Simple_Bayesian_Occupancy

Heather Gaya

4/6/2021

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. For this tutorial I am using NIMBLE version 0.10.1

Please send any questions or suggestions to heather.e.gaya(at)gmail.com or find me on twitter: doofgradstudent

Contents

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!

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.

```r
setwd("~/Desktop/U Georgia/R_Workshop/JAGS:NIMBLE/Occupancy_Models")
Birds <- read.csv("Bird_Obs.csv")
Sites <- read.csv("Site_Info.csv")
```

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	0.7538155
2	0.7121482
3	0.3465300
4	1.3521395
5	0.3153131

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

Site	Obs1	Obs2	Obs3	Obs4	Obs5	Obs6	noise1	noise2	noise3	noise4	noise5	noise6
1	0	0	0	0	0	0	15.0202826	9.605626	13.4102917	13.487420	6.086373	3.1292577
2	1	0	0	0	0	0	0.7592899	6.576631	19.0706884	6.312125	18.662786	11.9114211
3	0	0	0	0	0	0	18.2455555	8.332412	0.1204134	7.309415	12.786702	18.5008937
4	0	0	0	0	0	0	13.7249622	17.768580	15.5755710	10.767122	10.258653	2.3036472
5	0	0	0	0	0	0	11.0212305	19.227138	15.6166907	4.643094	5.062925	0.6347369

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.

```r
sum(rowSums(Birds[, 2:7]) > 0)  #17 sites
## [1] 17
```

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

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!

Let's think about the process that allows a site to be suitable. For CAWA, we know they seek out higher elevation - possibly because of lower temps or wetter conditions - 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:

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

where ψ refers to the probability a site is suitable.

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.

$$p(detect) = \begin{cases} \beta_0^p + \beta_1^p Noise & \text{if species present} \\ 0 & \text{otherwise} \end{cases}$$
$$Detect \sim Bernouli(p(detect))$$

That's pretty much going to be the meat of our model!

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 witha logit-link. This is because we want ψ to stay between 0 and 1, otherwise it wouldn't be a probability. We also will change out the nameless covariate "X" to be site elevation instead.

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

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

```
for (t in 1:6) {
    logit(p[i, t]) <- p.b0 + p.b1 * noise[i, t]
}
```

Remember, $p[i, t]$ just means the $i - th$ row of the $t - th$ column of p. So for site 1, time period 4, JAGS will use $p[1, 4]$.

Finally we tell JAGS that our data, *obs*, - 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[])

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

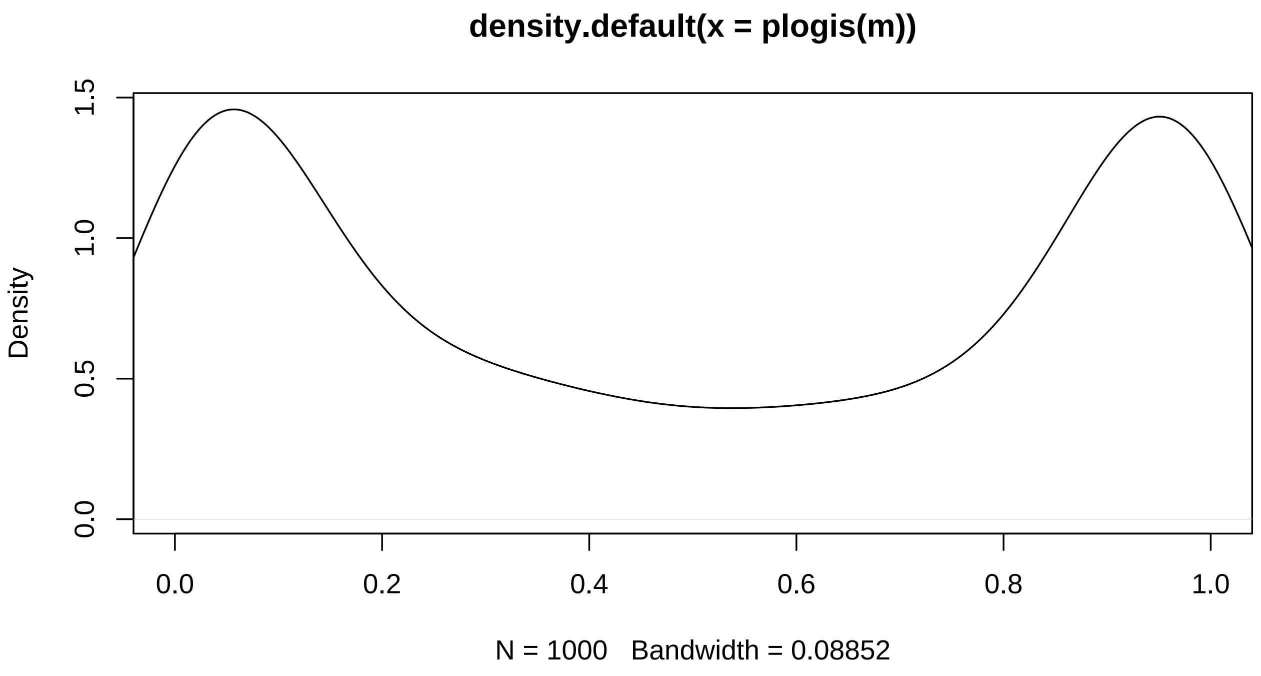
Those priors I chose seem kinda funky, huh? Let me diverge from our topic briefly…

A Note on 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.

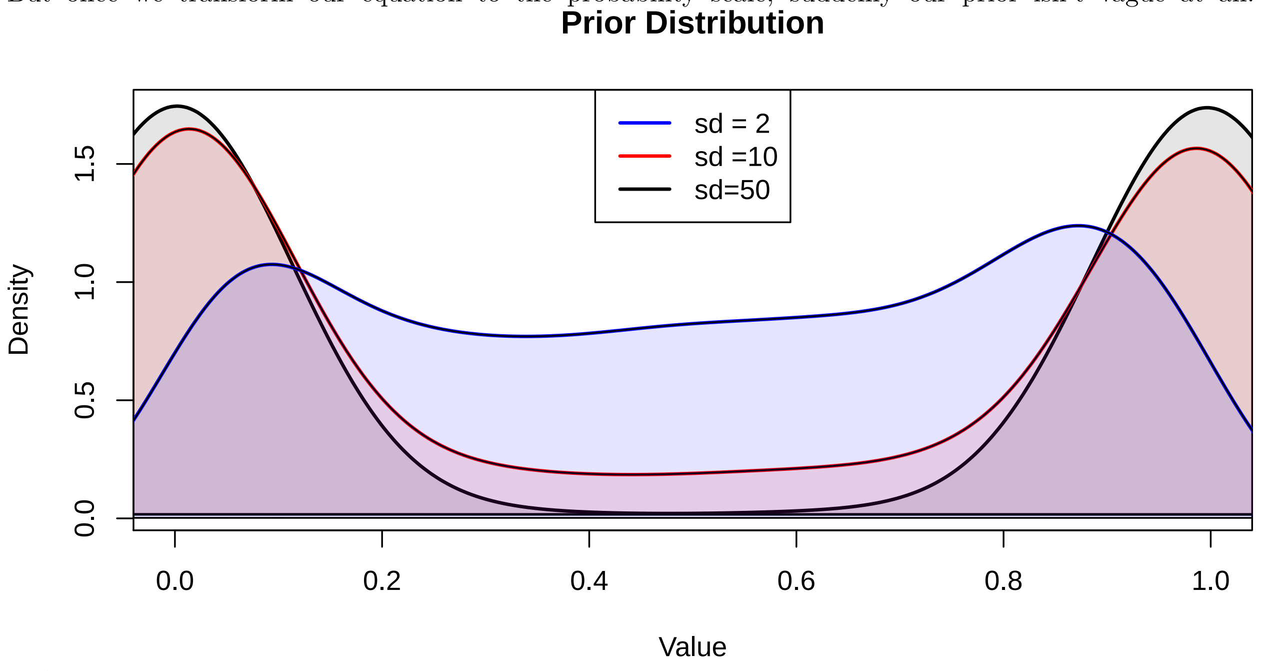
Here's two visual examples of how distributions change when we use the logit link. First, what if our prior is a uniform from -5 to 5?

```
m <- runif(1000, min = -5, max = 5)
plot(density(plogis(m)), xlim = c(0, 1))
```



Uh-oh, that doesn't look super flat. See how all the probability is getting squished up on the sides? It's not the end of the world, but it's not very flat.

What if we have a normal distribution with mean 0, standard deviation = 10? Before we transform the data, we're suggesting that the value could fall between ∼ -20 and 20, which is pretty vague. But once we transform our equation to the probability scale, suddenly our prior isn't vague at all!



In fact, the larger the standard deviation, the less vague this prior becomes.

A great paper to read for more info on this topic is: 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.

Up above, I've done something to avoid this issue - a normal distribution with mean 0 and precision of .377 ends up approximately uniform when transformed to the probability scale.

The other method 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, but your prior on the logit version of the variable. For instance, in our model we could do:

```
logit(p.b0) <- logit.pb0
logit.pb0 ~ dunif(-5, 5)
```

This isn't always the easiest, because you could run into issues where you end up with values larger than 1 or smaller than 0 if you're not careful, so I prefer method 1.

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!

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(0.2, 1.5, by = 0.025), n.graph = length(seq(0.2,
        1.5, by = 0.025)))
```

Next come initial values. 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.

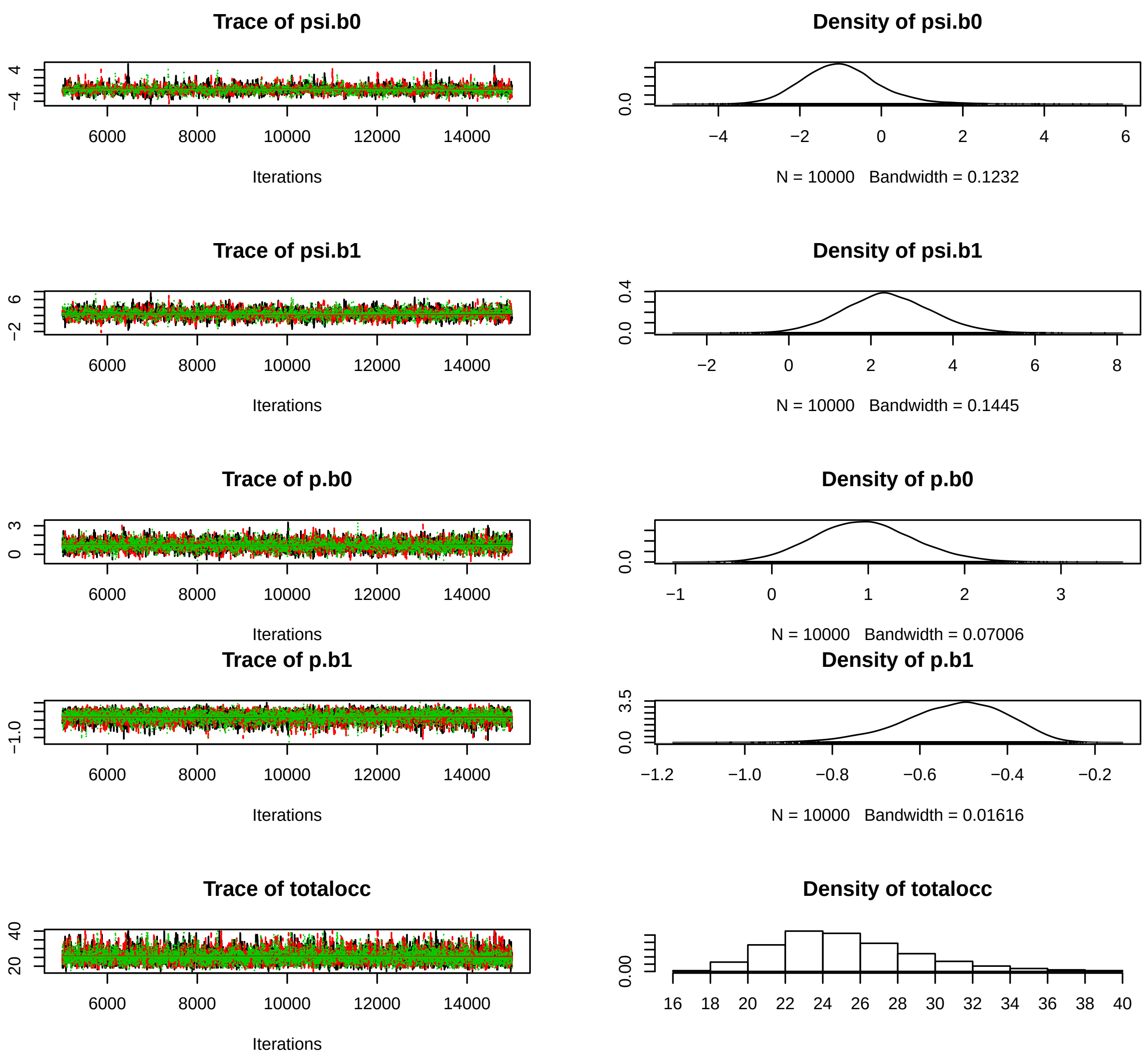
```
ji.Foo <- function() {
    list(psi.b0 = runif(1, 0, 1), psi.b1 = runif(1, 0, 1), p.b0 = runif(1,
        0, 1), p.b1 = runif(1, 0, 1), occ = rep(1, nrow(Sites)))
}
```

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")
## Warning: You attempted to start parallel chains without setting different PRNG
## for each chain, which is not recommended. Different .RNG.name values have been
## added to each set of initial values.
## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Tue Apr  6 15:07:32 2021
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 240
##    Unobserved stochastic nodes: 44
##    Total graph size: 1862
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 1000
## -------------------------------------------------| 1000
## ++++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 4000
## -------------------------------------------------| 4000
## ************************************************** 100%
## . . . . . . . . Updating 10000
## -------------------------------------------------| 10000
## ************************************************** 100%
## . . . . . Updating 0
## . Deleting model
## .
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
mod <- summary(Foo)
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 58 variables....
mod[1:5, ]
##             Lower95     Median   Upper95       Mean        SD Mode       MCerr
## psi.b0   -2.8606400 -1.0350950  1.024270 -0.9737717 0.9976949   NA 0.025461198
## psi.b1    0.2341730  2.3819450  4.528510  2.4055947 1.0923875   NA 0.022880568
## p.b0     -0.0506469  0.9454785  1.997010  0.9594042 0.5212611   NA 0.009502023
## p.b1     -0.7598060 -0.5115850 -0.301297 -0.5216102 0.1198406   NA 0.001828553
## totalocc 19.0000000 25.0000000 33.000000 25.7303667 3.8661819   24 0.073402107
##          MC%ofSD SSeff     AC.10     psrf
## psi.b0       2.6  1535 0.3542732 1.004312
## psi.b1       2.1  2279 0.2347103 1.002542
## p.b0         1.8  3009 0.1432137 1.000695
## p.b1         1.5  4295 0.0731399 1.000960
## totalocc     1.9  2774 0.1823097 1.001797
plot(Foo$mcmc[, 1:5, ])
```



From our summary statistics, our model seems to have converged and our plot looks great too. According to our output, 19 - 33 of our 40 sites were likely 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!

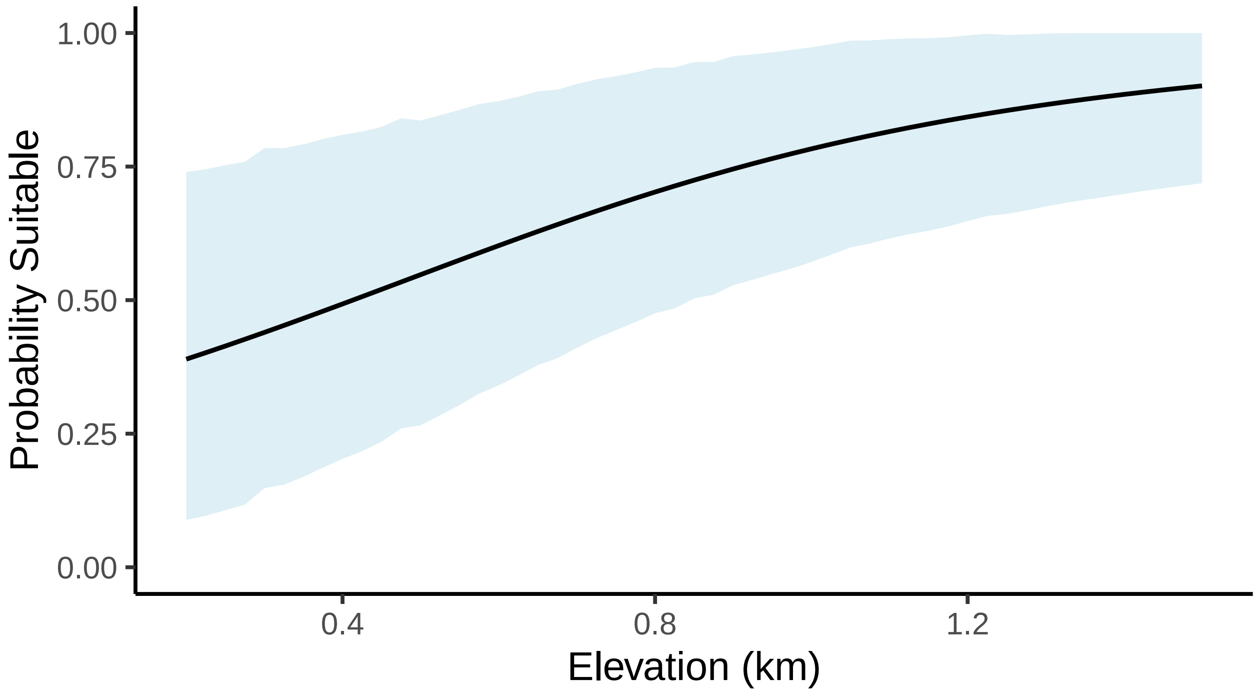
Graphing the Results

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). We can use the "grep" function to grab all the variables that have the word "graph" in them.

```
graph.vars <- mod[c(grep("graph", rownames(mod))), ]
birdplot <- data.frame(Elevation = seq(0.2, 1.5, by = 0.025),
    Mean = graph.vars[, 4], Low = graph.vars[, 1], High = graph.vars[,
        3])
```

Next let's stick it into ggplot (or base R).

```r
library(ggplot2)
## Warning: package 'ggplot2' was built under R version 3.6.2
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 :)

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.

```r
library(nimble)
library(coda)
nimble.birds <- nimbleCode({
    # this is change #1
    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])
        }  #end t
    }  # end 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(0.2, 1.5,
        by = 0.025), 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 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:

```
birds.mod.nimble <- nimbleMCMC(code = nimble.birds, constants = constants,
    data = data, inits = inits, monitors = params, niter = 30000,
    thin = 1, nchains = 3, nburnin = 10000, samplesAsCodaMCMC = TRUE)
## defining model...
## building model...
## setting data and initial values...
## running calculate on model (any error reports that follow may simply reflect missing values in model
## checking model sizes and dimensions...
## checking model calculations...
## model building finished.
## compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.
## compilation finished.
## running chain 1...
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## running chain 2...
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## running chain 3...
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

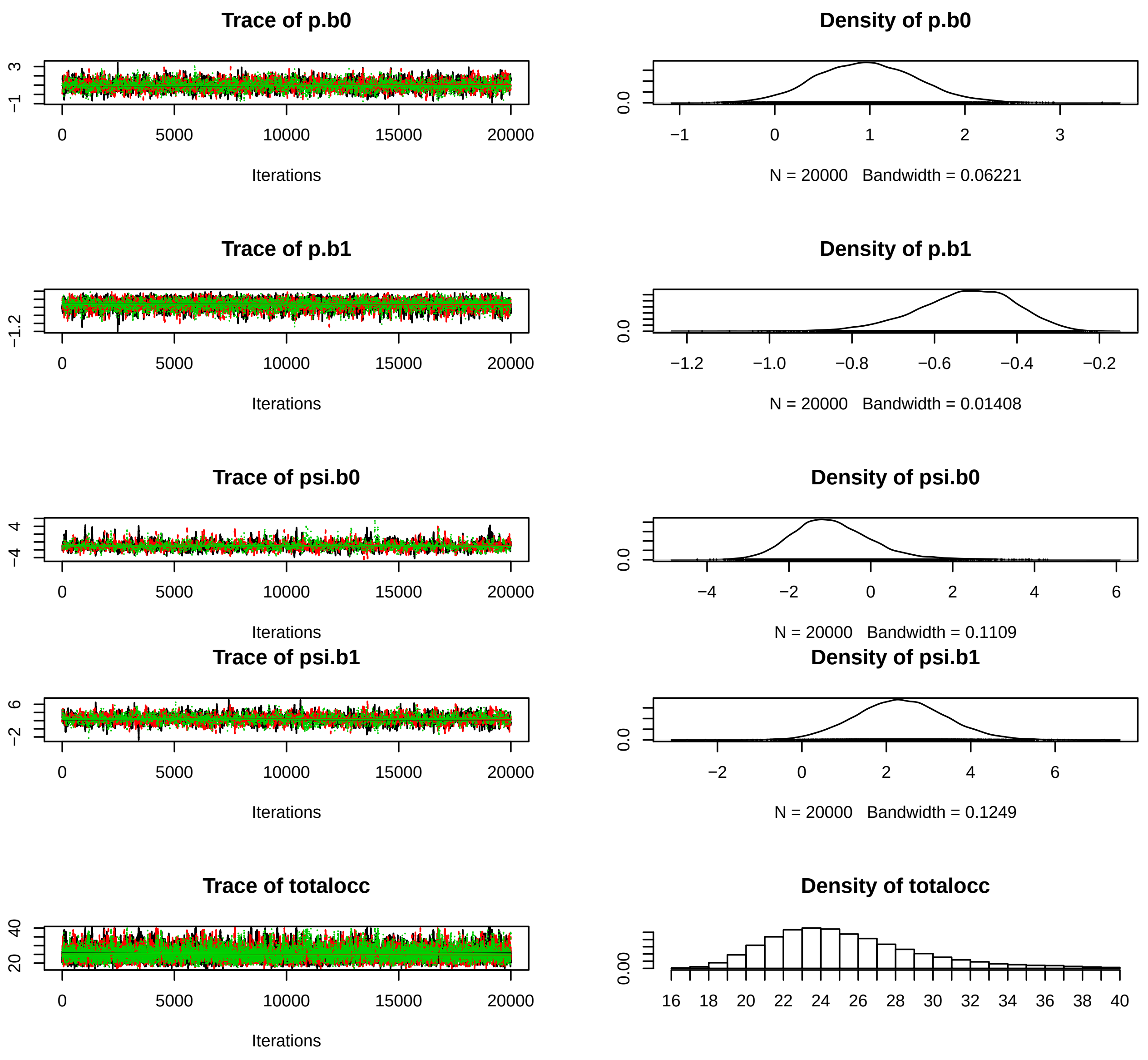
(Note that you can't extend runs if you do it this way.)

The longer way that is still useful for learning:

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

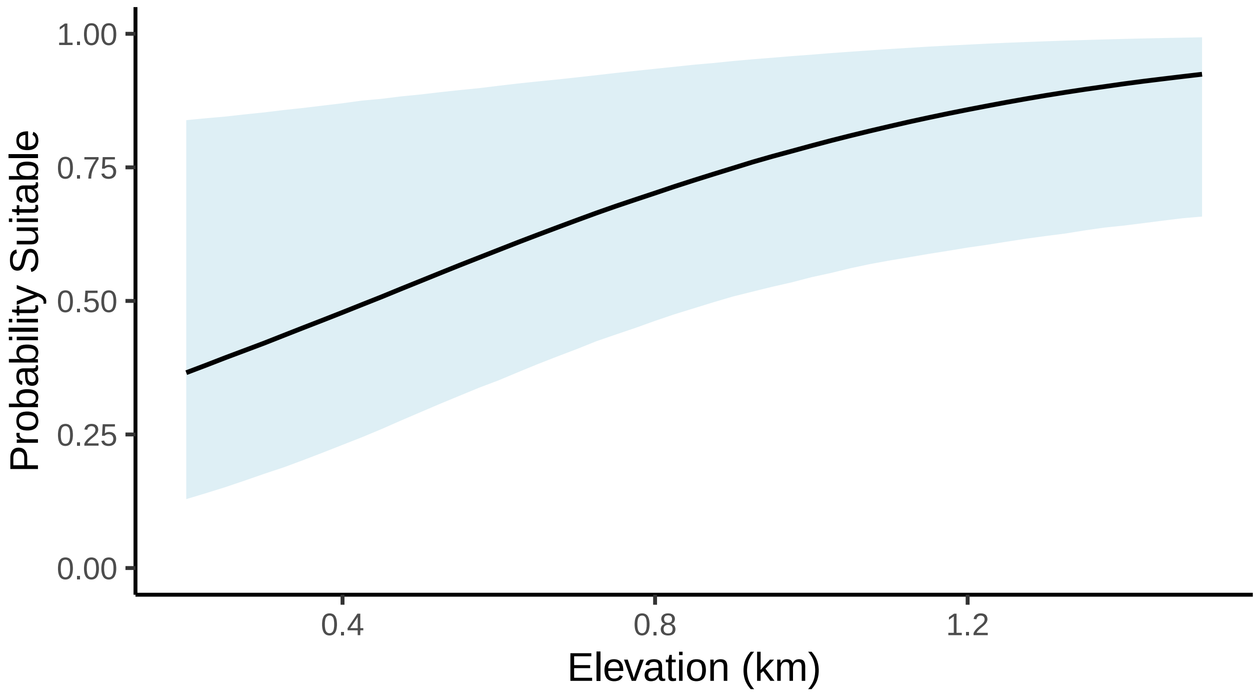
```
dimnames(birds.mod.nimble$chain1)[[2]]  #check what variables are in the output
##  [1] "graph.me[1]"  "graph.me[2]"  "graph.me[3]"  "graph.me[4]"  "graph.me[5]"
##  [6] "graph.me[6]"  "graph.me[7]"  "graph.me[8]"  "graph.me[9]"  "graph.me[10]"
## [11] "graph.me[11]" "graph.me[12]" "graph.me[13]" "graph.me[14]" "graph.me[15]"
## [16] "graph.me[16]" "graph.me[17]" "graph.me[18]" "graph.me[19]" "graph.me[20]"
## [21] "graph.me[21]" "graph.me[22]" "graph.me[23]" "graph.me[24]" "graph.me[25]"
## [26] "graph.me[26]" "graph.me[27]" "graph.me[28]" "graph.me[29]" "graph.me[30]"
## [31] "graph.me[31]" "graph.me[32]" "graph.me[33]" "graph.me[34]" "graph.me[35]"
## [36] "graph.me[36]" "graph.me[37]" "graph.me[38]" "graph.me[39]" "graph.me[40]"
## [41] "graph.me[41]" "graph.me[42]" "graph.me[43]" "graph.me[44]" "graph.me[45]"
## [46] "graph.me[46]" "graph.me[47]" "graph.me[48]" "graph.me[49]" "graph.me[50]"
## [51] "graph.me[51]" "graph.me[52]" "graph.me[53]" "p.b0"         "p.b1"
## [56] "psi.b0"       "psi.b1"       "totalocc"
gelman.diag(birds.mod.nimble[, 54:58, ])
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## p.b0           1.00       1.01
## p.b1           1.00       1.00
## psi.b0         1.01       1.01
## psi.b1         1.00       1.00
## totalocc       1.00       1.01
##
## Multivariate psrf
##
## 1
summary(birds.mod.nimble[, 54:58, ])
##
## Iterations = 1:20000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean     SD  Naive SE Time-series SE
## p.b0      0.9687 0.5299 0.0021633       0.010577
## p.b1     -0.5235 0.1201 0.0004904       0.002068
## psi.b0   -0.9217 1.0263 0.0041899       0.029570
## psi.b1    2.3418 1.0703 0.0043695       0.025754
## totalocc 25.7784 4.0431 0.0165060       0.084849
##
## 2. Quantiles for each variable:
##
##              2.5%     25%     50%     75%   97.5%
## p.b0     -0.03969  0.6010  0.9586  1.3253  2.0452
## p.b1     -0.78430 -0.5986 -0.5147 -0.4379 -0.3134
## psi.b0   -2.67408 -1.6044 -1.0138 -0.3390  1.4491
## psi.b1    0.28480  1.6200  2.3269  3.0460  4.4842
## totalocc 20.00000 23.0000 25.0000 28.0000 36.0000
plot(birds.mod.nimble[, 54:58, ])
```



To get the graph results, we can do the same things we did with JAGS.

```r
mod.n <- summary(birds.mod.nimble)
graph.vars <- mod.n$quantiles[c(grep("graph", rownames(mod.n$quantiles))),
    ]
birdplot <- data.frame(Elevation = seq(0.2, 1.5, by = 0.025),
    Mean = graph.vars[, 3], Low = graph.vars[, 1], High = graph.vars[,
        5])

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
```


Probability Suitable
Elevation (km)