A Basic Example of ANOVA in JAGS

Joel S Steele

The purpose

This demonstration is intended to show how a simple one-way ANOVA can be coded and run in the JAGS framework. This is by no means an exhaustive example, but rather an introduction to modeling with JAGS for a rather common analysis.

The tools

To begin we will need JAGS installed on our computer. This is not part of the R environment, but rather a separate program that can be downloaded and installed from http://mcmc-jags.sourceforge.net/. Once JAGS is installed an ready you can use any number of interfaces from within R. For this demonstration we will stick with the `rjags` package. Additionally, we will need the help of `lattice` for plotting, along with `knitr` and `kableExtra` to format our output.

```
library(rjags)
library(lattice)
library(knitr)
library(kableExtra)
# for nice display of inline code
opts_chunk$set(echo=TRUE, comment='')
# for nicely formatted tables
options(knitr.table.format = "latex")
```

The Data

Since this is a demonstrated example, we will be creating data directly. Most often data would be read into the R session from an external file, but here it is easiest to simply specify some group observations.

```
# generate data
mdat = list('grp'=gl(3,4), 'ngrp'=3, 'N'=12,
            'y'=c(2,1,3,2, 2,6,2,2, 6,10,7,5))
# quick visualization of the data
disp.mdat = matrix(mdat$y,ncol=mdat$ngrp,
                   dimnames = list(paste('obs',1:(mdat$N/mdat$ngrp),sep=' '),
                                   paste('group',1:mdat$ngrp,sep=' ')))
kable(disp.mdat,caption='Generated Data',booktabs=TRUE) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 1: Generated Data

	group 1	group 2	group 3
obs 1	2	2	6
obs 2	1	6	10
obs 3	3	2	7
obs 4	2	2	5

Modeling via JAGS

With the data now specified and available, we move on to the specification of the one-way ANOVA model in the JAGS language. Note that this is almost exactly how the same model would be specified in most BUGS based languages. In this example it is exact. Difference between JAGS and BUGS are few and far between. Please read the JAGS manual for further discussion about specific differences.

```
# specify the likelihood model
mt = '
model{
  for(i in 1:N){
    y[i] ~ dnorm(mu[i], err.prec)
    mu[i] = alpha + beta[grp[i]]
  }
  err.prec ~ dgamma(1.0E-3,1.0E-3)
  rse = pow(err.prec,-.5)
  alpha ~ dnorm(0,1.0E-3)
  beta[1] = 0
  for(i in 2:ngrp){
    beta[i] ~ dnorm(0,1.0E-3)
  }
}'
# compile the model in JAGS
ti = textConnection(mt)
cm = jags.model(ti, data=mdat, # model specification and data.
                n.chains=2, # two random walkers
                # initial values, one set per chain
                inits=list(
                  list('alpha'=0,'beta'=c(NA,.5,.5)),
                  list('alpha'=2,'beta'=c(NA,1,5))),
                # burn-in/adaptation steps before we trust
                n.adapt = 1000, quiet=TRUE)
close(ti) # clean up
# run the chains longer
update(cm,5000)
```

Monitoring the posteriors

The point of MCMC is to generate a reasonable sample from the posterior distribution. This, according to Bayes' formula, is the product of the likelihood and the priors. In short, how we think the data relate, tempered by how likely we feel the parameters that represent those relations are to occur.

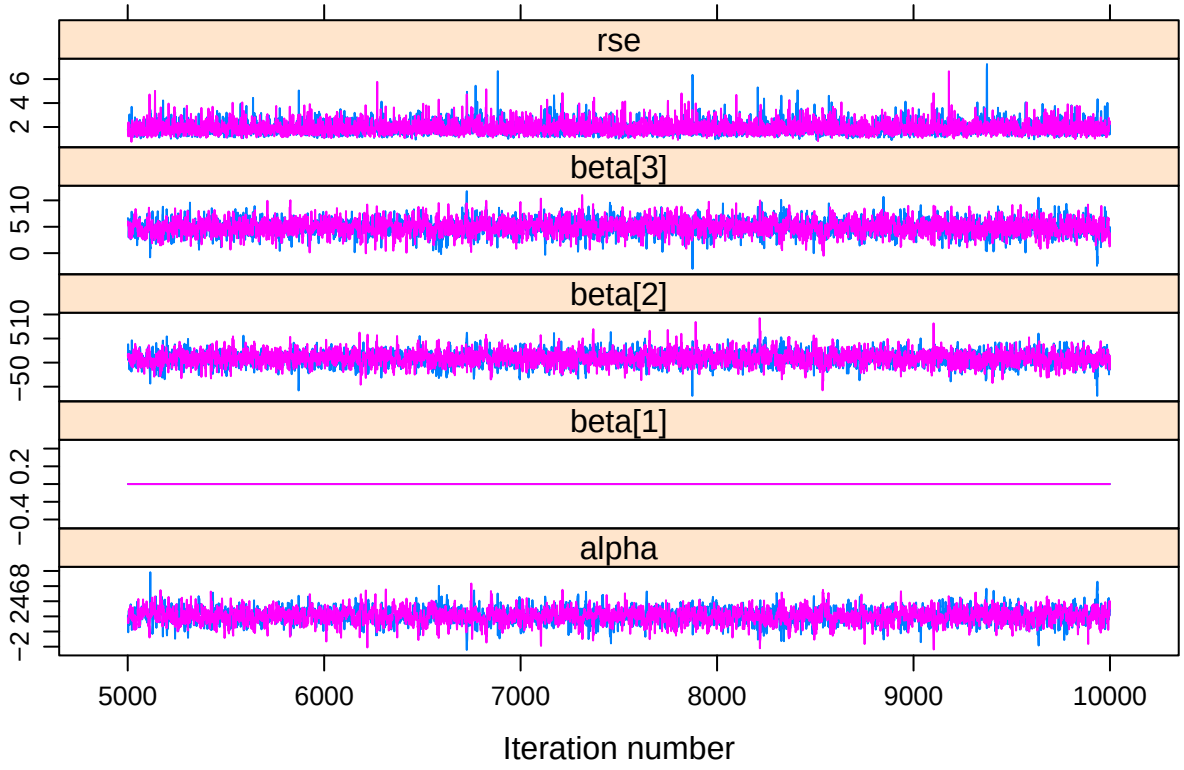
This implies that we need to keep track of the posterior samples that are created through the MCMC algorithm. We accomplish this be setting up monitors for each parameter of interest, as listed in the model specification above, and running the random walk chains (the MCMC simulations) for a number of steps. These are the samples that we will be computing our summary statistics from for reporting. All of this comes from the `coda` package which is loaded up by default with `rjags`.

```
# track the progress of our walkers relative to our parameters of interest
pos = coda.samples(cm, # our compiled model
                   # parameters of interest to monitor
                   variable.names=c('alpha','beta','rse'),
                   n.iter=5000) # steps to follow
```

Visualization

Diagnostics are not the same as with strictly likelihood based estimations. In the world of MCMC we are summarizing over plausible random samples based on the product of the likelihood and the priors, there is no analytic (or numeric) evaluation of the maximum of the likelihood function, rather enough samples are drawn to mimic a sampling distribution of sorts for each monitored parameter. What this means in practice is that model diagnostics are more of an art than a science. One very popular and useful approach is to visualize the posterior chains and look for the classic *hairy caterpillar* which implies that the samples are stable. Notice here that we are summarizing the samples from 5,001 to 10,000. Different windows can be used for diagnostics and inference.

```
xyplot(window(pos,start=5001,stop=10000))
```



Diagnostics

More formal assessments of convergence come from a number of researchers.

Here we examine the Geweke diagnostic test which produces a z-score difference between the beginning and ending of the chain for a monitored parameter. If the distribution is stationary, these segments should not be significantly different from each other.

```
geweke.diag(window(pos,start=5001,stop=10000))
```

```
[[1]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  alpha beta[1] beta[2] beta[3]     rse
 0.9461     NaN -1.3020 -1.2052 -1.0298
```

```
[[2]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  alpha beta[1] beta[2] beta[3]     rse
 1.1473     NaN -1.4835 -0.8017 -0.9331
```

Reporting

```
# nice format please
kable(summary(window(pos,start=5001,stop=10000))$statistics,
      digits=3, booktabs=TRUE,
      caption='Summary of Posteriors for model parameters') %>%
  kable_styling(latex_options =  "hold_position") %>%
  footnote(general='Estimates based on final 5,000 samples.')
```

Table 2: Summary of Posteriors for model parameters

	Mean	SD	Naive SE	Time-series SE
alpha	2.033	0.964	0.010	0.021
beta[1]	0.000	0.000	0.000	0.000
beta[2]	0.952	1.379	0.014	0.026
beta[3]	4.959	1.384	0.014	0.027
rse	1.924	0.525	0.005	0.007

Note:
Estimates based on final 5,000 samples.

```
# Hypothesis testing of parameters?
kable(summary(window(pos,start=5001,stop=10000))$quantiles,
      digits=3, booktabs=TRUE,
      caption='Credibility intervals from Posteriors of model parameters') %>%
  kable_styling(latex_options = 'hold_position') %>%
  footnote(general='Estimates based on final 5,000 samples.')
```

Table 3: Credibility intervals from Posteriors of model parameters

	2.5%	25%	50%	75%	97.5%
alpha	0.130	1.428	2.023	2.650	3.904
beta[1]	0.000	0.000	0.000	0.000	0.000
beta[2]	-1.799	0.108	0.962	1.802	3.676
beta[3]	2.170	4.105	4.953	5.834	7.708
rse	1.214	1.562	1.822	2.173	3.180

Note:
Estimates based on final 5,000 samples.

Comparsion(s)

```
# How does this compare to OLS?
kable(summary(lm(y~grp,mdat))$coefficients,
      digits=c(3,3,3,4), booktabs=TRUE,
      caption='OLS regression results') %>%
  kable_styling(latex_options = 'hold_position')
```

Table 4: OLS regression results

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2	0.882	2.268	0.0495
grp2	1	1.247	0.802	0.4433
grp3	5	1.247	4.009	0.0031