Today's tutorial is on Occupancy Modeling! Occupancy modeling is awesome! And much more applicable to what I actually do in my real life than plain linear regression. We will start with single species occupancy modeling under a very simple scenario. As always, I will run the model in both JAGS and NIMBLE. I hope you'll find my methods useful but remember that there are always many ways to do things, so if you know a faster or better method please let me know!

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** **2**

**2 Conceptualizing the Model** **3**

**3 Writing the Model in JAGS** **3**

**4 Note: Priors When Transforming Data** **5**

**5 Running the Model in JAGS** **6**

**6 Graphing the Results** **7**

**7 Running in NIMBLE** **8**

# 1 A Fake Scenario

Let's say we have some birds. They're cool, we like them. Let's say they're Canada Warblers (or CAWA), because those are the things I actually get paid to study, despite my love of all things herpetology. They're really pretty birds that migrate from South America to the USA/Canada just to breed! (I stole this image off Wikipedia).

We go out and do point counts - this involves standing in one spot and listening/watching for birds and recording what you see/hear. In addition to counting birds, we also record things about the environment on the day we go there - namely environmental noise. We do repeated observations at each point. We have previously recorded the elevation of the sites where we do point counts.

For this example, let's say we are only interested in presence/absence. The actual number of birds does not matter. We also assume the presence/absence of birds does not change with time. Obviously these are big assumptions, but let's start simple.

Here's what our data looks like. In one table, we have the site information - the site number and the site elevation (in km). We have 40 sites in this dataset.

| | Site | Elevation |
|---|---|---|
| 1 | 1 | 0.75 |
| 2 | 2 | 0.71 |
| 3 | 3 | 0.35 |
| 4 | 4 | 1.35 |
| 5 | 5 | 0.32 |
| 6 | 6 | 0.84 |

In another file, we have the observations for each site. We record if the species was present/absent on each of our 6 visits to each site as well as the environmental noise during the point count.

| | X | Obs1 | Obs2 | Obs3 | Obs4 | Obs5 | Obs6 | noise1 | noise2 | noise3 | noise4 | noise5 | noise6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 15.02 | 9.61 | 13.41 | 13.49 | 6.09 | 3.13 |
| 2 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0.76 | 6.58 | 19.07 | 6.31 | 18.66 | 11.91 |
| 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 18.25 | 8.33 | 0.12 | 7.31 | 12.79 | 18.50 |
| 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 13.72 | 17.77 | 15.58 | 10.77 | 10.26 | 2.30 |
| 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 11.02 | 19.23 | 15.62 | 4.64 | 5.06 | 0.63 |
| 6 | 6 | 1 | 0 | 0 | 0 | 0 | 1 | 2.17 | 13.33 | 2.67 | 2.11 | 2.86 | 3.05 |

To get an estimate of raw occupancy, we can see how many sites we ever even saw a CAWA. However, just because we didn't see CAWA, doesn't mean CAWA aren't at that site. This is why we go to the trouble of modeling this stuff at all.

```
sum(rowSums(Birds[,2:7]) > 0) #17 sites
```

We saw birds at 17 sites, but let's use our model to estimate how many of them actually have CAWA.

# 2 Conceptualizing the Model

Let's think about the data we have - 1's and 0's. What process creates a 1? That's pretty easy - that means a bird was there and we detected it.

But what causes a 0? 0's can be from one of two processes - either the bird was there (the site was suitable for birds) and we missed it, or the bird wasn't there and so we saw nothing. So that means we have two separate processes here that are mixing together - yay mixture models!

Let's think about the process that allows a site to be suitable. For CAWA, we know they seek out higher elevation - possibly b/c of lower temps or competition - but sometimes the birds aren't there even when the site seems pretty suitable (to us). So this gives us the idea that there's some probability involved, maybe other environmental factors, but generally speaking CAWA are more likely to be found at higher elevation.

There are multiple ways to model this, but I like to think of it as, where $\psi$ refers to the probability a site is suitable:

$$\psi = \beta_0 + \beta_1 X$$
$$Occ \sim Bernouli(\psi)$$

What about the process of detecting the birds? If the bird is not there, the probability of seeing the bird is 0 (false positives can be a thing, but we'll ignore them for now). If the bird is there, there's some probability we see it. This could be constant across all sites, change with time, change with location, who knows! We know that noise in the environment probably lowers detection probability, so we'll start with a simple model with only one covariate. Since we're ignoring the chance for false positives, we also know that the detection probability is 0 if the animal is not present at the site.

$$p(detect) = \begin{cases} \text{Species Present at Site} & \beta_0 + \beta_1 N \\ \text{Species Absent at Site} & 0 \end{cases}$$
$$Detected \sim Bernouli(p(detect))$$

That's pretty much going to be the meat of our model, these two equations.

# 3 Writing the Model in JAGS

Let's start by taking those two equations and turning them into JAGS-speak. First we tell jags that for each site, the probability of being occupied comes from a linear equation with

a logit-link. This is because we want $\psi$ to stay between 0 and 1, otherwise it wouldn't be a probability.

```
for (i in 1:n.sites) { #the end of this parenthesis is later in the code
    logit(psi[i]) <- psi.b0 + psi.b1*elevation[i]
    occ[i] ~ dbern(psi[i])
```

Next we tell jags that our probability of detection is a linear regression related to environmental noise with another logit link:

```
for (t in 1:6) {#the end of this parenthesis is later in the code too! :)
      logit(p[i,t]) <- p.b0 + p.b1*noise[i,t]
```

Finally we tell JAGS that our data - the present/absent information for each site at each time period - comes from a mixture of the probability of detecting it and the probability that it is present at the site at all.

```
obs[i,t] ~ dbern(p[i,t] * occ[i])
```

We'll need to add in priors for our intercepts and betas, but we might also want to add up how many sites actually have CAWA at them and compare this to our raw occupancy. If you leave brackets open in JAGS, it will just iterate through all the items in the vector - in this case from 1 to 40 (sites)

```
totalocc <- sum(occ[])
```

Finally, let's ask JAGS to give us some point estimates so we can graph the relationship between elevation and site suitability.

```
for (k in 1:n.graph){
    logit(graph.me[k]) <- psi.b0 + psi.b1*fake.elev[k]
   }
```

Let's put all that together into one big model code:

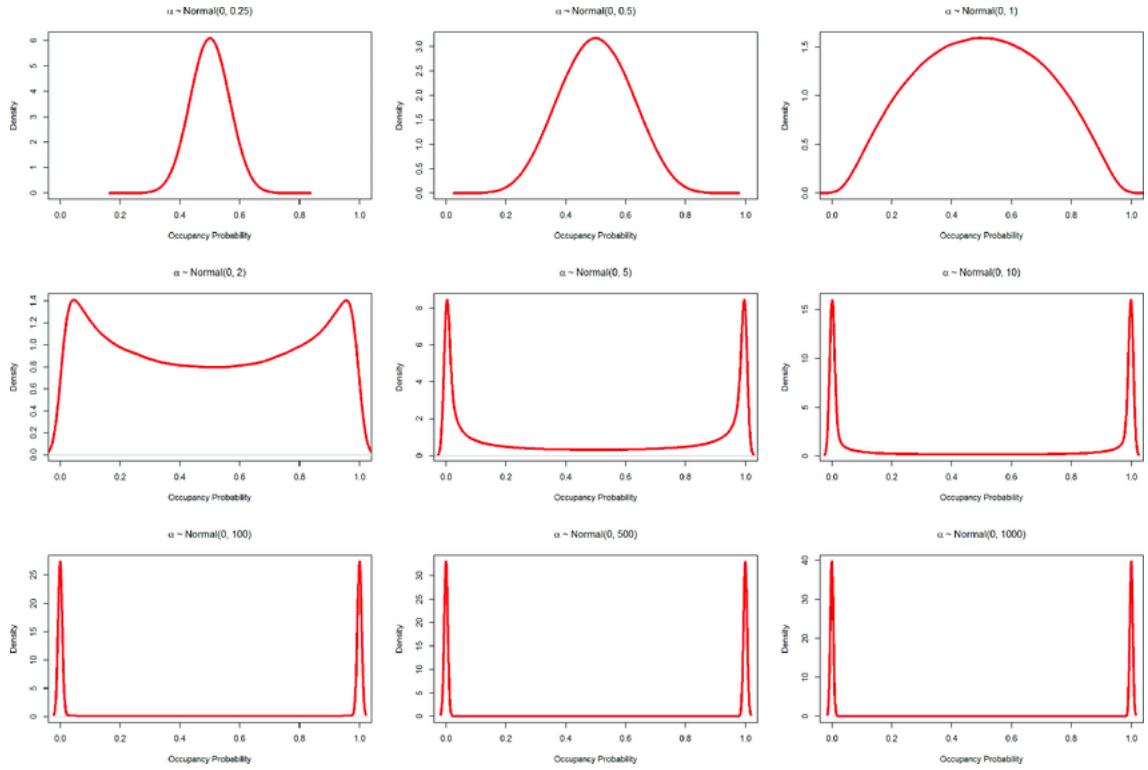
```
modelstring.occ = "
model {
  for (i in 1:n.sites) {
    logit(psi[i]) <- psi.b0 + psi.b1*elevation[i]
    occ[i] ~ dbern(psi[i])

    for (t in 1:6) {
      logit(p[i,t]) <- p.b0 + p.b1*noise[i,t]
      obs[i,t] ~ dbern(p[i,t] * occ[i])
    }
  }
  psi.b1 ~ dnorm(0, 0.37)
  psi.b0 ~ dnorm(0, 0.37)
  p.b0 ~ dnorm(0, 0.37)
  p.b1 ~ dnorm(0, 0.37)

  totalocc <- sum(occ[]) #this just says sum across all the sites in occ

  for (k in 1:n.graph){
    logit(graph.me[k]) <- psi.b0 + psi.b1*fake.elev[k]
    }
}
"
```

Those priors I chose seem kinda funky, huh? Let me diverge from our topic briefly...

# 4 Note: Priors When Transforming Data

Priors can be tricky things. When we don't transform our data, it's not too hard to judge if our priors are "vague" or not - we just set bounds that we know are reasonable to not influence our result. But when we transform data, such as when we use logit links, the priors may also need to change.

Here's a good example of how distributions change when we use the logit link. Say we have a normal distribution with mean 0, standard deviation = 5. Maybe this represents our intercept for our site occupancy linear equation. Before we transform the data, we're suggesting that the value could fall between ∼-10 and 10, which is pretty vague. But once we transform our equation to the probability scale, suddenly our prior isn't vague at all! Now it's saying the probability has to be either ∼ 0 - .2 or .8-1 -that's a pretty informative prior! Check out this figure from Northrup and Gerber (2018). Each panel shows a different normal prior with mean = 0 and standard deviations transformed to the probability scale. As the standard deviation goes up, we see the distribution push more and more likelihood to the very extremes - 1's and 0's. Crazy, huh?




Citation: Northrup, Joseph & Gerber, Brian. (2018). A comment on priors for Bayesian occupancy models. PLOS ONE. 13. e0192819. 10.1371/journal.pone.0192819.

There are two general ways of dealing with this issue. One method is by using a "Jeffreys Prior", which tries to avoid this by being "invariant under re-parameterization." In other words, if we think we've set a vague prior and then we use the logit function on our equation,

we'd still like to be using a vague prior when we're done. I am unfortunately completely unqualified to explain the idea any further than that. The other method, the one I use, is simply to put your priors on the scale they're going to be used on - e.g. if they're going to be logit transformed, make sure they make sense on the probability scale. Up above, I've done just that - a normal distribution with mean 0 and precision of .377 ends up approximately uniform when transformed to the probability scale. I could probably have also just used a uniform distribution to begin with. There's no one "best" way to do things, but it's important to always consider what your priors are doing before you run your model!

# 5 Running the Model in JAGS

Time to actually run the model! First we give data, including data on the fake elevations we want to get point estimates for.

```
jd.Foo <- list(n.sites = nrow(Birds),elevation = Sites$Elevation, noise =
    as.matrix(Birds[,8:13]), obs = as.matrix(Birds[,2:7]), fake.elev =seq
    (.2, 1.5, by = .025), n.graph = 53)
```

Next come inits. For occupancy modeling, or really for any models where you have Bernoulli draws, JAGS can get a little stuck on the initial values unless you supply them. I find the best way to avoid initialization errors (which lead to the dreaded "node inconsistent with parents" error) is to just initialize all (or most) of the sites as occupied.

```
ji.Foo <- function(){list(psi.b0 = runif(1,0,1), psi.b1 = runif(1,0,1), p.
    b0 = runif(1,0,1), p.b1 = runif(1,0,1), occ = rep(1,nrow(Sites)))}
```

Next params to monitor:

```
params <- c("psi.b0", "psi.b1", "p.b0", "p.b1", "totalocc", "graph.me")
```

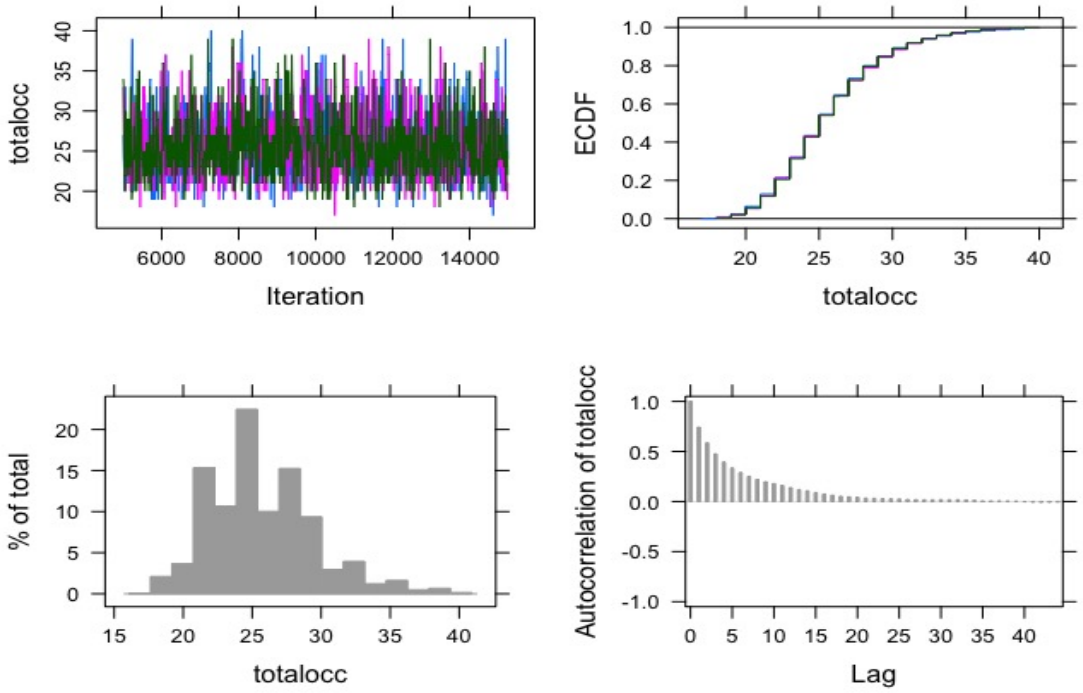
And finally we run the model:

```
Foo <- run.jags(model = modelstring.occ, monitor = params, data = jd.Foo,
    n.chains = 3, inits = ji.Foo, burnin = 4000, sample = 10000, adapt =
    1000, method = "parallel")
mod <- summary(Foo)
mod
plot(Foo)
```

| | Lower95 | Median | Upper95 | Mean | psrf |
|---|---|---|---|---|---|
| psi.b0 | -2.83 | -1.06 | 0.99 | -1.00 | 1.00 |
| psi.b1 | 0.32 | 2.39 | 4.53 | 2.43 | 1.00 |
| p.b0 | -0.03 | 0.97 | 2.02 | 0.98 | 1.00 |
| p.b1 | -0.77 | -0.52 | -0.30 | -0.53 | 1.00 |
| totalocc | 19.00 | 25.00 | 33.00 | 25.64 | 1.00 |

From our summary statistics, our model seems to have converged, but we can plot it just in case. Note that the histogram of our "total occupancy" variable will be a little different ("blockier" if you will) than we're normally used to, since total occupied sites has to be an integer. Plotting is also a great time to make double sure your priors weren't terrible- if you

see something odd in the posterior plots for your model, it may be time to re-evaluate your priors. But we're good here.




So according to our output, about 18-33 of our 40 sites were actually occupied by CAWA, even though we only saw CAWA at 17 sites! The real answer (since I simulated this data, I know the true values) was 25 sites, which is right in the middle of our credible interval. Not bad!

# 6 Graphing the Results

Let's graph the probability of bird occupancy at a site according to our posterior values.

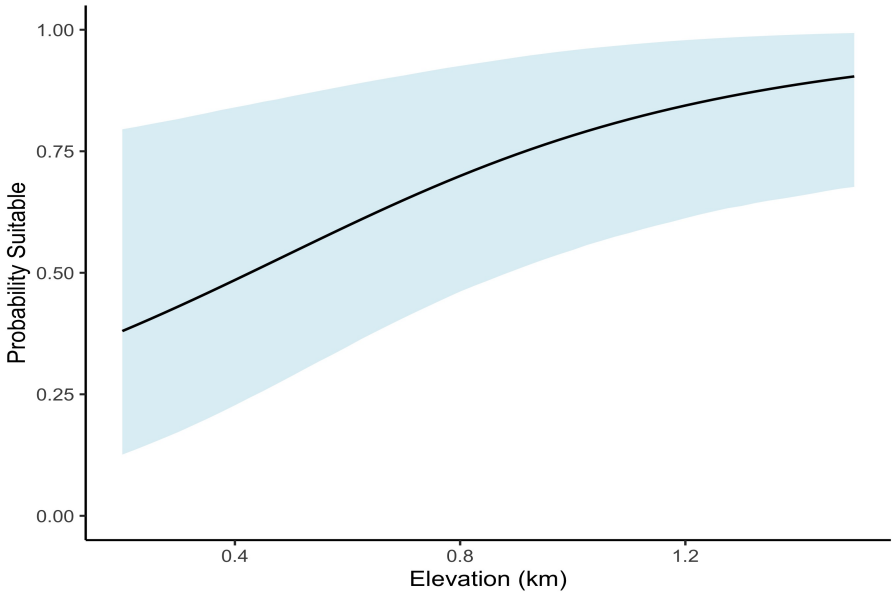
First we grab all the data from our results summary objects (called "mod" as above).

```
birdplot <- data.frame(Elevation =seq(.2, 1.5, by = .025), Mean = mod
    [6:58,4], Low = mod[6:58,1], High = mod[6:58,3])
```

Next let's stick it into ggplot (base R answer in attached code, but I'll skip it for now).

```
library(ggplot2)
G2 <- ggplot(birdplot, aes(x = Elevation))+
  geom_smooth(aes(y = Mean, ymin = Low, ymax = High), fill = "lightblue",
    colour = "black", stat = "Identity")+
  labs(x = "Elevation (km)", y = "Probability Suitable")+
  ylim(0,1)+
  theme_classic(base_size = 18)+
  theme(legend.position = "bottom",
        panel.grid.minor = element_blank(),
        panel.grid.major.x = element_blank(),
```

```
        strip.background =element_blank())
G2
```




Obviously you can make this graph prettier :)

# 7 Running in NIMBLE

The NIMBLE model requires only 2 changes - one in the top of the model as always, and another in our total occupancy variable. NIMBLE does not like open, empty brackets - so you can't just ask it to sum over all the sites without telling it exactly which sites to sum over.

```
library(nimble)
library(coda)
nimble.birds <- nimbleCode({ #this is change #1
  ## Loop over sites
  for (i in 1:n.sites) {
    logit(psi[i]) <- psi.b0 + psi.b1*elevation[i]
    occ[i] ~ dbern(psi[i])

    ## Loop over replicates within site
    for (t in 1:6) {
      logit(p[i,t]) <- p.b0 + p.b1*noise[i,t]

      #The actually observed data is obs[i,t]
      obs[i,t] ~ dbern(p[i,t] * occ[i])
    }
  }
  ## Priors
  psi.b1 ~ dnorm(0, 0.37)
  psi.b0 ~ dnorm(0, 0.37)
```

```
  p.b0 ~ dnorm(0, 0.37)
  p.b1 ~ dnorm(0, 0.37)

  # How many sites are truly occupied?
  totalocc <- sum(occ[1:n.sites]) #this is difference #2. NIMBLE makes you
    specify indexing.

  #Let's also graph the relationship between occupancy and elevation.
    for (k in 1:n.graph){
    logit(graph.me[k]) <- psi.b0 + psi.b1*fake.elev[k]
    }
})
```

Time to specify our parameters, data and constants. Remember that in NIMBLE data means parameters that come from distributions. So in this case our only data is our presence/absence data, everything else is a constant.

```
params <- c("psi.b0", "psi.b1", "p.b0", "p.b1", "totalocc", "graph.me")
data <- list(obs = as.matrix(Birds[,2:7]))
constants <- list(n.sites = nrow(Birds),elevation = Sites$Elevation,
  noise =as.matrix(Birds[,8:13]), fake.elev =seq(.2, 1.5, by = .025), n.
   graph = 53)
inits <- list(psi.b0 = runif(1,0,1), psi.b1 = runif(1,0,1), p.b0 = runif
   (1,0,1), p.b1 = runif(1,0,1), occ = rep(1,nrow(Sites)))
```

Now we can run the model! I encourage you to do this the normal long way, but there's actually a "short" way to run the model through one command. Both ways produce the same answer but the long way is a little less "black boxy" if you get an error (especially if your code takes hours to run and then you find out the error was an initial value or something).

The fast way:

```
nimbleMCMC(code = nimble.birds, constants = constants, data = data, inits
   = inits, monitors = params, niter = 30000, thin = 1, nchains =3,
   nburnin = 10000, samplesAsCodaMCMC = TRUE)
```

The longer way that is still useful for learning:

```
prepbirds <- nimbleModel(code = nimble.birds, constants = constants, data
   = data, inits = inits)
prepbirds$initializeInfo() #everything is good to go!
mcmcbirds <- configureMCMC(prepbirds, monitors = params, print = T )
birdsMCMC <- buildMCMC(mcmcbirds)
Cmodel <- compileNimble(prepbirds)
Compbirds <- compileNimble(birdsMCMC, project = prepbirds)
bird.mod.nimble <- runMCMC(Compbirds, niter = 30000, thin = 1, nchains =3,
    nburnin = 10000, samplesAsCodaMCMC = TRUE)
```

To check the code, we will want to do the standard 3 checks - looking at the convergence for the non-graphing nodes, checking the summary stats to see if we got reasonable answers and plotting the chains.

```
gelman.diag(mcmc.list(bird.mod.nimble$chain1[,54:58],bird.mod.nimble$
   chain2[,54:58],bird.mod.nimble$chain3[,54:58]) )
```

Our model appears to have converged.

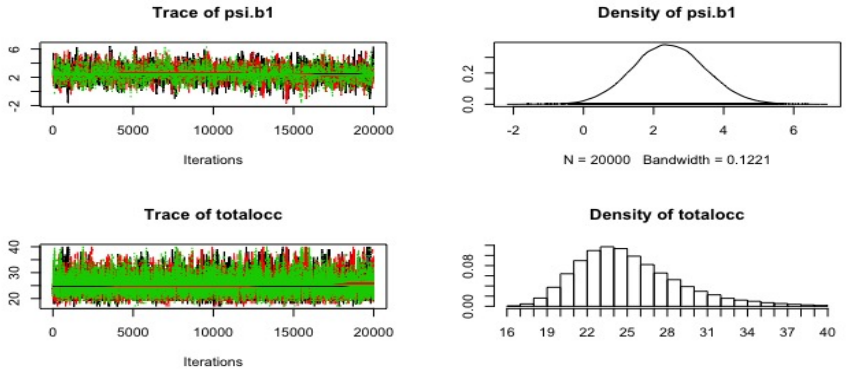
| | Point est. | Upper C.I. |
|---|---|---|
| p.b0 | 1.00 | 1.01 |
| p.b1 | 1.00 | 1.00 |
| psi.b0 | 1.01 | 1.02 |
| psi.b1 | 1.00 | 1.01 |
| totalocc | 1.00 | 1.01 |

```
summary(mcmc.list(bird.mod.nimble$chain1[,54:58],bird.mod.nimble$chain2
    [,54:58],bird.mod.nimble$chain3[,54:58]))
```

| | 2.5% | 25% | 50% | 75% | 97.5% |
|---|---|---|---|---|---|
| p.b0 | -0.01 | 0.63 | 0.97 | 1.32 | 2.08 |
| p.b1 | -0.80 | -0.60 | -0.51 | -0.44 | -0.32 |
| psi.b0 | -2.72 | -1.67 | -1.10 | -0.47 | 1.11 |
| psi.b1 | 0.45 | 1.74 | 2.44 | 3.14 | 4.52 |
| totalocc | 20.00 | 23.00 | 25.00 | 28.00 | 35.00 |

Results are fairly similar from what we found in JAGS.

```
plot(mcmc.list(bird.mod.nimble$chain1[,54:58],bird.mod.nimble$chain2
    [,54:58],bird.mod.nimble$chain3[,54:58]))
```




In this case, everything has converged and we get pretty much what we expected! For code on how to graph the NIMBLE results, check the associated R code on the github repository.