Hierarchical analysis of distance sampling data in R using `HierarchicalDS`: a vignette

Paul B. Conn (paul.conn 'at' noaa.gov)

July 15, 2013

HierarchicalDS version 2.1

Contents

1 Introduction

This vignette describes the R package, `HierarchicalDS`, which was designed to perform MCMC inference using a hierarchical modeling formulation for distance sampling data. The initial publication associated with this work is Conn *et al.* (2012), who described a hierarchical framework for double observer distance sampling data. The model structure in `HierarchicalDS` is quite similar, but has been expanded to include additional modeling features, such as the ability to include intrinsic conditionally autocorrelated (ICAR) spatial random effects, to include zero inflation in underlying abundance intensity, and to account for species misclassification under a double observer setup (see e.g. Conn *et al.*, Accepted).

The `hierarchicalDS` package is a flexible and powerful tool for analyzing distance sampling data, but there are some possible drawbacks associated with this flexibility. First, it assumes some familiarity with Markov chain Monte Carlo (MCMC; Gelman *et al.*, 2004) diagnostics to interpret Markov chain convergence. A basic knowledge of linear models and design matrices is useful for setting starting values for MCMC estimation and for interpretation of parameter estimates. Some intuition about parameter identifiability will be useful too; for instance, with single observers only it will not be possible to estimate the point independence correlation parameter (see below), nor will it be possible to implement species misidentification unless informative priors are used. Similarly, it would typically be a bad idea to implement advanced features such as spatially autocorrelated random effects or zero inflation with sparse data. In short, `hierarchicalDS` is a sharp tool; watch that you don't get cut!

The remainder of this vignette is structured as follows. First, we describe the model structure of associated with `hierarchicalDS`. Our description is admittedly brief, as basic components (including Gibbs sampling algorithms) have been described elsewhere (e.g. Conn *et al.*, 2012, Accepted). Next, we describe procedures for installation, focusing on the Windows environment (it is assumed that those using Unix, Linux, MacOS, etc. already know what they are doing). Next, we simulate a distance sampling dataset to use as an example when describing implementation of HierarchicalDS. Finally, we include a small troubleshooting section.

2 Model structure

Temporarily assuming no zero inflation, the basic model structure assumed by `HierarchicalDS` can be summarized by a directed, acyclic graph (Fig. 1). Under this setup, an areal model for the log of abundance intensity is assumed, where the log of abundance intensity over a collection of sites ($\boldsymbol{\nu}$) can be written as

$$\log(\boldsymbol{\nu}) = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\eta} + \boldsymbol{\epsilon}.$$

Here, $\mathbf{X}$ is a design matrix constructed using user-input covariates and regression formulae, $\boldsymbol{\beta}$ is a vector of regression coefficients, $\boldsymbol{\eta}$ represent spatially autocor-

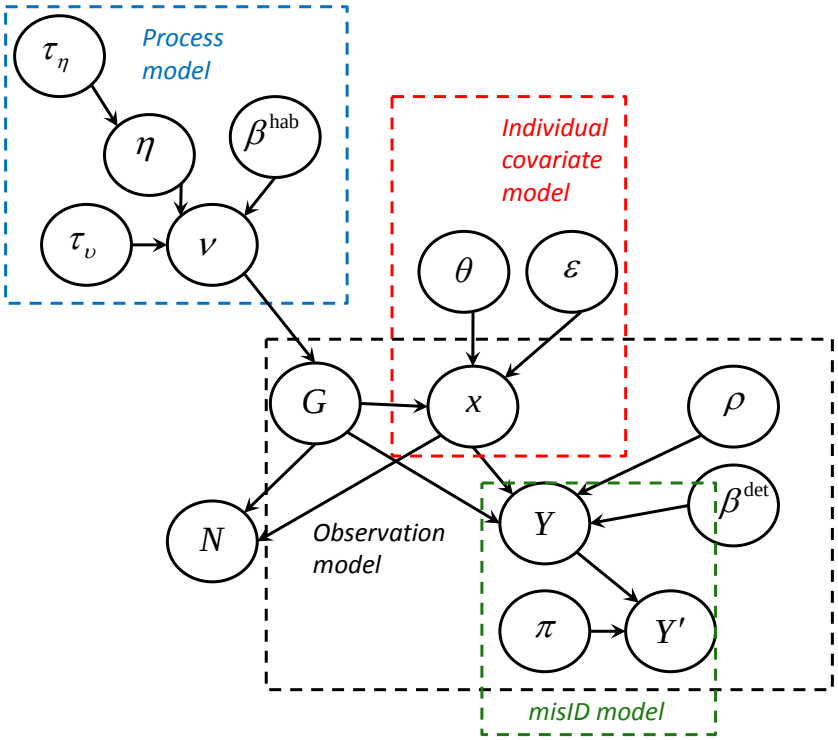


Figure 1: Directed, acyclic graph for the hierarchical model for distance sampling (here, assuming no zero inflation).

related random effects, and $\boldsymbol{\epsilon}$ represents additional, spatially uncorrelated error that can be used to impart extra-Poisson variation.

Conditional on abundance intensity for each species (separate abundance intensity models can be written for each species), and additional quantities such as the effective area sampled in each cell (also input by the user), we assume that the true numbers of clusters of animals $\mathbf{G}$ observed in each cell for each species is a realization of a Poisson process. If individual covariates associated with groups are available (e.g., group size, etc.), a variety of user-specified probability density and mass functions can be specified to model them with species-specific parameters. This is necessary for our "complete data" (Dempster *et al.*, 1977) representation of state space since individual covariates for undetected groups are modeled in areas where sampling occurs.

We assume that the detection of each these groups by either single- or double-observers is the outcome of a Bernoulli process, where detection probability (success probability) is modeled on the probit scale as a function of covariates.

Detection covariates can be entered on a cell-by-cell basis (as when sightability conditions may alter detection rates), or can be specified to be a function of individual covariates (e.g. group size). In the case of double observers, we allow a correlation between Bernoulli success probabilties, which occurs via a bivariate normal distribution on the probit scale. In this case, the correlation can increase linearly with distance from the transect line, in an effort to duplicate "point independence" (Borchers *et al.*, 2006; Laake & Borchers, 2004) - a modeling construct designed to combat the situation where some groups of animals are just must more detectable than other groups, especially at a distance (when this happens, detections by double observers are no longer independent).

3 Installation

There are several ways to install `hierarchicalDS`. The easiest is to simply download it from CRAN. In an R window, this can be achieved by typing

```
install.packages("hierarchicalDS")
```

Typically, the version of `hierarchicalDS` on CRAN will lag slightly behind development versions. The latest source code is publicly available at `https://github.com/NMML/Hierarchical_DS`, which can also be used to download base R functions for debugging if one runs into indeciphable errors (also see Troubleshooting below).

Github no longer hosts binaries or .zip files; these can currently be found on the following Google drive:
`https://drive.google.com/folderview?id=0BzomuXDutCcRT2VHMUlqWDhqNlk&usp=sharing`

This is the preferred place for downloading installation versions of the package between stable CRAN releases. There are several ways to install `hierarchicalDS` locally. The most stable way I have found is to open a DOS terminal window, navigate to the directory where `hierarcichalDS.tar` resides, and type `Rcmd INSTALL hierarchicalDS.tar`. There are other options from within R; for instance, one can use the `install.packages` function:

```
setwd("mydir")  #mydir is where the binary resides
install.packages("hierarchicalDS.tar.gz", repos = NULL, type = "source")
```

4 Example

I now consider implementing `hierarchicalDS` on a simulated data example. We will simultaneously demonstrate the use of complex model features, includ-

ing zero infation, spatial random effects, and species misidentification. The complete R code for the example is as follows:

```
library(hierarchicalDS)
# 1) First, simulate data
S = 100  #Number of grid cells; square grid assumed for simulation

n.transects = S  #one transect per cell
Observers = matrix(NA, 2, n.transects)
set.seed(11112)
for (i in 1:n.transects) {
    Observers[, i] = sample(c(1, 2, 3), size = 2, replace = FALSE)
}
Sim = simulate_data(S = S, Observers = Observers, misID = TRUE, ZIP = TRUE,
    tau.pois = 15, tau.bern = 10)
Dat = Sim$Dat

# 2) declare inputs and call hierarchicalDS
Obs.cov = array(0, dim = c(2, n.transects, 1))
Obs.cov[1, , ] = 1
n.obs.cov = 1
Adj = square_adj(sqrt(S))
misID.mat = matrix(0, 2, 3)  # misID matrix
misID.mat[1, ] = c(1, -1, 2)
misID.mat[2, ] = c(-1, 3, -1)
misID.models = c(~1, ~1, ~1)
MisID = vector("list", max(misID.mat))
MisID[[1]] = 2  #parameters for getting it right
MisID[[2]] = 1
MisID[[3]] = 3  #parameters for getting it right
misID.symm = TRUE
Mapping = c(1:S)
Area.trans = rep(1, S)
n.bins = length(unique(Dat[, "Distance"]))
Area.hab = rep(1, S)
Bin.length = rep(1, n.bins)  #equal bin lengths
Hab.cov = data.frame(rep(log(c(1:sqrt(S)/sqrt(S))), each = sqrt(S)))
# (covariate on abundance intensity same as used to simulate data)
colnames(Hab.cov) = c("Cov1")
Hab.pois.formula = vector("list", 2)
Hab.bern.formula = Hab.pois.formula
for (i in 1:2) {
    Hab.pois.formula[[i]] = ~Cov1
    Hab.bern.formula[[i]] = ~1
}
detect = TRUE
```

```
Det.formula = ~Observer + Distance + Group
n.species = nrow(misID.mat)
Cov.prior.parms = array(0, dim = c(n.species, 2, 1))
Cov.prior.parms[1, , 1] = c(2, 1)
Cov.prior.parms[2, , 1] = c(2, 1)
Cov.prior.fixed = matrix(0, n.species, dim(Cov.prior.parms)[3])
Cov.prior.pdf = Cov.prior.fixed
Cov.prior.pdf[, 1] = c("pois1", "pois1")  #model group size as a zero truncated poisson
Cov.prior.n = matrix(2, 2, 1)
point.ind = TRUE  #include point independence
spat.ind = FALSE  #do not make spatially independent
fix.tau.nu = FALSE
srr = TRUE
srr.tol = 0.5
misID = TRUE
last.ind = FALSE
cor.const = FALSE
grps = TRUE
post.loss = FALSE
M = t(Out$G.true * 10)
M[which(M < 30)] = 50
Control = list(iter = 21000, burnin = 1000, thin = 10, MH.cor = 0.2, MH.nu = matrix(0.2,
    2, S), MH.misID = matrix(0.1, 3, 1), RJ.N = matrix(rep(5, S * n.species),
    n.species, S), adapt = 100, iter.fix.N = 100)
hab.pois = matrix(0, n.species, 2)  #covariates are intercept, index
hab.pois[1, 1:2] = c(log(50), 0)
hab.pois[2, 1:2] = c(log(10), -2)
hab.bern = matrix(0, n.species, 1)
hab.bern[, 1] = c(1, 1)
# provide some initial values to ensure MCMC doesn't start out at weird
# place
Inits = list(hab.pois = hab.pois, hab.bern = hab.bern, tau.nu = c(500, 500),
    MisID = MisID)
misID.mu = vector("list", max(misID.mat))
misID.sd = misID.mu
misID.mu[[1]] = 0
misID.mu[[2]] = 0
misID.mu[[3]] = 0
misID.sd[[1]] = 1.75
misID.sd[[2]] = 1.75
misID.sd[[3]] = 1.75
Prior.pars = list(a.eta = 1, b.eta = 0.01, a.nu = 1, b.nu = 0.01, beta.tau = 0.01,
    misID.mu = misID.mu, misID.sd = misID.sd)
# (1,.01) prior makes it closer to a uniform distribution near the origin
adapt = TRUE
```

```
ZIP = TRUE
set.seed(8327329)  #chain1
Out = hierarchical_DS(Dat = Dat, Adj = Adj, Area.hab = Area.hab, Mapping = Mapping,
    Area.trans = Area.trans, Observers = Observers, Bin.length = Bin.length,
    Hab.cov = Hab.cov, Obs.cov = Obs.cov, n.obs.cov = n.obs.cov, Hab.pois.formula = Hab.pois
    Hab.bern.formula = Hab.bern.formula, detect = detect, Det.formula = Det.formula,
    Cov.prior.pdf = Cov.prior.pdf, Cov.prior.parms = Cov.prior.parms, Cov.prior.fixed = Cov
    Cov.prior.n = Cov.prior.n, pol.eff = NULL, ZIP = ZIP, point.ind = point.ind,
    spat.ind = spat.ind, fix.tau.nu = fix.tau.nu, srr = srr, srr.tol = srr.tol,
    misID = misID, last.ind = last.ind, cor.const = cor.const, Inits = Inits,
    grps = grps, M = M, Control = Control, adapt = adapt, Prior.pars = Prior.pars,
    misID.mat = misID.mat, misID.models = misID.models, misID.symm = misID.symm,
    post.loss = post.loss)

### 3) plot and summarize results; note that chain would need to be run a
### lot longer to summarize the posterior very well!!!
plot(Out$MCMC)
summary_N(Out)
post_loss(Out)
```

Okay, now let's run through this bit by bit. To start with, we'll load the `hierarchicalDS` library:

```
# library(hierarchicalDS)
```

```
source("c:/users/paul.conn/git/hierarchicalDS/hierarchicalDS/R/simulate_data.R")
source("c:/users/paul.conn/git/hierarchicalDS/hierarchicalDS/R/hierarchical_DS.R")
source("c:/users/paul.conn/git/hierarchicalDS/hierarchicalDS/R/mcmc_ds.R")
source("c:/users/paul.conn/git/hierarchicalDS/hierarchicalDS/R/spat_funcs.R")
```

Now we'll simulate some data using the `simulate_data` function. This function is hardwired to produce data for two species; abundance for species one increases linearly as one moves west to east but stays relatively constant for species two. Both are subject to zero inflation, with a spatial regression model for both the zero component and non-zero component. This is a particularly data hungry formulation, so data are simulated assuming the *entire area* is covered by transects.

```
# 1) First, simulate data
S = 100  #Number of grid cells; square grid assumed for simulation

n.transects = S  #one transect per cell
Observers = matrix(NA, 2, n.transects)
set.seed(11112)
```

```
for (i in 1:n.transects) {
    Observers[, i] = sample(c(1, 2, 3), size = 2, replace = FALSE)
}
Sim = simulate_data(S = S, Observers = Observers, misID = TRUE, ZIP = TRUE,
    tau.pois = 15, tau.bern = 10)
Dat = Sim$Dat
```

Let's take a look at the contents of the simulated data list object

```
names(Sim)

## [1] "Dat"          "G.true"       "True.species"
```

The data frame object `Sim$Dat` includes data suitable to pass to the analysis function `hierarchical_DS`, while the matrix object `Sim$G.true` holds the true number of groups of each species for each grid cell (recall there are $S = 100$ grid cells). The latter is useful for checking that `hierarchicalDS` is estimating a reasonable number of animals (at least for simulated data!). Finally, `Sim$True.species` holds the true species identity for each record.

Let's take a look at the first few records for the data frame holding the data we'll input into `hierarchicalDS` :

```
head(Dat)

##   Transect Match Observer Obs Species Seat Distance Group
## 1        1     1        3   1       3    1        2     3
## 2        1     1        2   1       1    0        2     3
## 3        1     2        3   0       1    1        4     3
## 4        1     2        2   1       1    0        4     3
## 5        1     3        3   1       1    1        3     3
## 6        1     3        2   1       1    0        3     3
```

The order in which columns appear is important; `hierarchicalDS` expects data will be in a similar format. The order expected is as follows:

1. The first column gives a numeric transect value, for $1, 2, \ldots, T$ (note that T will often be less than S). `hierarchicalDS` assumes that the investigator enters in a separate transect number for each transect-grid cell combination. For instance, if one transect spans 4 grid cells (recall there are S grid cells), then data collected on that transect would need to be partitioned by grid cell and the data would be entered as if 4 separate transects had been conducted. Note that transect numbers are associated with grid cell numbers using an input variable that will be described later.

2. The second column, 'Match,' holds a value for each animal cluster (group) that is detected. For double observers, there will be two rows for each cluster (rows corresponding to the same animal cluster get the same value of

'Match'); for single observers there will just be one row per cluster, and thus one value of 'Match'. No two groups of animals should have the same 'Match' value.

3. The third column, 'Observer' holds a factor variable indicating which observer was involved with a particular observation (or non-observation in the case of double observers).

4. The fourth column 'Obs' holds an indicator describing whether the particular observer saw the animal or not (for single observer data, all entries will be a '1').

5. The next row, 'Species' gives a numeric value for observed species. When there are > 1 species, `hierarchicalDS` allows the possibility of 'unknown' species observations, which are by default assumed to have a code equal to the number of species plus one (so for example, the first record in our simulated data was a record where species was entered as unknown).

6. The next record, 'Seat' is an example of an observer-level covariate. Observer level covariates are entered in columns between 'Species' and 'Distance,' and record variables the investigator can record on an observer-by-observer basis (e.g. experience level, etc.) or on a transect-by-transect basis (e.g. survey condition covariates). The investigator should take care to make sure that the class of these variables (e.g. numeric, factor) is the entered in the same manner one wishes to analyze the data in (i.e. factor variables will be analyzed as categorical, numeric variables will be analyzed as continuous).

7. The next column, 'Distance' gives distances of observed animals from the survey platform. If distance data are binned, distances should be entered as factor variables; if continuous, `hierarchicalDS` assumes that the investigator has standardized all distances to the $(0, 1)$ interval (with 1 corresponding to a truncation distance).

8. Finally, the remaining columns represent individual level covariates associated with detected animal groups. The individual covariate 'Group' is required, and is assumed to occur in the column immediately after 'Distance.' Other individual covariates could be entered here as well, but are not required.

Okay, now that we have our data defined, let's continue by specifying some different observer options.

```
Obs.cov = array(0, dim = c(2, n.transects, 1))
Obs.cov[1, , ] = 1
n.obs.cov = 1
```

Here, we define a 3-D array, `Obs.cov`, whose first dimension is equal to two (for double observers), the second dimension is equal to the number of transects (recall that each transect-grid cell combination is entered in as a separate 'transect'), and whose third dimension is the number of observer-level covariates. Here, we have just one covariate, 'Seat,' so the third dimension is 1. After defining the array, we fill the array for the 'front seat' observer to have a value of 1.0. We also declare that there is one observer-level covariate by setting `n.obs.cov=1`.

The next thing we'll do is to specify an adjacency matrix, `Adj`, which describes the neighborgood structure associated with the particular spatial grid cell topology one is using. This is only needed if one wants to estimate spatially autocorrelated random effects. There are several tools that are provided to help the user construct such a matrix; in particular, the function `square_adj` constucts a square adjacency matrix (assuming a queen's move neighborhood), the function `rect_adj` constucts an adjacency matrix on a rectangle, and `rect_adj_RW2` constructs an RW2 adjacency matrix (which provides a greater degree of smoothing; see **?**). In practice, I have found it useful to construct these adjacency matrices on a rectangle, and then to delete rows and columns of that rectangle that are not actually in the survey area (in the likely case that the survey area isn't a perfect rectangle). Of course, one will usually also want to be projecting their study area from 3-D space to 2-D space using standard tools from geography; I personally find the `sp` package and related libraries (e.g. `rgeos,rgdal,raster,maptools`) to be quite helpful in this regard. For our particular case, we'll simply use a square adjacency matrix:

```
Adj = square_adj(sqrt(S))
```

Next, we'll establish a structure for modelling species misidentification (this step can be ignored if only conducting single species analysis, or when species misidentification is assumed negligible):

```
misID.mat = matrix(0, 2, 3)  # misID matrix
misID.mat[1, ] = c(1, -1, 2)
misID.mat[2, ] = c(-1, 3, -1)
misID.models = c(~1, ~1, ~1)
MisID = vector("list", max(misID.mat))
MisID[[1]] = 2  #parameters for getting it right
MisID[[2]] = 1
MisID[[3]] = 3  #parameters for getting it right
misID.symm = TRUE
```

The first thing we've done for our simulated dataset is to establish a $2x3$ matrix, where rows define true species (recall there are two species in our simulated dataset), and columns define observation types. Because species classification probabilities should sum to one for each species, we need to specify which clasification probability is obtained by subtraction. This is done by specifying a '-1' for specific entries of of `misId.mat`. In this example, we specify symmetric misclassification probabilities with the option `misID.symm=TRUE`. As such, the misclassification probability of misclassifying species 2 as species 1 ($\pi^{[1|2]}$) is set equal to the probability of misclassifying species 1 as species 2 ($\pi^{[2|1]}$). This specification helps render misclassification probabilities indentifiable (Conn *et al.*, Accepted) without having to resort to informative prior distributions. This also means we have to estimate one less classification probability; when using symmetric classification parameters our convention is to replace classification parameters set by constraint to -1 (so in this case we set the first column of the second row of misID.mat to -1). So, our specification means that we are estimating parameters associated with $\pi^{[1|1]}$, $\pi^{[3|1]}$, and $\pi^{[2|2]}$, and obtaining the rest through constraints. The non-negative entries of `misID.mat` provide an index to the entries of the formula vector, `misID.models`, as well as to the list entries for initial values. Here, we have simply specified intercept formula (type `? formula` for more information) for each of these species classification parameters. Note that these parameters are modeled with a multinomial logit link function and we could, in theory, specify more complicated formulae (e.g. making them a function of group size, etc.). However, these parameters are tend to be poorly estimated and imposing additional structure will likely only increase issues with identifiability. The other thing we have done here is to provide some initial values for these parameters via `MisID`. If we had considered a more complicated model for misclasification, each list entry would be a vector.

The next thing we'll do is to specify some details associated with our transects and tesselated study area:

```
Mapping = c(1:S)
Area.trans = rep(1, S)
n.bins = length(unique(Dat[, "Distance"]))
Area.hab = rep(1, S)
Bin.length = rep(1, n.bins)  #equal bin lengths
```

The first thing we've done is to define a vector called `Mapping`, which maps our transects to the grid cells in our tesselated study area. In the simulated example, we just have a one to one mapping between transects and grid cells, so it is easy to construct the `Mapping` vector. In general, the length of the `Mapping` vector is equal to the number of 'transects' (recalling that an actual transect that spans multiple grid cells is broken into multiple segments). For example, in a hypothetical study with three transects (1,2,3) that were conducted in grid cells 1, 3, and 10, then we'd have `Mapping=c(1,3,10)`. The second thing we've done here is to specify the proportion of area that each of our transects covers in relation to the total area of the grid cell. In the simple simulated

data example, our transect covers the entire area of the grid cell, so we simply have all entries of `Area.trans` equal one. If, instead, we had three transects that covered 20%, 30%, and 10% of their respective grid cells, this vector would be specified as `Area.trans=c(0.2,0.3,0.1)`. Within this code block, we have also specified the relative area of our grid cells through `Area.hab`. This vector can be used to account for grid cells that are of different size, or have different amounts of target habitat. Note that all modeling of abundance intensity occurs at `Area.trans=1`, so we wouldn't want to enter very large or very large numbers here (i.e., it's better to standardize Area.trans so that it has a mean near one). For distance data that are binned (which they are for our simulated example), the other things we have done here is to specify the number of distance bins via `n.bins`, as well as indicate the relative length of the each bin via `Bin.length`.

The next thing we've done when setting up analysis for our simulated dataset is to specify the habitat covariates and fixed effects models for spatial regression. We enter habitat covaraiates in the data frame `Hab.cov`, where each column specifies a separate habitat covariates, and the column name provides the name of the covariate (there should be S rows in the `Hab.cov` data frame). Here, we've simply provided the same easterly-westerly covariate used to generate the simulated data (which is hardwired in the `simulate_data` function). We also provide formulae for (1) the Poisson models for abundance intensity (via `Hab.pois.formula`), and, where zero inflation is modeled, (2) the Bernoulli model for abundance intensity [here, zero infation is modeled as in **?**]. We provide formula in a list object, since we can potentially have different regression models for each species (here, our list is of length 2 since we have 2 species). In particular, we allow the fixed effects model for the Poisson component of abundance intensity to include our easterly-westerly covariate (here, names 'Cov1'), and specify an intercept model for the the Bernoulli zero-inflation model.

```
Hab.cov = data.frame(rep(log(c(1:sqrt(S)/sqrt(S))), each = sqrt(S)))
# (covariate on abundance intensity same as used to simulate data)
colnames(Hab.cov) = c("Cov1")
Hab.pois.formula = vector("list", 2)
Hab.bern.formula = Hab.pois.formula
for (i in 1:2) {
    Hab.pois.formula[[i]] = ~Cov1
    Hab.bern.formula[[i]] = ~1
}
```

Our next little code chunk is relatively simple...we (1) specify that we're interested in estimating a detection function, specifying `detect=TRUE` (note that `detect=FALSE` can be specified for strip transects with detectability equal one), (2) specify a model for the detection function through `Det.formula` (note that the model is writted as a function of column names appearing in `Dat`), and (3) indicate the number of species being modeled:

```
detect = TRUE
Det.formula = ~Observer + Distance + Group
n.species = nrow(misID.mat)
```

Next, we specify prior distributions for individual covariates:

```
Cov.prior.parms = array(0, dim = c(n.species, 2, 1))
Cov.prior.parms[1, , 1] = c(2, 1)
Cov.prior.parms[2, , 1] = c(2, 1)
Cov.prior.fixed = matrix(0, n.species, dim(Cov.prior.parms)[3])
Cov.prior.pdf = Cov.prior.fixed
Cov.prior.pdf[, 1] = c("pois1", "pois1")
Cov.prior.n = matrix(2, 2, 1)
```

This part actually takes some care, and is the one place we need to specify somewhat informative priors, especially if initial parameter values are not provided. This is because vague priors can lead to outlandish initial covariate values when MCMC is initialized and make the data augmentation algorithm unstable. We start by initializing a 3-D array for individual covariate priors, `Cov.prior.parms`. In general, the dimension of this arrray should be ($n.species \times max.par \times n.cov$) where $n.cov$ gives the number of individual covariates nad $max.par$ gives the maximum number of prior parameters required for an individual covariate prior. In our case, we just have one individual covariate (group size), and have specified a zero truncated Poisson distribution for it via 'pois1' (see documentation for `hierarchical_DS` for alternative PMF and PDF alternatives for individual covariates). Note that group size should always use a zero truncated PMF to preclude the possibility of group sizes of 0! By convention, the zero truncated Poisson uses a Gamma(α, β) prior; since the zero-truncated Poisson is just a Poisson random variable + 1, our prior effectively has a mean of 3.0 for both species (note that data were simulated with a mean of 4.0 and 2.0 for species 1 and 2, respectively). In `hierarchicalDS`, we have the option of fixing the parameters of individual covariate distribution to initial values specified by the user by toggling elements of the `Cov.prior.fixed` matrix to be 1. Here, we set them all zero so that parameters associated with group size are estimated. The final thing we've done here is to specify the number of parameters associated with each covariate distribution by filling the matrix `Cov.prior.n`, where the rows of `Cov.prior.n` indicate species, and the columns indicate the number of individual covariates. In this case, we've filled the matrix with the number '2,' since the prior distributions for groups size for each species have 2 parameters. Often, group size will be the only individual covariate of interest.

The next thing we've done is to define a number of other `hierarchicalDS` options:

```
point.ind = TRUE  #include point independence
spat.ind = FALSE  #do not make spatially independent
fix.tau.nu = FALSE
srr = TRUE
srr.tol = 0.5
misID = TRUE
last.ind = FALSE
cor.const = FALSE
grps = TRUE
post.loss = FALSE
```

Specifically, we've turned on point independence with `point.ind=TRUE`, so that a correlation parameter for double observer detections will be estimated on the probit scale (the range of the correlation parameter is restricted to [-0.95,0.95] to avoid numerical errors); we've enabled estimation of spatially autocorrelated random effects by setting `spat.ind=FALSE`; and we've enabled estimation of τ_ν, which allows for estimation of overdispersion relative the Poisson distribution (above and beyond that explained by fixed effects and spatial random effects). By setting `srr=TRUE`, we enable a reduced dimension version of spatial random effects, where the reduced rank effects are defined to be orthogonal to the design matrix associated with fixed effects; only positive eigenvectors associated with the spectral decomposition of the spatial process with eigenvalues greater than `srr.tol` are modeled. This approach follows recent developments in the statistics literature desribing spatially restricted regression (e.g. Hughes & Haran, 2012; Reich *et al.*, 2006), and allow for a smoother, stabler implementation that preserves the primacy of fixed effects for explaining variation in abundance. Here, we have set `srr.tol=0.5`; for a smoother surface, one might consider increasing this value. The option `misID=TRUE` turns on species misidentification. The next two options, `last.ind` and `cor.const` pertain to point independence model only. The typical setup is to have observer independence assumed in the first bin, and increasing linearly with distance (`last.ind=FALSE`). However, we've encountered certain cases where double observers have slightly different views out of side of an airplane which are accentuated the closer an animal is to the trackline. For these, cases we've allowed (`last.ind=TRUE`), which assumes observers obtain indpendent detections at maximal distances. The option `cor.const=TRUE` will impose even more structure; in particular, when `last.ind=FALSE`, posterior samples are limited to positive values for the correlation parameter, and when `last.ind=FALSE`, the correlation parameter will be restricted to negative values. The next option, `grps`, indicates whether group size should be modeled (if `grps=FALSE`, it is assumed that all detections are of single animals). Finally, `post.loss` toggles whether or not a posterior loss criterion (sensu Gelfand & Ghosh, 1998) should be calculated for model comparison. This criterion is somewhat time consuming to compute, but is only computed for iterations that are stored (so computation time is reduced by storing less iterations; see the Control$thin option below).

The next thing we define is the `M` matrix, which defines the maximum number of animals for each species that is allowed in a given transect. This is to help set the dimension of the data augmentation matrix, but has practical ramifications for inference. Optimally, we'd set this upper bound high enough so that we would never hit it, but low enough to avoid extraneous calculations (covariate values are simulated for all rows $\leq M$). In practice, some trial and error/tuning may be needed here (`hierarchicalDS` outputs warning messages when bounds are hit). Of course, the higher detection probabilities are, the lower `M` will need to be; tightening up model structure will (eliminating overdispersion and/or random effects, providing informative priors for species classification probabilities, etc.) will also, decrease the likelihood of hitting up against upper bounds. In our experience a limited amount of flirtation with the upper bound is sometimes acceptable, particularly at the initial stages of MCMC estimation, when the Markov chain is still converging to the posterior distribution (especially if vague priors are assumed and initial value generation is automated). For our example, we've initialized `M` as a $(2 \times T)$ matrix (T being the number of transects), and set values equal to 10 times the true abundance (increasing smaller `M` values to be a minimum of 50):

```
M = t(Out$G.true * 10)
M[which(M < 30)] = 50
```

The next thing we'll do is specify some MCMC options through the list object, `Control`:

```
Control = list(iter = 21000, burnin = 1000, thin = 10, MH.cor = 0.2, MH.nu = matrix(0.2,
    2, S), MH.misID = matrix(0.1, 3, 1), RJ.N = matrix(rep(5, S * n.species),
    n.species, S), adapt = 100, iter.fix.N = 100)
```

Here, we've indicated the number of MCMC iterations through `iter`, the number of iterations to discard as a burn-in `burnin`, the number of iterations to store values for (posterior samples will be stored for one out of every `thin` iterations). Although there is no real advantage to discarding MCMC output from an analysis standpoint, the sheer volume of values stored can be quite large for `hierarchicalDS`, so one can run into memory limitations if not careful (limiting the number of stored posterior samples to a reasonable number can fix this problem). We then specify a few Metropolis-Hastings tuning parameters; `MH.cor` controls the length of a uniform proposal kernel for the correlation parameter (if double observers with point independence); similarly `MH.nu` is a $(2 \times \#sampledcells)$ matrix providing the length of the uniform proposal kernel for Metropolis-Hastings updates of ν, the log of abundance intensity; `MH.misID` give the length of the uniform kernel proposal distribution for misidentification parameters on the multinomial-logit scale (where the number of rows is equal to the number of misclassification probabilites that are modeled, and the number of columns is the maximum number of parameters associated with a misclassfication model). Next, `RJ.N` specify control values influencing reversible

jump updates of the number of animals in the surveyed area of a given cell. In particular, it should be a matrix with number of rows equal to the number of species, and number of columns equal to the number of grid cells that are actually sampled. The values making up the matrix control the maximum number of latent (unobserved animals) that can be proposed to be added or subtracted to the local population at every iteration of the Markov chain. For large populations, these should ideally be set larger than for small populations (here, we've set them all to 5). Ideally, acceptance rates for all of these updates (Metropolis-Hastings and reversible jump) would be in the 0.3-0.4 range to promote good mixing and fast convergence to the posterior distribution (Gelman *et al.*, 2004). An adapt phase that automatically adjusts these initial inputes to try to achieve optimal acceptance rates can be implemented by setting the number of adapt iterations > 0 (here we've set `adapt=100`) and `adapt=TRUE` (see below). In practice, it may be difficult to achieve globally optimal acceptance rates unless the `adapt` phase is quite long; however, if poor initial values are provided/generated, the adapt phase may be somewhat unstable. Tuning these values 'by hand' may sometimes be necessary.

To start the Markov chain in a reasonable quadrant of parameter space, we next provide some initial values for certain parameters using the `Inits` list object.

```
hab.pois = matrix(0, n.species, 2)  #covariates are intercept, index
hab.pois[1, 1:2] = c(log(50), 0)
hab.pois[2, 1:2] = c(log(10), -2)
hab.bern = matrix(0, n.species, 1)
hab.bern[, 1] = c(1, 1)
Inits = list(hab.pois = hab.pois, hab.bern = hab.bern, tau.nu = c(500, 500),
    MisID = MisID)
```

Although we could provide more values than those included here (type `?hierarchical_DS` for a full list), these few seem reasonably sufficient. In addition to the initial values for species classification parameters (`MisID`, described previously), we've also defined initial values for the precision (recall that precision is 1/variance) of the $\boldsymbol{\nu}$ parameters through `tau.nu` (there are two values, one for each species), and for fixed effects of habitat regression models. For the Poisson model for abundance, recall that we specified a linear relationship with a habitat covariate, so we need to provide initial values for each parameter (and separate parameters for each species) - we start these slightly off from their true values. The Bernoulli submodel for zero-inflation was simply specified as an intercept-only model so we only require one initial value for each species.

The next thing on our task list is to provide a list of prior distribution parameters through the list object `Prior.pars`:

```
misID.mu = vector("list", max(misID.mat))
misID.sd = misID.mu
misID.mu[[1]] = 0
```

```
misID.mu[[2]] = 0
misID.mu[[3]] = 0
misID.sd[[1]] = 1.75
misID.sd[[2]] = 1.75
misID.sd[[3]] = 1.75
Prior.pars = list(a.eta = 1, b.eta = 0.01, a.nu = 1, b.nu = 0.01, beta.tau = 0.01,
    misID.mu = misID.mu, misID.sd = misID.sd)
```

Here, we enter conjugate Gamma($a.eta, b.eta$) priors for the precision of spatial random effects Gamma($a.nu, b.nu$) priors for the precision of ν parameters, and a Normal($0, 1/beta.tau$) prior for fixed effect regression parameters. We also put normal prior distributions on the parameters associated with misclassification models. Here, we've assumed parameters all have Normal($0, 1.75$) priors in the multinomial logit scale (this results in an approximately uniform distribution on the real scale).

There's just a few more things to specify before we can run our model. This includes specifying that we want to include an adapt phase, and the we want to estimate zero inflation (currently, if `spat.ind=FALSE`, spatial random effects will be estimated for both the Poisson and Bernoulli submodels). Finally, we provide a seed to the random number generator so that we can exactly duplicate our analysis in the future (should we wish)/

```
adapt = TRUE
ZIP = TRUE
set.seed(8327329)  #chain1
```

Now, we can call the function `hierarchical_DS` using all predefined options. Note, however, that this particular analysis takes ≈ 40 hours. You can load the resulting dataset by simply typing `data(sim_data)`!

```
Out = hierarchical_DS(Dat = Dat, Adj = Adj, Area.hab = Area.hab, Mapping = Mapping,
    Area.trans = Area.trans, Observers = Observers, Bin.length = Bin.length,
    Hab.cov = Hab.cov, Obs.cov = Obs.cov, n.obs.cov = n.obs.cov, Hab.pois.formula = Hab.pois
    Hab.bern.formula = Hab.bern.formula, detect = detect, Det.formula = Det.formula,
    Cov.prior.pdf = Cov.prior.pdf, Cov.prior.parms = Cov.prior.parms, Cov.prior.fixed = Cov.
    Cov.prior.n = Cov.prior.n, pol.eff = NULL, ZIP = ZIP, point.ind = point.ind,
    spat.ind = spat.ind, fix.tau.nu = fix.tau.nu, srr = srr, srr.tol = srr.tol,
    misID = misID, last.ind = last.ind, cor.const = cor.const, Inits = Inits,
    grps = grps, M = M, Control = Control, adapt = adapt, Prior.pars = Prior.pars,
    misID.mat = misID.mat, misID.models = misID.models, misID.symm = misID.symm,
    post.loss = post.loss)

##
##  Beginning adapt phase
##
##  11  eigenvectors selected for spatially restricted regression
```

```
##
##  11  eigenvectors selected for spatially restricted regression
##
##  12  eigenvectors selected for spatially restricted regression
##
##  12  eigenvectors selected for spatially restricted regression
## iteration  1  of  100  completed
##
## Approximate time till completion:  0  hours
##
##  total elapsed time:  7.05807411670685  minutes
##
##  Beginning MCMC phase
##
##  11  eigenvectors selected for spatially restricted regression
##
##  11  eigenvectors selected for spatially restricted regression
##
##  12  eigenvectors selected for spatially restricted regression
##
##  12  eigenvectors selected for spatially restricted regression
## iteration  1  of  21000  completed
##
## Approximate time till completion:  24.6  hours
## iteration  1001  of  21000  completed
## iteration  2001  of  21000  completed
## iteration  3001  of  21000  completed
## iteration  4001  of  21000  completed
## iteration  5001  of  21000  completed
## iteration  6001  of  21000  completed
## iteration  7001  of  21000  completed
## iteration  8001  of  21000  completed
## iteration  9001  of  21000  completed
## iteration  10001  of  21000  completed
## iteration  11001  of  21000  completed
## iteration  12001  of  21000  completed
## iteration  13001  of  21000  completed
## iteration  14001  of  21000  completed
## iteration  15001  of  21000  completed
## iteration  16001  of  21000  completed
## iteration  17001  of  21000  completed
## iteration  18001  of  21000  completed
## iteration  19001  of  21000  completed
## iteration  20001  of  21000  completed
##
```

```
##  total elapsed time:  2441.22804905176  minutes
```

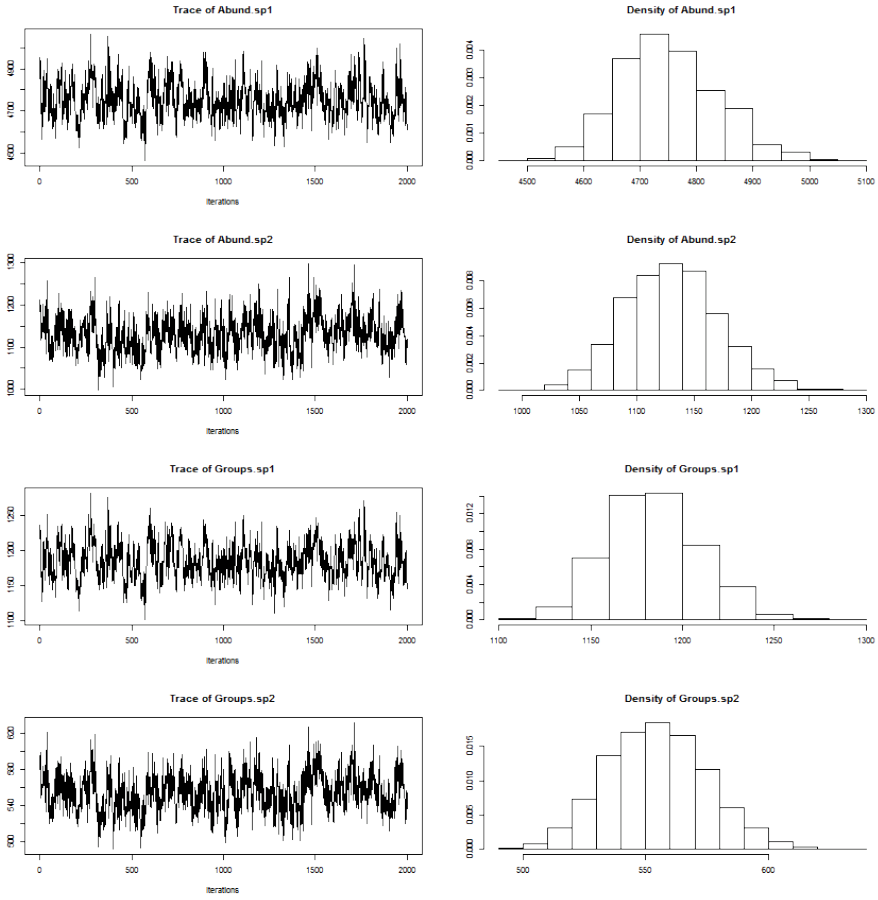
Let's take a look at the structure of the output list produced after a run by `hierarchicalDS`:

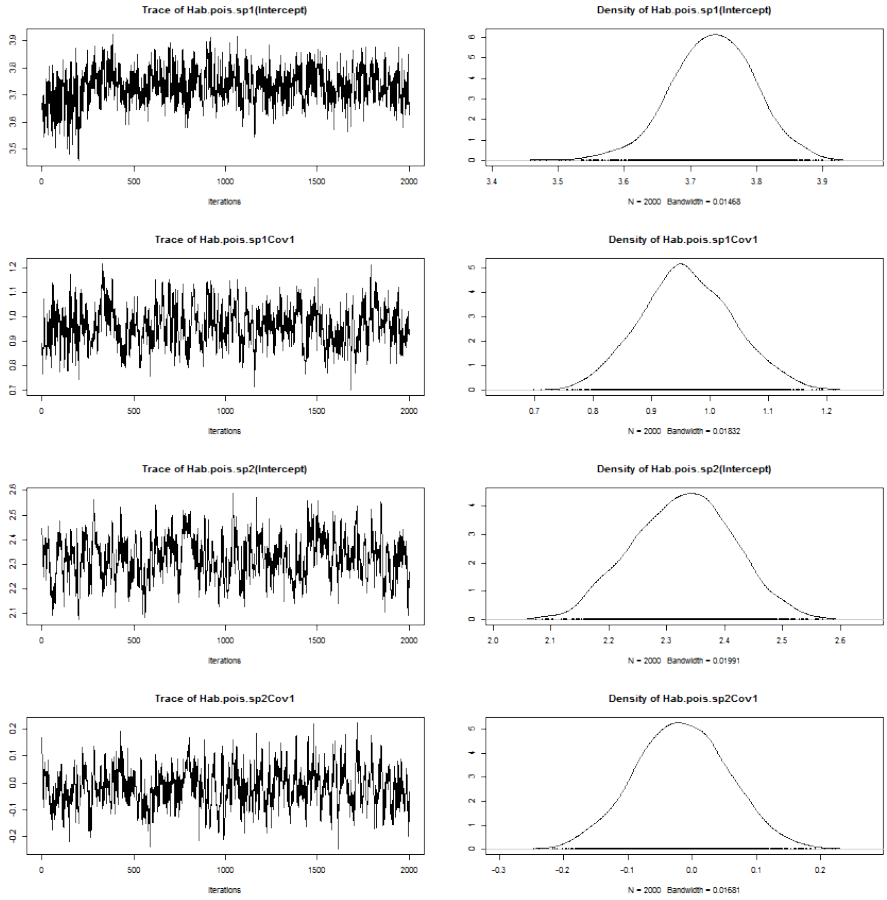
```
names(Out)

## [1] "Post"     "MCMC"     "Accept"   "Control"  "Obs.N"    "Pred.N"
## [7] "Obs.det"  "Pred.det"
```

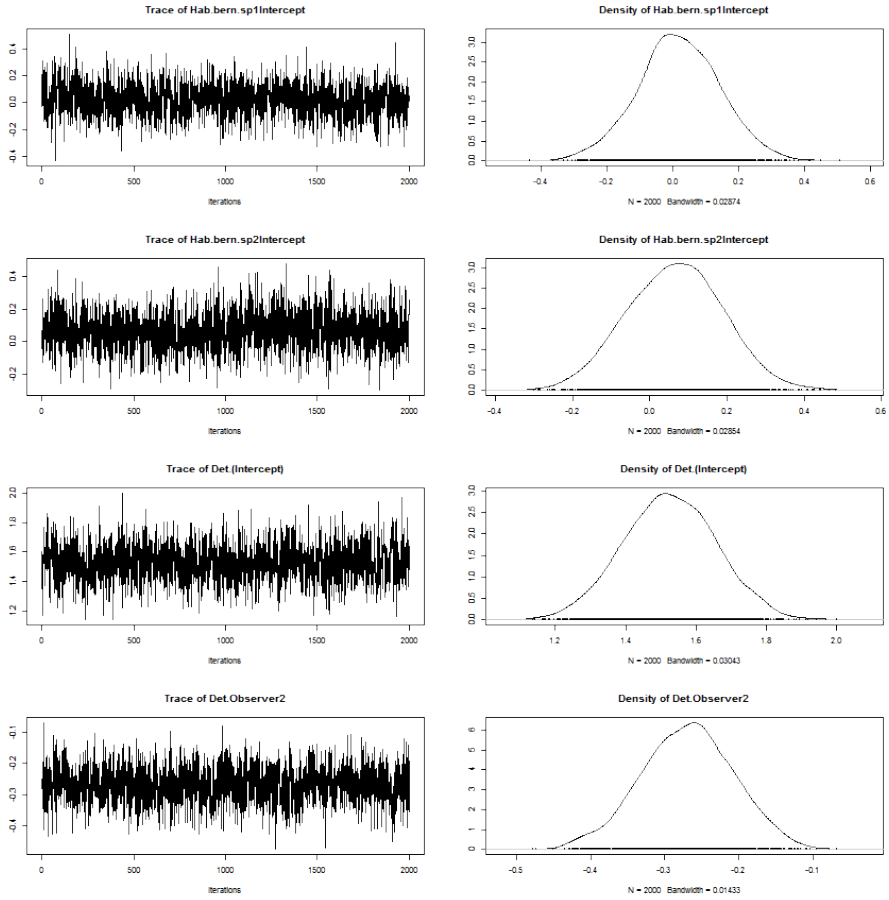
The first item in the list, `Out$Post`, is itself a list that includes objects `G` and `N`, which are each 3D arrays holding posterior samples for the number of groups (`G`) and total number of individuals (`N`). The dimension of each of these objects is ($\#species \times \#iterations \times S$). These objects can thus be used to plot posterior maps of abundance (see below). The second object in our list is `Out$MCMC` which is actually a `coda` object holding posterior samples of most other parameters of interest. By loading the `coda` R package, we can then bring to bear many existing procedures for MCMC diagnostics and plots. Next, `Out$Accept` holds acceptance numbers for parameters and latent variables subject to accept/reject steps (e.g. Metropolis-Hastings or reversible jump). This can be useful for further tuning of MCMC. Similarly, `Out$Control` returns the final value of the `Control` used in estimation (these values are often adjusted during the adapt phase). The next two values `Obs.N` and `Pred.N` are soon to be deprecated. Finally, `Obs.det` and `Pred.det` hold observed and predicted detections that are used by the `post_loss` function to calculate a posterior loss statistic for model comparison; these are only filled if `post_loss=TRUE` when calling `hierarchicalDS`.

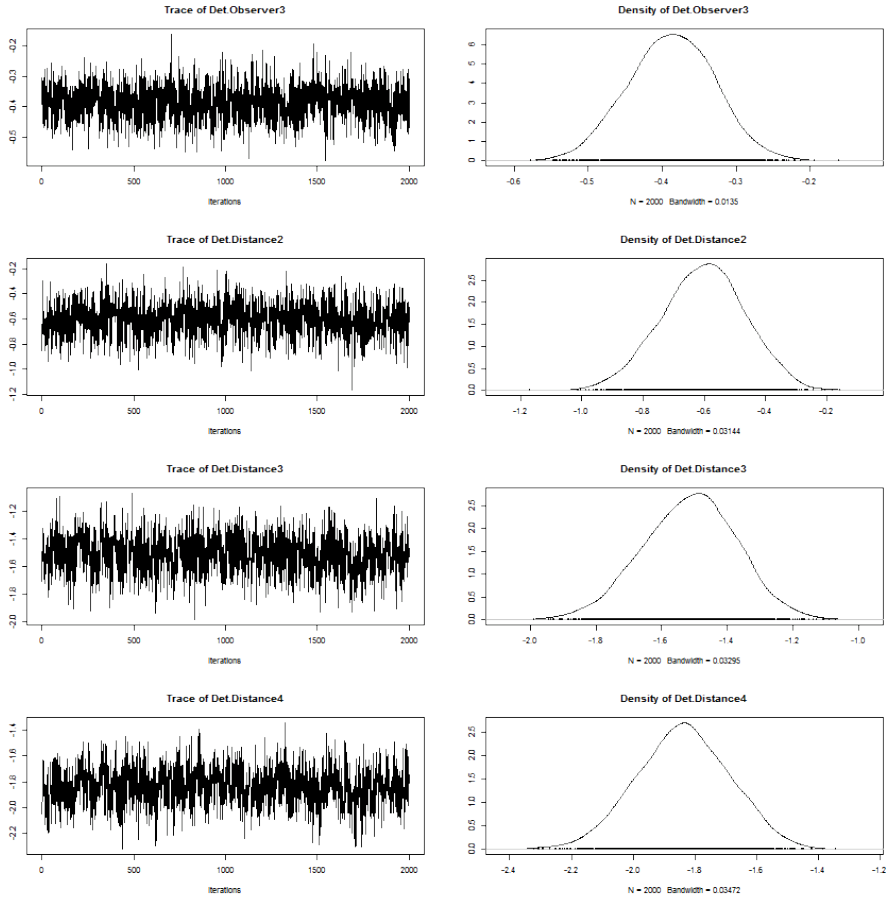
Let's call a few functions to visualize out output. To get `coda`-like output (i.e. trace and kernel density plots), we can simply do the following:

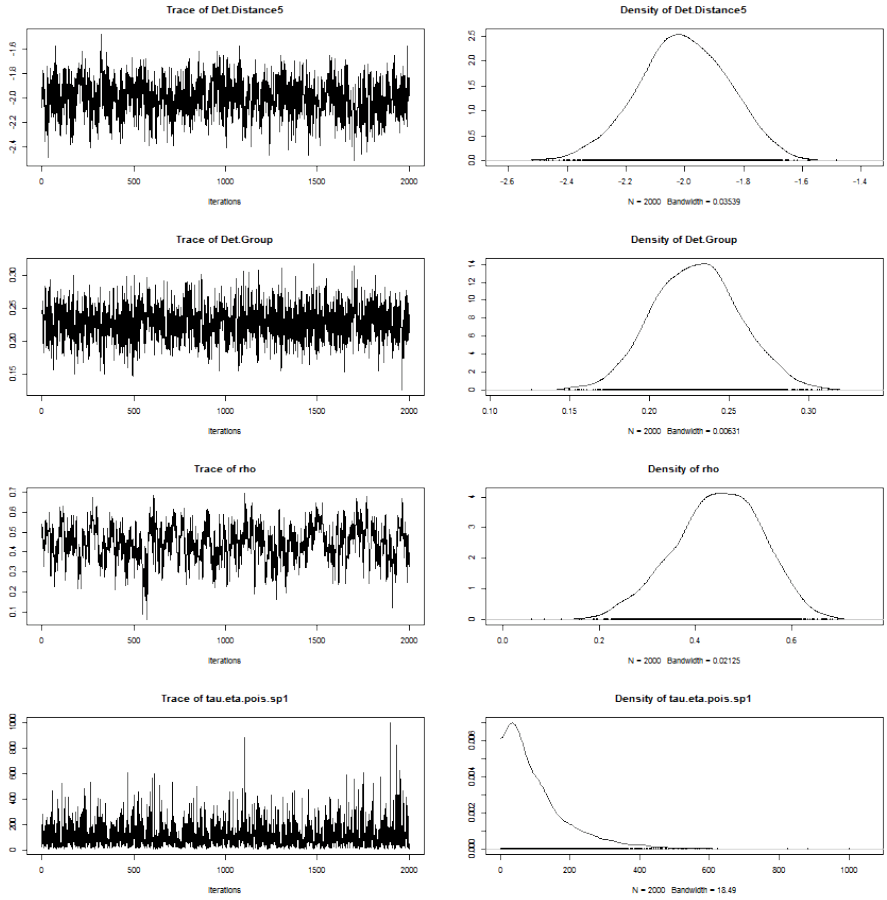
```
plot(Out$MCMC)
```

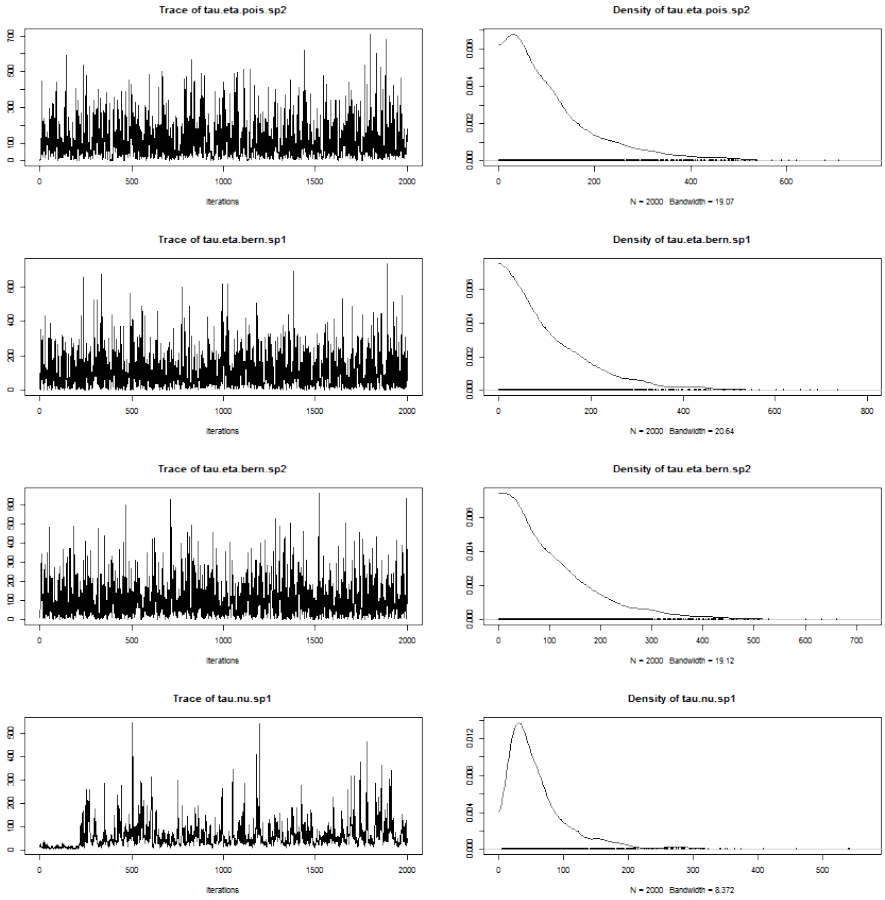

Trace of Abund.sp1
Density of Abund.sp1
Trace of Abund.sp2
Density of Abund.sp2
Trace of Groups.sp1
Density of Groups.sp1
Trace of Groups.sp2
Density of Groups.sp2
Iterations

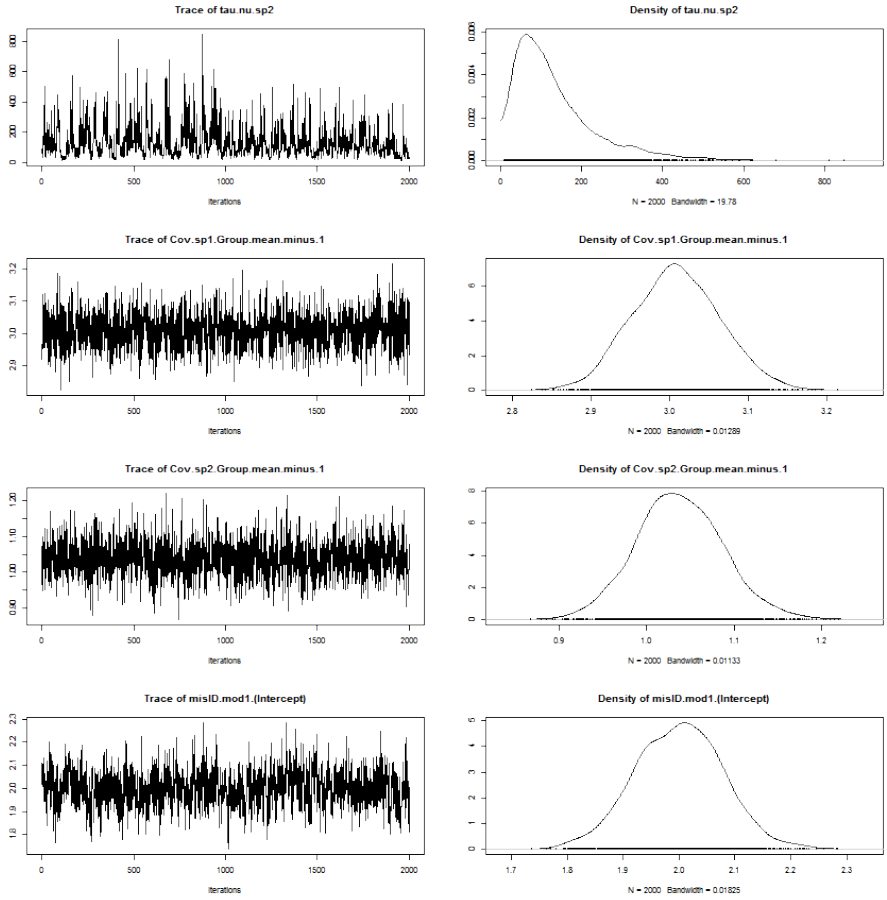

Trace of Hab.pois.sp1(Intercept)
Density of Hab.pois.sp1(Intercept)
Iterations
N = 2000 Bandwidth = 0.01468
Trace of Hab.pois.sp1Cov1
Density of Hab.pois.sp1Cov1
Iterations
N = 2000 Bandwidth = 0.01832
Trace of Hab.pois.sp2(Intercept)
Density of Hab.pois.sp2(Intercept)
Iterations
N = 2000 Bandwidth = 0.01991
Trace of Hab.pois.sp2Cov1
Density of Hab.pois.sp2Cov1
Iterations
N = 2000 Bandwidth = 0.01681

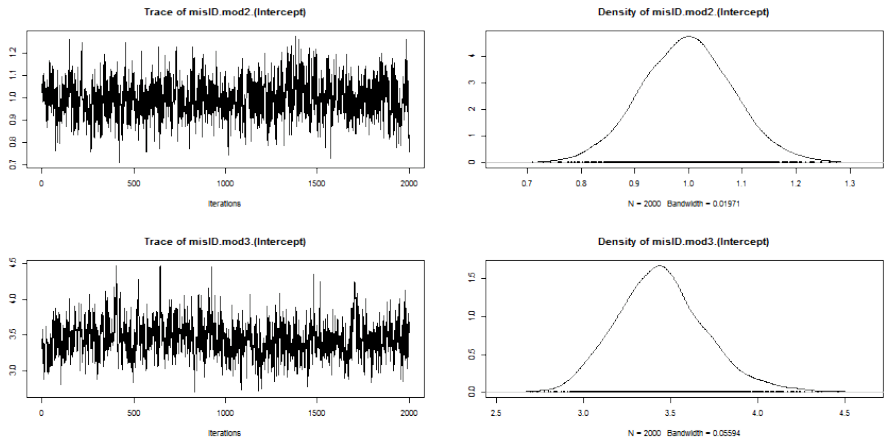

Trace of Hab.bern.sp1Intercept
Density of Hab.bern.sp1Intercept
Iterations
N = 2000 Bandwidth = 0.02874
Trace of Hab.bern.sp2Intercept
Density of Hab.bern.sp2Intercept
Iterations
N = 2000 Bandwidth = 0.02854
Trace of Det.(Intercept)
Density of Det.(Intercept)
Iterations
N = 2000 Bandwidth = 0.03043
Trace of Det.Observer2
Density of Det.Observer2
Iterations
N = 2000 Bandwidth = 0.01433


Trace of Det.Observer3
Density of Det.Observer3
Iterations
N = 2000 Bandwidth = 0.0135
Trace of Det.Distance2
Density of Det.Distance2
Iterations
N = 2000 Bandwidth = 0.03144
Trace of Det.Distance3
Density of Det.Distance3
Iterations
N = 2000 Bandwidth = 0.03295
Trace of Det.Distance4
Density of Det.Distance4
Iterations
N = 2000 Bandwidth = 0.03472


Trace of Det.Distance5
Density of Det.Distance5
N = 2000 Bandwidth = 0.03539
Trace of Det.Group
Density of Det.Group
N = 2000 Bandwidth = 0.00631
Trace of rho
Density of rho
N = 2000 Bandwidth = 0.02125
Trace of tau.eta.pois.sp1
Density of tau.eta.pois.sp1
N = 2000 Bandwidth = 18.49
Iterations


Trace of tau.eta.pois.sp2
Density of tau.eta.pois.sp2
Iterations
N = 2000 Bandwidth = 19.07
Trace of tau.eta.bern.sp1
Density of tau.eta.bern.sp1
Iterations
N = 2000 Bandwidth = 20.64
Trace of tau.eta.bern.sp2
Density of tau.eta.bern.sp2
Iterations
N = 2000 Bandwidth = 19.12
Trace of tau.nu.sp1
Density of tau.nu.sp1
Iterations
N = 2000 Bandwidth = 8.372

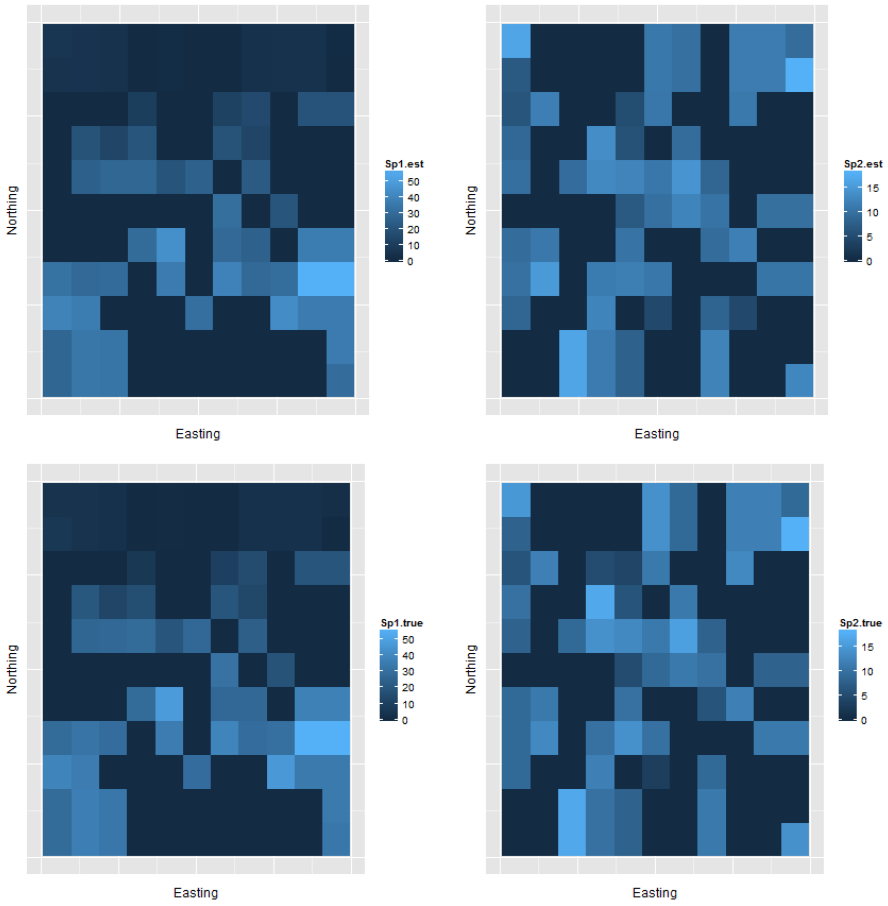

Trace of tau.nu.sp2
Density of tau.nu.sp2
Iterations
N = 2000 Bandwidth = 19.78
Trace of Cov.sp1.Group.mean.minus.1
Density of Cov.sp1.Group.mean.minus.1
Iterations
N = 2000 Bandwidth = 0.01289
Trace of Cov.sp2.Group.mean.minus.1
Density of Cov.sp2.Group.mean.minus.1
Iterations
N = 2000 Bandwidth = 0.01133
Trace of misID.mod1.(Intercept)
Density of misID.mod1.(Intercept)
Iterations
N = 2000 Bandwidth = 0.01825



In our enviable position of having the 'true' answer regarding abundance, we can thus see that true abundances (species 1: 4633, species 2: 1118) are in the 'meat' of estimated posterior distibutions. We could also get a table of posterior moments and credible intervals by using the function `table.mcmc`. I used to have a plotting function for producing maps of output, but dependencies were unstable (e.g. `ggplot2`) and I removed it. Here's some code that can help in plotting, however.

```
library(sp)
library(raster)
r1 = raster(nrows = sqrt(S), ncol = sqrt(S))
Tmp <- rasterToPolygons(r1, na.rm = FALSE)
iter.start = 1
New.dat = matrix(0, S, 2 * n.species)
for (isp in 1:n.species) {
    New.dat[, isp] = apply(Out$Post$G[isp, iter.start:(dim(Out$Post$G)[2]),
        ], 2, "mean")
```

```
}
New.dat[, 3:4] = Sim$G.true
colnames(New.dat) = c("Sp1.est", "Sp2.est", "Sp1.true", "Sp2.true")
Tmp@data = cbind(Tmp.grid@data, New.dat)
library(ggplot2)
library(plyr)
library(grid)
Tmp@data$id = rownames(Tmp@data)
tmp1 <- fortify(Tmp, region = "id")
tmp2 <- join(tmp1, Tmp@data, by = "id")
new.colnames = colnames(tmp2)
new.colnames[1:2] = c("Easting", "Northing")
colnames(tmp2) = new.colnames
pushViewport(viewport(layout = grid.layout(2, 2)))
tmp.theme = theme(axis.ticks = element_blank(), axis.text = element_blank())
p1 = ggplot(tmp2) + aes(Easting, Northing, fill = Sp1.est) + geom_raster() +
    tmp.theme
print(p1, vp = viewport(layout.pos.row = 1, layout.pos.col = 1))
p2 = ggplot(tmp2) + aes(Easting, Northing, fill = Sp2.est) + geom_raster() +
    tmp.theme
print(p2, vp = viewport(layout.pos.row = 1, layout.pos.col = 2))
p3 = ggplot(tmp2) + aes(Easting, Northing, fill = Sp1.true) + geom_raster() +
    tmp.theme
print(p3, vp = viewport(layout.pos.row = 2, layout.pos.col = 1))
p4 = ggplot(tmp2) + aes(Easting, Northing, fill = Sp2.true) + geom_raster() +
    tmp.theme
print(p4, vp = viewport(layout.pos.row = 2, layout.pos.col = 2))
```



5 Speeding up analyses

Clearly, the time it took to run our simulated data analysis (40 hours) is not ideal. Profiling the code, it's clear that the greatest time sinks are (i) data augmentation associated with RJMCMC, and (ii) updating species and species classification parameters. The time associated with (i) can be reduced by decreasing `M` values (the maximum number of animals in the sampled area of each sampled cell). The trick, however, is decreasing these to a level where the posterior distribution doesn't bump up against the upper bound. This is less likely to happen when detection probability is high, so one effective strategy is to decrease the distance threshold for analysis so that the detection curve is relatively high for all modeled distances. In our simulated example the total value of `M` across all sampled sites and species was 21,820 - this requires considerable computation to update latent covariates and parameters at each MCMC iteration. Similarly, if species classification rates are high enough, turning off estimation

of species misidentification (i.e. by setting `misID=FALSE`) would dramatically improve computation speed.

6 Troubleshooting

My general strategy for fitting models using `hierarchicalDS` is to start simply and build up. For instance, one probably wants to make sure the model is behaving reasonably before adding in data-hungry features such as estimation of spatial random effects and zero inflation. Analysts should also take care to fit models that do not extrapolate 'past the range of observed data.' With a single predictive covariate, this isn't too difficult as one can simply impose a 'cap' on the covariate such that it can't take more extreme values in unsampled areas than it does in sampled areas. However, it can be more difficult to diagnose when there are multiple covariates. Since abundance intensity is modeled on the log scale, predictions past the range of observed data can easily lead to overpredictions in unsampled areas that are orders of magnitude greater than they should be, simply because of over-extrapolation.

Currently, there are only limited tools for detecting problems and mismatches between hierarchicalDS function inputs; for instance, inputting a vector that is too short or too long. Although I am still adding to the list of checks (and appreciate user input), some ability to troubleshoot (e.g., by downloading and stepping through source code) would be useful on the part of the analyst.

References

Borchers, D. L., Laake, J. L., Southwell, C. & Paxton, C. G. M. (2006). Accomodating unmodeled heterogeneity in double-observer distance sampling surveys. *Biometrics*, **62**, 372–378.

Conn, P. B., Laake, J. L. & Johnson, D. S. (2012). A hierarchical modeling framework for multiple observer transect surveys. *PLoS ONE*, **7**, e42294.

Conn, P. B., McClintock, B. T., Cameron, M. F., Johnson, D. S., Moreland, E. E. & Boveng, P. L. (Accepted). Accommodating species identification errors in transect surveys. *Ecology*.

Dempster, A. P., Laird, N. M. & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B (Methodological)*, **39**, 1–38.

Gelfand, A. E. & Ghosh, S. (1998). Model choice: A minimum posterior predictive loss approach. *Biometrika*, **85**, 1–11.

Gelman, A., Carlin, J. B., Stern, H. S. & Rubin, D. B. (2004). *Bayesian Data Analysis, 2nd Edition*. Chapman and Hall, Boca Raton.

Hughes, J. & Haran, M. (2012). Dimension reduction and alleviation of confounding for spatial generalized mixed models. *ArXiv*, ***1101.6649v1 [stat.ME]***.

Laake, J. & Borchers, D. (2004). Methods for incomplete detection at distance zero. *Advanced Distance Sampling* (eds. S. Buckland, D. Anderson, K. Burnham, J. Laake, D. Borchers & L. Thomas), pp. 108–189. Oxford University Press, Oxford, U.K.

Reich, B., Hodges, J. & Zadnik, V. (2006). Effects of residual smoothing on the posterior of the fixed effects in disease-mapping models. *Biometrics*, **62**, 1197–1206.