Hey y'all!

This tutorial aims to expand what we talked about in Part 1 about linear regression. Today we'll add in categorical variables and model selection. As before, I'll show you how to run it in both JAGS and NIMBLE and I'll go over how to graph the results with credible intervals!

I'm assuming for now that everyone has downloaded JAGS and NIMBLE and has them setup on their computers. Before you use NIMBLE make sure R, and Rtools or Xcode are updated on your computer (otherwise a weird "shared library" error can come up). JAGS is downloadable here: https://sourceforge.net/projects/mcmc-jags/ and NIMBLE can be found here: https://r-nimble.org/download.

Please send any questions or suggestions to heather.e.gaya@gmail.com or find me on twitter @doofgradstudent

Contents

1 A Fake Scenario

As we saw in Part 1, Joe has gone out in the world and collected 50 frogs. He recorded each frog's age (in days), weight (in g), left back leg length (cm), the distance the animal was from the road (m) and what species the frog is (A or B). His data looks like this (but with 50 rows):

	Frog	Age	Weight	Leg	Dist	Species
1	1	191	15.44	4.90	57.61	A
2	2	184	21.45	3.44	92.08	A
3	3	243	21.39	4.37	54.87	A
4	4	770	114.28	3.42	189.96	A
5	5	614	64.52	3.73	38.64	A
6	6	771	79.07	3.18	13.33	A

Previously, Joe was only interested in how weight was related to age, leg length and distance from the road, but now he wants to model the two species separately.

When we only used continuous variables, we could just multiply betas (β) by the variable itself to model the relationship. But "Species" isn't informative numerically - there's no order to Species A vs Species B - so we have to think about this a different way.

There are many options here for the actual equation, depending on what we think the relationship is with species - do we think the intercept is different? Maybe one frog grows faster than the other, so we would expect a difference in the relationship with age?

Before, we had:

$$E(weight) = \beta_0 + \beta_1 A + \beta_2 L + \beta_3 D$$
$$Actualweight \sim Normal(\mu = E(weight), \sigma = sd)$$

where A was age, L was leg length and D was distance to road.

Let's imagine a model where just the intercepts are different. Maybe this means we think the average weight of species B is always going to be heavier, but it will still have the same relationships with the other variables.

Conceptually, we might think of our expected weight equation as: E(weight) ¡- interceptA*(1 if species A, 0 if species B) + interceptB*(1 if species B, 0 if species A) + age X something + leg X something2 + distance X something3.age*something + leg*something2 + distance*something3

In more mathy format,

$$E(weight) = \begin{cases} \text{species = A} & \beta_0^A + \beta_1 A + \beta_2 L + \beta_3 D \\ \text{species = B} & \beta_0^B + \beta_1 A + \beta_2 L + \beta_3 D \end{cases}$$
$$Actualweight \sim Normal(\mu = E(weight), \sigma = sd)$$

However in JAGS (or NIMBLE) we can't use this "ifelse" type statement. So we have to resort to a trick called "nested indexing".

2 A Quick Note on Nested Indexing

Nested indexing can be a little confusing at first glance, but it saves a ton of time and headache once you get used to the idea! We're used to thinking of variables as either one value (think, β_1) or a vector of lots of values (e.g. leg values of all frogs in our dataset) but variables can be all sorts of dimensions.

Let's say we have a variable s, a vector of length 50 that represents if the frog is species A or species B. 1's are species A and 2's are species B. Let's also say we have two intercept values, one for species A and the second for species B.

```
s <- as.numeric(Frogs$Species)
fake.intercepts <- c(2, 5)
```

Normally in R, if we want the 5th individual's species, we would type s[5] and get out "1". And if we want the intercept for species A, we would type fake.intercepts[1] and get out "2". So we can conveniently put these together to say "hey, what's the intercept value for individual 5?". This is the joy of nested indexing.

```
fake.intercepts[s[5]]  #this equals 2
```

3 Writing Model 1 JAGS

As before, we can save the model as a textstring in R to send to JAGS later. First, we start with the meat of the model - the equation we had up above (but now with nested indexing!). Also remember that JAGS requires the normal distribution be defined with mean and precision instead of standard deviation.

```
for (i in 1:n.frogs){
  meanweight[i] <- beta0[s[i]] + age[i]*beta1 + leg[i]*beta2 + distance[i]
   *beta3
  weight[i] ~ dnorm(meanweight[i], prec)
}
```

Now let's get some priors in there and convert between precision and standard deviation

```
beta0[1] ~ dunif(-50,50)
beta0[2] ~ dunif(-50,50)
beta1 ~ dunif(-50,50)
beta2 ~ dunif(-50,50)
beta3 ~ dunif(-50,50)

prec <- 1/(sd *sd)
sd ~ dunif(0.0001, 100)
```

And finally we stick it all together into one model!

```
modelstring.Frogs1 = "
  model
{
for (i in 1:n.frogs){
  meanweight[i] <- beta0[s[i]] + age[i]*beta1 + leg[i]*beta2 + distance[i]
   *beta3
    weight[i] ~ dnorm(meanweight[i], prec)
```

```
}

beta0[1] ~ dunif(-50,50)
beta0[2] ~ dunif(-50,50)
beta1 ~ dunif(-50,50)
beta2 ~ dunif(-50,50)
beta3 ~ dunif(-50,50)

prec <- 1/(sd *sd)
sd ~ dunif(0.0001, 100)
}
"
```

Time to send the model to JAGS. First we give the model parameters to monitor:

```
params <- c("beta0", "beta1", "beta2", "beta3","sd")
```

And give JAGS the data we have on the frogs. We also need to add the new variable "s" that we created up above.

```
data <- list(weight = Frogs$Weight, distance = Frogs$Dist,
             age = Frogs$Age, leg = Frogs$Leg,
             n.frogs = 50, s = s)
```

Next initial values - TBH I don't really use initial values most of the time unless I have to. JAGS will try to make its own initial values if you don't make them and I'm a lazy programmer so I like that! The big caveat is that sometimes JAGS picks terrible initial values and then throws weird errors, so that's something to be prepared for. But as long as I'm telling people on the internet what to do, I might as well be a bad influence every now and then :)

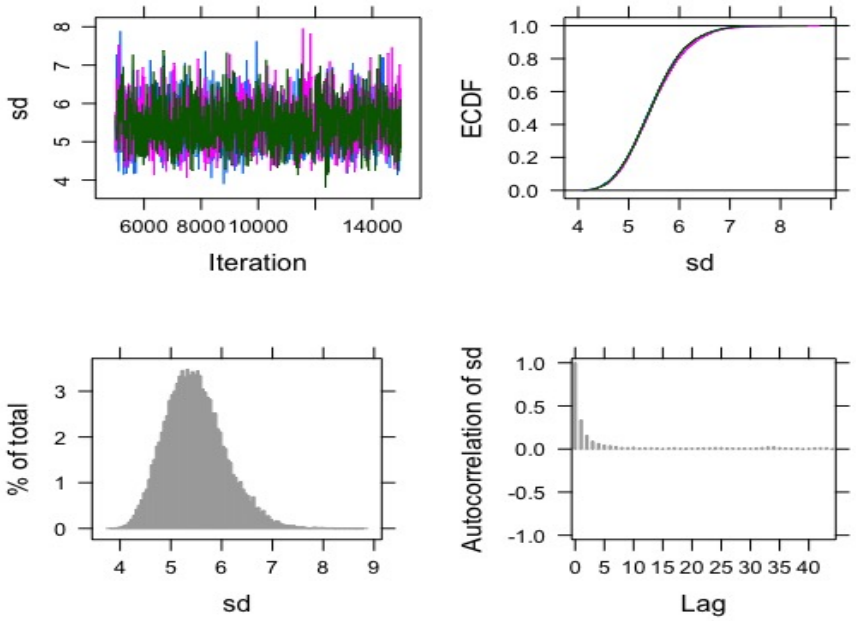
Finally we run the model and check the summary statistics

```
Frog.mod1 <- run.jags(model = modelstring.Frogs1,
                      monitor = params,
                      data = data, n.chains = 3,
                      sample = 10000, method = "parallel")
summary(Frog.mod1)
```

	Lower95	Median	Upper95	Mean	psrf
beta0[1]	-18.24	-7.63	2.77	-7.69	1.01
beta0[2]	-19.22	-8.08	1.91	-8.15	1.01
beta1	0.12	0.12	0.13	0.12	1.00
beta2	-4.35	-2.08	0.19	-2.06	1.02
beta3	0.14	0.16	0.19	0.17	1.00
sd	4.37	5.44	6.65	5.48	1.00

Looking good, but let's do a visual inspection just for good measure.

```
plot(Frog.mod1)
```



Obviously you should look at all the plots, not just this one, but they all look fairly similar. Chains are mixing well and the autocorrelation drops off pretty quickly. Looks like our model has converged!

4 Model 1 in NIMBLE

Running the model in NIMBLE is almost the same as running it in JAGS, except we can skip the precision part.

```
nimbleFrogs1 <-
  nimbleCode({ #don't forget this part

for (i in 1:n.frogs){
  meanweight[i] <- beta0[s[i]] + age[i]*beta1 + leg[i]*beta2 + distance[i]
   *beta3
  weight[i] ~ dnorm(meanweight[i], sd = sd )
}

beta0[1] ~ dunif(-50,50)
beta0[2] ~ dunif(-50,50)
beta1 ~ dunif(-50,50)
beta2 ~ dunif(-50,50)
beta3 ~ dunif(-50,50)

sd ~ dunif(0.0001, 100)
})
```

As always, we need to define our params, data, constants and inits arguments. Don't forget the new variable "s" is now a new constant.

```
params <- c("beta0", "beta1", "beta2", "beta3","sd")
```

```
data <- list(weight = Frogs$Weight, distance = Frogs$Dist,
             age = Frogs$Age, leg = Frogs$Leg)
constants <- list(n.frogs = 50, s = s)
inits <- list(beta0 = runif(2, -10, 10), beta1 = runif(1,-10,10), beta2 =
   runif(1,-10,10), beta3 = runif(1,-10, 10), sd = runif(1, .001, 100))
```

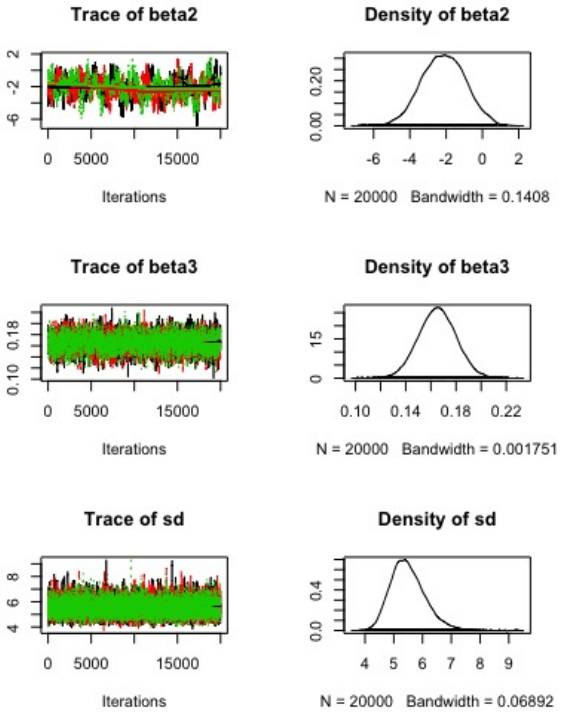
Time for our 7 step process to run the model. Notice that this time we'll prepare for using model selection by enabling WAIC. We'll want to check convergence as normal

```
prepfrogs <- nimbleModel(code = nimbleFrogs1, constants = constants,
                         data = data, inits = inits)
prepfrogs$initializeInfo() #everything is good to go!
mcmcfrogs <- configureMCMC(prepfrogs, monitors = params, print = T )
frogsMCMC <- buildMCMC(mcmcfrogs, enableWAIC = TRUE) # a change here to
   allow us to use WAIC
Cmodel <- compileNimble(prepfrogs) #compiling the model itself in C++;
Compfrogs <- compileNimble(frogsMCMC, project = prepfrogs) # compile the
   samplers next
Frog.mod.nimble <- runMCMC(Compfrogs, niter = 30000, thin = 1, nchains =3,
    nburnin = 10000, samplesAsCodaMCMC = TRUE, WAIC = TRUE)
summary(Frog.mod.nimble$samples)
gelman.diag(Frog.mod.nimble$samples)
```

	Mean	SD
beta0[1]	-7.68	5.51
beta0[2]	-8.11	5.49
beta1	0.12	0.00
beta2	-2.08	1.22
beta3	0.17	0.02
sd	5.51	0.60

	Point est.	Upper C.I.
beta0[1]	1.03	1.09
beta0[2]	1.02	1.08
beta1	1.00	1.01
beta2	1.02	1.07
beta3	1.00	1.00
sd	1.00	1.00

Happy convergence! But per always, we should plot our chains just in case.



Our graph for beta2 looks maybe a little wiggly, but not enough to worry about.

5 Model Selection: DIC and WAIC

So now we have our new model... but is it really any better than any other model we might try? This is where model selection comes in. There's no one way that's agreed upon for Bayesian model selection. DIC is convenient and already built into JAGS but there's not theoretical basis for why it works. WAIC is a more robust option than DIC, but requires slightly more calculation (unless you run NIMBLE, which has WAIC built in). Both options give fairly similar answers and are appropriate when the model is not a mixture model, so we're good to use them for our simple linear regressions.

But what are they? Gelman, Hwang, and Vehtari 2013 give a great explanation of these methods but I will paraphrase in a hopefully helpful way. First, let's talk about DIC since most people (currently) seem to use this if they do model selection for linear regression.

DIC is a "somewhat Bayesian version of AIC". The goal of DIC is similar to that of AIC - estimate the likelihood of that model and correct for the number of parameters. The model with the lowest DIC score is considered the "best" model OF THE MODELS WE TESTED. This is not a test for the "true model" only a test for the "best of the options available". DIC is calculated via:

$$DIC = -2log(p(y|\theta_{bayes})) + 2p_{DIC}$$

If we break this apart, we can see the similarities with AIC. Firstly, $p(y|\theta_{bayes})$ simply refers to the posterior estimate of θ, AKA the fit of the model. How well does the model fit the data we gave it? Intuitively we have a sense that a better fit to the data makes a better

model, so this part of the formula is logical. We then take the log of it and multiply it by -2 to make it similar to AIC. The second half of the equation tries to correct for potential overfitting of data. In general, the more parameters we add, the better the model will seem to "fit" the data, but that doesn't mean it's going to be any good at predicting anything. We want to balance the fit with the data and the ability to predict points not in our dataset. Similar to the k in AIC, $2p_{DIC}$ refers to the effective number of parameters in the model. The more parameters we have, the higher the value for this half of the equation and the higher the value of DIC.

To extract DIC from our JAGS run, we simply run:

```
Dic_1 <- extract(Frog.mod1, what = "dic")
```

We find that the mean deviance (the value of interest) is 310.5. Cool! But useless unless we run more models and compare values.

WAIC is (shocker) also fairly similar but attempts to approximate cross-validation to help determine model fit. WAIC uses a similar measure of goodness-of-fit as DIC, (log of the average posterior likelihood for each data point), but uses the posterior variance in log-likelihood, with larger variances resulting in harsher penalties.

If we wanted to calculate WAIC for the model we just ran in JAGS, we would have to adjust the model a little bit to calculate the deviance of each point and then monitor that parameter:

```
modelstring.Frogs1 = "
  model
{
for (i in 1:n.frogs){
  meanweight[i] <- beta0[s[i]] + age[i]*beta1 + leg[i]*beta2 + distance[i]
   *beta3
    weight[i] ~ dnorm(meanweight[i], prec)
    loglik[i] <- logdensity.norm(weight[i], meanweight[i], prec) #for WAIC
     calculation
}

beta0[1] ~ dunif(-50,50)
beta0[2] ~ dunif(-50,50)
beta1 ~ dunif(-50,50)
beta2 ~ dunif(-50,50)
beta3 ~ dunif(-50,50)

prec <- 1/(sd *sd)
sd ~ dunif(0.0001, 100)
}
"

params <- c("beta0", "beta1", "beta2", "beta3","sd", "loglik")
data <- list(weight = Frogs$Weight, distance = Frogs$Dist,
             age = Frogs$Age, leg = Frogs$Leg,
             n.frogs = 50, s = s)
Frog.mod1 <- run.jags(model = modelstring.Frogs1,
                      monitor = params,
                      data = data, n.chains = 3,
                      sample = 10000, method = "parallel")
```

We could then calculate WAIC by hand via:

```
like <- rbind(as.matrix(Frog.mod1b$mcmc[1][,7:56]), as.matrix(Frog.mod1b$
    mcmc[2][,7:56]),as.matrix(Frog.mod1b$mcmc[3][,7:56]))
fbar <- colMeans(like) #mean log-likelihood
Pw <- sum(apply(like,2,var)) #mean variance in log-likelihood
WAIC<- -2*sum(fbar)+2*Pw
WAIC #336.52
```

Note that because JAGS already calculates DIC, there's no real reason to go through all this work. DIC will work just fine for our purposes! But if you want to know *how* WAIC works, this is probably the best way to learn.

If we want to calculate WAIC from NIMBLE, however, we simply have to enable it during the run commands and then extract it via:

```
Frog.mod.nimble$WAIC #326.54
```

We aren't going to get the same value from our hand calculation as from NIMBLE because of a slight difference in how NIMBLE calculates WAIC, but the difference is not very important. The main thing is that we compare WAIC values that have been calculated the same way - either by hand or via NIMBLE.

Of course, the output of DIC or WAIC is not particularly meaningful on its own. It is only meaningful in comparison to other DIC or WAIC values for models run with the same data. So let's run some other models!

6 Models 2 and 3 in JAGS

Returning to our models, let's test a model where one frog species grows faster than the other. Let's say we expect a difference in the relationship with age, but no difference in the intercept.

```
modelstring.Frogs2 = "
  model
{
for (i in 1:n.frogs){
  meanweight[i] <- beta0 + age[i]*beta1 + leg[i]*beta2 + distance[i]*beta3
    [s[i]]
  #note that the only thing that changes is the indexing of beta0 and
    beta3
  weight[i] ~ dnorm(meanweight[i], prec)
}

#but now beta0 is only one value and  beta3 can be two different values
beta0 ~ dunif(-50,50)
beta1 ~ dunif(-50,50)
beta2 ~ dunif(-50,50)
beta3[1] ~ dunif(-50,50)
beta3[2] ~ dunif(-50,50)


#I like to convert precision to standard deviation because it confuses me
    otherwise.
```

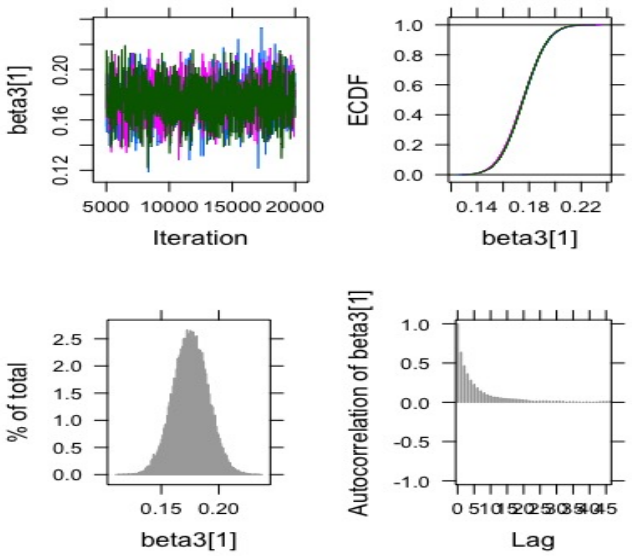
```
prec <- 1/(sd *sd)
sd ~ dunif(0.0001, 100)
}
"
```

Now let's run the model through JAGS and see what we get.

```
Frog.mod2 <- run.jags(model = modelstring.Frogs2,
                      monitor = params, inits = inits2,
                      data = data, n.chains = 3,
                      sample = 15000, method = "parallel")
Frog.mod2
```

	Lower95	Median	Upper95	Mean	psrf
beta0	-18.39	-8.58	1.89	-8.51	1.01
beta1	0.12	0.12	0.13	0.12	1.00
beta2	-4.19	-1.92	0.36	-1.93	1.01
beta3[1]	0.15	0.18	0.21	0.18	1.00
beta3[2]	0.12	0.15	0.18	0.15	1.00
sd	4.27	5.29	6.49	5.34	1.00

Let's plot it and check like always (Be sure to check all the plots, even though I only show 1!)



Looks like our model converged. Let's extract our DIC value and see how it compares to model 1.

```
Dic_2 <- extract(Frog.mod2, what = "dic")
Dic_2 #307.8
```

Seems that this model is slightly better (according to DIC) than model 1. Notice the difference isn't all that large, so the difference in fit isn't much better. Let's test one more model for funsies. Maybe Joe thinks about his frogs and again and thinks that maybe weight

is only related to age, but he still suspects there's a different equation (both intercept and slope) for both species.

```
modelstring.Frogs3 = "
  model
{
for (i in 1:n.frogs){
  meanweight[i] <- beta0[s[i]] + age[i]*beta1[s[i]]
  #got to reindex everything
  weight[i] ~ dnorm(meanweight[i], prec)
}

#adjust priors to match
beta0[1] ~ dunif(-50,50)
beta0[2] ~ dunif(-50,50)
beta1[1] ~ dunif(-50,50)
beta1[2] ~ dunif(-50,50)

prec <- 1/(sd *sd)
sd ~ dunif(0.0001, 100)
}
"
```

Per usual, we adjust the parameters to monitor, data and initial values, then send it all to JAGS.

```
params3 <- c("beta0", "beta1","sd")
data3 <- list(weight = Frogs$Weight,
              age = Frogs$Age, n.frogs = 50, s = s)
inits3 <- function(){list(beta0 = runif(2, -10, 10), beta1 = runif
   (2,-10,10))}
Frog.mod3 <- run.jags(model = modelstring.Frogs3,
                      monitor = params3, inits = inits3,
                      data = data3, n.chains = 3,
                      sample = 10000, method = "parallel")
Frog.mod3
plot(Frog.mod3)
```

Again, everything seems to have converged, so we can check the DIC value of this model as well. We'll stick everything in a table together so we can see how the 3 models compare to one another.

```
Dic_3 <- extract(Frog.mod3, what = "dic")
data.frame(Mods = 1:3, DICS = c(sum(Dic_1$deviance), sum(Dic_2$deviance),
   sum(Dic_3$deviance)))
```

	Mods	DIC
1	1	310.50
2	2	307.78
3	3	377.05

Okay, that model was much worse! But remember that DIC only tells you about models you've tested - in fact, the true model I used to create this data didn't separate species at

all! So it's important to think about all your assumptions and all the models that might make sense. Remember - all models are wrong, but some models are useful :)

7 Models 2 and 3 in NIMBLE

Running the output from NIMBLE would be redundant, but here's those same models as above but in NIMBLE. Notice that basically nothing changes - the main differences are just in the way the models are sent to their various programs and the use of sd instead of precision.

```
nimbleFrogs2 <-
  nimbleCode({ #don't forget this part
for (i in 1:n.frogs){
  meanweight[i] <- beta0 + age[i]*beta1 + leg[i]*beta2 + distance[i]*beta3
    [s[i]]
  #note that the only thing that changes is the indexing of beta0 and
    beta3
  weight[i] ~ dnorm(meanweight[i], sd = sd)
}

#but now beta0 is only one value and  beta3 can be two different values
beta0 ~ dunif(-50,50)
beta1 ~ dunif(-50,50)
beta2 ~ dunif(-50,50)
beta3[1] ~ dunif(-50,50)
beta3[2] ~ dunif(-50,50)

sd ~ dunif(0.0001, 100)
})
```

```
nimble.Frogs3 <-
  nimbleCode({
for (i in 1:n.frogs){
  meanweight[i] <- beta0[s[i]] + age[i]*beta1[s[i]]
  #got to reindex everything
  weight[i] ~ dnorm(meanweight[i], sd = sd)
}

#adjust priors to match
beta0[1] ~ dunif(-50,50)
beta0[2] ~ dunif(-50,50)
beta1[1] ~ dunif(-50,50)
beta1[2] ~ dunif(-50,50)

sd ~ dunif(0.0001, 100)
})
```

Once we have the models, we run them through NIMBLE as we did with model 1, making sure our monitors match the parameters we want out of our models.

Similar to our DIC table with JAGS, we can also check which of the 3 models appears to be the best choice of the options available.

```
data.frame(Mods = 1:3, WAIC = c(Frog.mod.nimble$WAIC, Frog.mod.nimble2$
    WAIC, Frog.mod.nimble3$WAIC))
```

	Mods	WAIC
1	1	326.54
2	2	325.14
3	3	382.14

As we expected, model 2 is the "best" model. Now let's see how to graph the results of that model.

8 Graphing the JAGS Output

I, for one, am fairly lazy. Especially when it comes to math. If a program can do the work for me, I'm happy to let it do it! So for graphing output where multiple variables have credible intervals, I like to have JAGS do the work for me.

We can add a few lines to our code to get some nice point estimates at equally spaced points along the line we want to graph. Let's make a graph showing the relationship between expected weight and distance to road.

First, we add some new variables into our model that represent the predicted weights for species A and species B at different distances along the road. We'll hold the other variables constant at their mean values.

```
modelstring.Frogs2graph = "
  model
{
for (i in 1:n.frogs){
  meanweight[i] <- beta0 + age[i]*beta1 + leg[i]*beta2 + distance[i]*beta3
    [s[i]]
  weight[i] ~ dnorm(meanweight[i], prec)
}
beta0 ~ dunif(-50,50)
beta1 ~ dunif(-50,50)
beta2 ~ dunif(-50,50)
beta3[1] ~ dunif(-50,50)
beta3[2] ~ dunif(-50,50)

prec <- 1/(sd *sd)
sd ~ dunif(0.0001, 100)

#Let's graph distances from 0 to 200 m from the road (reasonable, given
    our data ranges from 1.7 to 198)

for (k in 1:201 ){ #can't iterate starting at 0, so we'll just use k-1 in
    our equation and start at 1 instead

  graph_w1[k] <- beta0 + 392*beta1 + 4*beta2 + (k-1)*beta3[1] #species 1
  graph_w2[k] <- beta0 + 392*beta1 + 4*beta2 + (k-1)*beta3[2] #species 2
}
```

```
}
"
```

Now we'll want to monitor these new variables graphw1 and graphw2.

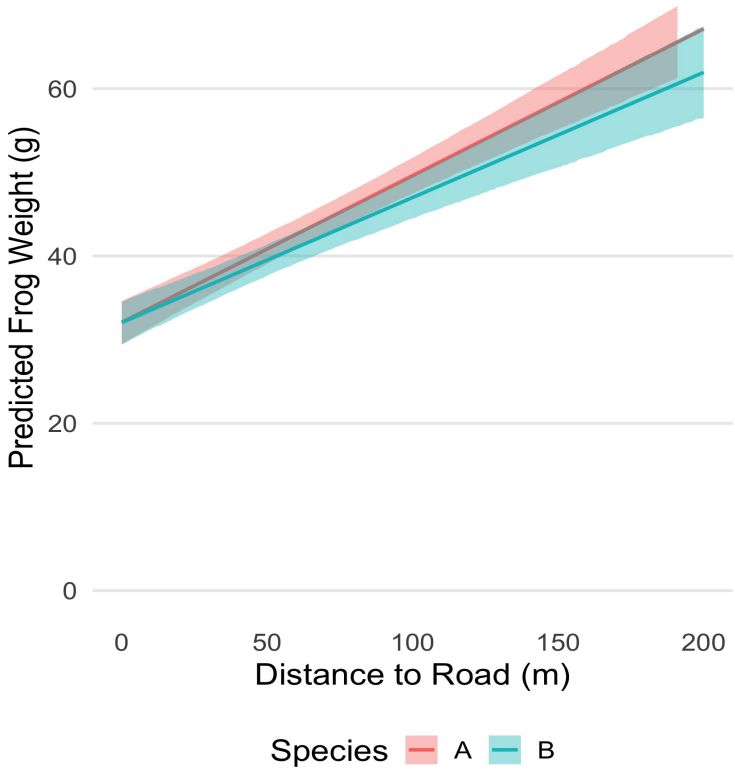
```
params2 <- c("beta0", "beta1", "beta2", "beta3","sd", "graph_w1", "graph_w2")
Frog.mod2graph <- run.jags(model = modelstring.Frogs2graph,
                      monitor = params2,
                      data = data, n.chains = 3,
                      sample = 15000, method = "parallel")
```

This will produce 402 more nodes, so I'm not going to look at my output this time. We already know the model should converge at this point since we haven't added any new stochastic information to the model, so we don't have to worry about convergence as long as we run the same number of chains as before. Instead, we extract the means and CI's from the output without looking at them and throw them into a nice dataframe for ggplot to deal with. (If you want the base R graph, it's available in the R code, but I'll skip it for now)

```
res <- summary(Frog.mod2graph)
graphme <- data.frame(low = res[7:408,1], upper  = res[7:408,3],
                      mean = res[7:408,4], Species = rep(c("A", "B"), each = 201),
                      dist = c(0:200, 0:200))
```

And then we throw it into ggplot! Obviously you can customize your graph however, but here's the basic idea.

```
library(ggplot2)
ggplot(data = graphme, aes(x = dist, group = Species, col = Species, fill = Species))+
  geom_smooth(aes(y = mean, ymin = low, ymax = upper), stat = "Identity")+
  labs(x = "Distance to Road (m)", y = "Predicted Frog Weight (g)")+
  ylim(0,70)+
  theme_minimal(base_size = 18)+
  theme(legend.position = "bottom",
        panel.grid.minor = element_blank(),
        panel.grid.major.x = element_blank(),
        strip.background =element_blank())
```



Tragically not the most interesting of graphs. But yay! Now you can graph output, perform model selection and write linear regression with categorical variables (hopefully). The same exact procedure can be done in NIMBLE - just extract the values from the mcmc chains and send them to ggplot!