGs is a successful general purpose MCMC sampler for inference. It is very useful for inference, and particularly Bayesian inference. The acronym BUGS denotes “Bayesian inference Using Gibbs Sampling”. We shall be using the JAGS software (“Just Another Gibbs Sampler”) which is based on BUGS. It has however been designed so that it can be more easily adapted by other researchers. For example, it is still a big topic in research to determine how you measure whether a model fits your data well. JAGS was written to make it possible to include different diagnostic fit measures in the program.

The algorithm that JAGS uses is a form of MCMC known as Gibbs sampling. You will take a course next year in which you study the underlying theory of Markov chains as well as Monte Carlo integration formally. Here, we shall use these tools in a very intuitive way in order to complete our exploration of Bayesian Inference. The essential idea in Gibbs sampling is that we simulate from the conditional density of each parameter in turn, given the current simulated values of all other parameters and the data. When we don’t have an expression for the conditional density of a parameter, it is possible to use a version of rejection sampling known as adaptive rejection sampling to simulate from that density.

1.1 Specifying the model

You have been given a model file `binomial.jags` which can be used to fit a hierarchical model for π_i to the rat litter data we discussed on Monday. We believe the data are distributed as a Binomial density, and we believe that π_i that is a random variable itself taken from a population of such π_i s. So at the first level we have:

$$y_i \sim \text{Binomial}(\pi_i, n_i)$$

in the model we discussed on Monday, we decided that:

$$\pi_i \sim \text{Beta}(a, b)$$

On Monday we also saw that we should put a density on a and b , such that:

$$p(a, b) \propto (a + b)^{-5/2}$$

It is important that we use such a density to limiting the number of simulated values a and b where $a+b$ is large. It is not easy in JAGs to define our own priors, so for the purposes of demonstration we shall use an independent Gamma prior for a and b . This Gamma density will have a value $a = 0.01$, $r = 0.01$. This means we are predicting that the prior mean is 1, but that the prior variance is very large. However, we might worry about the posterior when $a + b$ is large and for reasons like this many researchers write their own McMC samplers in native code.

1.2 JAGs language

Perhaps it is easiest to explain JAGS by looking at the model specification. In JAGS, we would write this Beta-Binomial model as:

```
model{
  for (i in 1:N){
    y[i] ~ dbin(pi[i], n[i])
    pi[i] ~ dbeta(a,b)
  }
  a ~ dgamma(1, 0.01)
  b ~ dgamma(1, 0.01)
}
```

In order to explain the model formulation you might notice that:

- The language is slightly R-like
- The entire model is wrapped within `model{...}`
- The individual data specification is wrapped within `for (i in 1:N){...}` construction.
- \sim denotes a random variable, and \leftarrow denotes a function. So for example:
 - `y[i] ~ dbin(pi[i], n[i])` is telling us that each data point $y_i \sim \text{Binomial}(\pi_i, n_i)$
- We have some additional priors outside the data loop, such as $a \sim \text{Gamma}(0.02, 0.01)$. This is suggesting the prior mean is 2, but that the prior variance is really quite large.

Normal working practice with JAGS is to start with a model that works, and gradually increase the complexity of the model until we have a JAGS model code that matches the model we wish to analyse.

1.3 Running JAGs

It is possible to run an entire analysis from a JAGS script. However, in order to understand all aspects of the use of simulation we shall walk through one analysis. For this first demonstration, you have been given a file containing some working jags code, `betabinomial.jags`, a file containing some data in jags format `rats.dump` as well as some initial values in `ratsbetabinomial.inits`. We will run through a short demonstration using jags from the command line.

To start JAGs you just need to type:

```
jags
```

at a shell prompt and jags should start running. The jags prompt is `.` (a dot)

1.3.1 Load the model

The first step is to load the model, which we can do by typing:

```
model in betabinomial.jags
```

This is where we have a lot of potential for trouble with model syntax errors and unfriendly error messages. Make sure you have saved the sample file in the working directory. You can type `pwd` to check you are in the correct directory. This is your opportunity to fix major script errors.

1.4 Load data

You have been supplied data in plain text files. To load the rats data you need to enter:

```
data in rats.dump
```

The data are in what's known as R dump format. In other words, with your own data you need to load data into R and use the dump command to generate data of the correct format. You can check, but the dump file essentially has the format:

```
y <- c(0,0,2, etc. )  
n <- c(20, 20, etc.)  
N <- 71
```

If you think of python, this is equivalent to storing a script file containing your data as `y=[0,0,2 etc.]`. The main difference with R is that you need to use `c(...)` to enclose the data.

1.4.1 Compile the model

The next step is where some byte code containing our model and our data is generated.

```
compile,nchains(1)
```

This is the stage where we find out about a load more errors in our model code (for example if we said that there were $N = 70$ data points, but in fact there were 71 JAGs would complain that something was wrong.

There is an important general point at this stage. We can decide how many chains we wish to run. For demonstration purposes we shall compile just one. But there is an argument, as this is a simulation technique, that we should compile a number of chains. If, in the next step, we use a different starting point for each chain, we should be very happy if all chains end up giving us the same answer.

1.4.2 Load initial values

Usually, it is necessary to specify a starting point for the simulation. Again, we have provided a file `betabinom.inits` containing some initial values for each parameter. These also have to be in R dump format. This file looks like:

```
a <- 1  
b <- 1
```

as explained earlier.

The command to load these initial values into our first chain is given as:

```
parameters in betabinom.inits,chain(1)
```

If we have other chains, we specify another file name and indicate that it is the 2nd or 3rd chain.

1.4.3 Initialize

If all has been going well so far, we can initialize the model. If there are any parameters without suggested starting values, JAGs will now try and guess some.

```
initialize
```

1.4.4 Burn In

Because we don't know whether our initial starting values were sensible (or because we made sure they weren't), we need to allow the simulation algorithm to find its way to a sensible position in terms of parameter values. Hence it is conventional to discard the first few samples. Here we shall have a "burn-in" of 4000.

```
update 4000
```

1.4.5 Monitor and simulate

We're now ready to start the simulation proper. We need to tell JAGS which parameters we are interested in, and set a monitor on each of those. We will also collect information on the deviance. `monitor` tells jags to monitor a particular model parameter. These must match the names of parameters in our model specification file (`binomial.bugs`)

```
monitor epsilon,thin(1)
monitor a,thin(1)
monitor b,thin(1)
monitor pi,thin(1)
monitor deviance,thin(1)
```

The idea of the `,thin(1)` argument (which is optional) is that we might want to keep a proportion of samples. `Thin=1` means we keep all observations, `thin=10` means we keep one observation in 10.

So, with the monitors set, we can run the simulation for a further number of iterations.

```
update 5000
```

1.4.6 Extract results

There is a well established set of routines for posterior analysis, called coda (convergence output and diagnostic analysis). There are other methods beyond those included here. So it is convenient to extract our data in a format ready for analysis by coda.

You can just type `coda *` to throw all output into one coda file `CODAIindex.txt` and `CODAchain1.txt`.. Personally, I find it easier to keep different parameter posteriors in different files. The optional argument `stem()` to the jags command tells it to use that filename.

```
coda a,stem(a)
coda b,stem(b)
coda pi,stem(pi)
coda deviance,stem(deviancebetabin)
```

1.5 Posterior results

Now that we have text files in our working directory full of posterior simulations, all we need to do is summarise them. It turns out that there is an **R** package called coda which contains a number of these standard features for examining the output and checking for convergence. Rather conveniently, it has a menu interface.

Start R (type `R` at a shell), then issue the command:

```
codamenu()
```

This should give you three options, but we are only interested in
Option 1, Read BUGS output files

You are then asked to specify the index file (this will have “index” in its name, so type something like:

```
aindex.txt
```

You should then get asked for output files. You need to type the name of the corresponding output file (which has “chain” in its name):

```
achain1.txt
```

Or, if you have run multiple chains

```
achain1.txt  
achain2.txt  
chain3.txt
```

To end the input routine, you just need to enter a blank line. You should then be offered the main menu, which will include:

```
1 output analysis  
2 Diagnostics.
```

Have a look at the output analysis first, i.e., enter 1.

You should find that when you examine the summary statistics you get a posterior mean for a in the region of 2.4 and a posterior mean for b in the region of 14.5

You can return to the main menu by entering (4). A lot of research work continues in the area of model fit diagnostics, but there are some established criteria to help judge whether a posterior sample has indeed come from a convergent chain. For example, try plotting the autocorrelations (menu item 5 on the diagnostics menu) and see what you think. If you are curious about all the methods offered here, enter `help.start()` at the R prompt, and look for the link to the coda package. The R help system is very good in giving you full references to the literature used.

1.6 Posterior credible intervals

We’ve decided not to spend too much time on R programming, so you have been given an R script (`catplot.R`) which will produce caterpillar plots.

If you put this file in your working directory and enter:

```
source("catplot.R")
```

you should have access to a function called `catplot`.

The only difficulty is that you have to read the coda files in manually, calculate the summary manually and extract the part of the summary that gives you the posterior quantiles. Enter the following commands in R to read the data in (`read.coda()`), form the summary (`summary()`), extract the quantiles (`[[2]]`) and pass the summary to the `catplot` function.

```
pi <- read.coda("pichain1.txt", "piindex.txt")
posteriorsummary <- summary(pi)[[2]]
source("catplot.R")
catplot(posteriorsummary)
```

This gives us the posterior 95% credible interval for each π_i .

It's also interesting to plot this credible interval against the maximum likelihood estimate. If we load the data `rats.csv`, we can calculate the mle for each individual sample as $\frac{y_i}{n_i}$ and pass this as an argument to the `catplot()` function.

```
rats <- read.csv("rats.csv")
mle <- rats$y/rats$n
catplot(posteriorsummary, xd=mle, xl="mle")
```

In order to appreciate what we are doing, we can add a diagonal line for $x = y$ which tells us when the sample maximum likelihood estimate and posterior maximum likelihood are in agreement. We can then add a vertical green line to tell us where the population maximum likelihood estimate $\frac{\sum x_i}{\sum n_i}$ lies.

```
abline(0,1)
abline(v=sum(rats$y)/sum(rats$n), col = "green")
```

You should satisfy yourself that for samples which have lower counts of y_i than the “population average”, the posterior estimate has been moved upwards, and in samples which have higher counts of y_i than the “population average” the posterior has been moved downwards. The idea behind this shrinkage is that we can balance “evidence” from the population with “evidence” from the sample in a meaningful way.

The plot is rather crowded, but you can add the sample sizes:

```
catplot(posteriorsummary)
text(c(1:71), mle+0.05, rats$n, col = "red", cex=0.5)
```

You should see for example that the two rightmost posterior 95% credible intervals are quite wide, and that the sample size is relatively small. The third posterior summary from the right is slightly narrower, and is based on a larger sample size.

1.7 That $p(a, b) \propto (a, b)^{-5/2}$ prior

We briefly mentioned some of the problems on Monday in the choice of prior, and now we are using a simulation technique where we don't have this prior.

If we load the posterior values of a and b into R (code given below), we can plot the joint density of our simulation. We need to load a library called MASS which has a 2d density function (`kde2d`), and we can then plot a contour plot.

```

a <- read.coda("achain1.txt", "aindex.txt")
b <- read.coda("bchain1.txt", "bindex.txt")
pi <- read.coda("pichain1.txt", "piindex.txt")
require(MASS)
denstrtransformed <- kde2d(log(a/b), log(a+b))
contour(denstrtransformed)

```

Recall that the x axis of this plot is the prior mean, and the y axis is the prior sample size.

- You can change the specification of your prior for a and b in your JAGs model, and refit the model. You should find that across a wide range of sensible values, the x axis in this plot doesn't alter much. However, if you have priors that allow for large values of a and b , you will find that more density is given to larger values of $\log(a + b)$ on the y axis.

This is a bit of a worry. We are therefore going to try studying this problem from another perspective.

2 A generalised linear model formulation

There is an entirely different but very equivalent approach to the same problem. Rather than use a conjugate prior for π we shall use a generalised linear model, and shall deal with individual uncertainty by including a suitable term in the linear predictor. Thus we shall model:

$$\pi_i = \left(\frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \right)$$

and we have the linear predictor η_i given by:

$$\eta_i = \beta_0 + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$.

So, our key assumption of exchangeability now applies to ϵ_i . If we had additional data on the experiment that was relevant, it can be added in the standard way:

$$\eta_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

using all the tools we have in linear regression (multiple regression, interactions etc).

This model is specified in JAGs as follows:

```

model{
  for (i in 1:N){
    y[i] ~ dbin(pi[i], n[i])
    pi[i] <- exp(b0+epsilon[i])/(1+exp(b0+epsilon[i]))
  }
}

```

```

    epsilon[i] ~ dnorm(gamma, tau)
  }

gamma <- 0
b0 ~ dnorm(2, 0.0001)
tau ~ dgamma(0.01, 0.01)
}

```

You might like to note:

- $y[i] \sim \text{dbin}(\pi[i], n[i])$ is telling us that each data point $y_i \sim \text{Binomial}(\pi_i, n_i)$
 - Each $\pi = \text{invlogit}(\eta_i)$, i.e. $\pi_i = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}$ where $\eta_i = \beta_0 + \epsilon_i$
- We have some additional priors outside the data loop, so the “intercept” $\beta_0 \sim \text{Normal}(2, 10000)$ **One big thing to note. The JAGs family parameterise the Normal distribution by means of a precision**, i.e.

$$x \sim \text{Normal}(\mu, \tau)$$

where $\tau = \frac{1}{\sigma^2}$

As before, we need to load the model in `binomial.jags`, load the data, compile, load a modified initial value file `rats.inits`, initialize and then burnin.

```

model in binomial.jags
data in rats.dump
compile
parameters in rats.inits
initialize
update 5000

```

We now have a model set up and ready to run. The only thing to note is that we need to monitor different model parameters, and form coda files from these.

```

monitor epsilon
monitor b0
monitor tau
monitor deviance
update(5000)
coda b0,stem(b0)
coda tau,stem(tau)
coda epsilon,stem(epsilon)
coda deviance,stem(deviance)

```

If you go into R and use `codamenu()` on the simulated `b0` posterior, you should find that when you examine the summary statistics you get a posterior mean in the region of -1.95 . If we transform this into a proportion using:

$$\frac{\exp(-1.95)}{1 + \exp(-1.95)} = 0.124$$

which sounds like a value comparable to the simple m.l.e. This is our posterior estimate of the “population level” proportion.

The posterior for ϵ_i is now telling us something about how the proportion varies in different experiments.

3 Exercise

- Write all the above commands in a script file (save all your commands from `model` in to coda `deviance` in a text file called something like `jagsscript.txt`).
- Adjust this script so that you set a monitor for `pi`, and produce a coda output for `pi`
- Run this script by typing `>jags jagsscript.txt` at the bash prompt
- Use R to produce a caterpillar plot for pi_i

4 Optional: running jags from R

If you are comfortable with R, you can run `jags` from R. You need to load the data into a data frame called `rats`, and then

```
rats <- read.csv("rats.csv")
library(rjags)
model1 <- jags.model("binomial.jags", list(y=rats$y, n=rats$n, N=71),
  inits=list(tau=1))
update(model1, n.iter=4000, progress.bar="gui")
posterior1 <- coda.samples(model1, c("b0", "epsilon", "tau"),
  n.iter=1000, progress.bar="gui")
summary(posterior1)
posterior2 <- coda.samples(model1, c("b0"), n.iter=1000,
  progress.bar="gui")
plot(posterior2)
```

This has the advantage that you are working in an environment that can process and prepare the data for JAGS, and can summarise the posterior.