Fitting occupancy models with `spOccupancy`

Jeffrey W. Doser

September 21, 2021

Contents

0.1 Introduction

This vignette provides worked examples and explanations on fitting single species and multispecies occupancy models available in the `spOccupancy` R package. We will provide step by step examples on how to fit the following models:

1. Occupancy model using `PGOcc`.
2. Spatial occupancy model using `spPGOcc`.
3. Multispecies occupancy model using `msPGOcc`.
4. Spatial multispecies occupancy model using `spMsPGOcc`.
5. Integrated occupancy model using `intPGOcc`.
6. Spatial integrated occupancy model using `spIntPGOcc`.

Here we will provide only a brief description of each model, with full statistical details of each model in the Gibbs sampler vignette. We will also show how `spOccupancy` provides functions for posterior predictive checks as a Goodness of Fit assessment, model comparison and assessment using the Widely Applicable Information Criterion (WAIC), and out of sample predictions using standard R helper functions (i.e., `predict`).

To get started, we load the `spOccupancy` package, as well as the `coda` package, which we will use for some MCMC diagnostics. We will also use the `stars` and `ggplot2` packages to create some very basic plots of our results.

```
library(spOccupancy)
library(coda)
library(stars)
```

```
Loading required package: abind

Loading required package: sf

Linking to GEOS 3.6.2, GDAL 2.2.3, PROJ 4.9.3
```

```
library(ggplot2)
```

0.2 Example data set: Foliage-gleaning birds at Hubbard Brook

As an example data set throughout this vignette, we will use data from twelve foliage-gleaning birds collected from point count surveys at Hubbard Brook Experimental Forest (HBEF) in New Hampshire, USA. Specific details on the data set are available on the Hubbard Brook website and Doser et al. (2021). The data are provided in the `spOccupancy` package and are loaded with `data(hbef2015)`. Some brief information on the data collection protocol and the species included in the data set are found via `help(hbef2015)`.

```
data(hbef2015)
str(hbef2015)
```

```
List of 4
 $ y       : num [1:12, 1:373, 1:3] 0 0 0 1 0 1 1 0 0 0 ...
  ..- attr(*, "dimnames")=List of 3
  .. ..$ : chr [1:12] "AMRE" "BAWW" "BHVI" "BLBW" ...
  .. ..$ : chr [1:373] "1" "2" "3" "4" ...
  .. ..$ : chr [1:3] "1" "2" "3"
 $ occ.covs: num [1:373, 1:2] -0.889 -0.765 -0.413 -0.14 -0.13 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "Elevation" "Elevation.2"
 $ det.covs:List of 3
  ..$ day  : num [1:373, 1:3] -1.62 -1.62 -1.62 -1.62 -1.62 ...
  ..$ tod  : num [1:373, 1:3] -1.565 -1.378 -1.084 -0.79 -0.549 ...
  ..$ day.2: num [1:373, 1:3] 2.61 2.61 2.61 2.61 2.61 ...
 $ coords  : num [1:373, 1:2] 280000 280000 280000 280001 280000 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:373] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "Easting" "Northing"
```

The object `hbef2015` is a list comprised of the detection-nondetection data (`y`), covariates on the occupancy portion of the model (`occ.covs`), covariates on the detection portion of the model (`det.covs`), and the spatial coordinates of each site for use in spatially-explicit models. This list is in the exact format required for input to `spOccupancy` model functions. `hbef2015` contains data on 12 species in the three-dimensional array `y`. Here we will use data on the Ovenbird (OVEN) to display single species models, so we next subset the `hbef2015` list to only include data from OVEN in a new object `ovenHBEF`.

```r
sp.names <- attr(hbef2015$y, "dimnames")[[1]]
ovenHBEF <- hbef2015
ovenHBEF$y <- ovenHBEF$y[sp.names == "OVEN", , ]
table(ovenHBEF$y)
```

```

  0   1 
569 539 
```

We see OVEN is detected at around half of all site-replicate combinations.

0.3 Single species occupancy models

0.3.1 Basic model description

Let z_j be the true presence (1) or absence (0) of a species at site j, with $j = 1, \dots, J$. We assume this latent occurrence process arises from a Bernoulli process following

$$\begin{aligned} &z_j \sim \text{Bernoulli}(\psi_j), \\ &\text{logit}(\psi_j) = \boldsymbol{x}_j' \cdot \boldsymbol{\beta}, \end{aligned} \tag{1}$$

where ψ_j is the probability of occurrence at site j, which is a function of site-specific covariates $\boldsymbol{X}$ and a vector of regression coefficients ($\boldsymbol{\beta}$).

We do not directly observe z_j and rather we observe an imperfect representation of the latent occurrence process. Let $y_{j,k}$ be the observed detection (1) or nondetection (0) of a species of interest at site j during replicate k for each of $k = 1, \dots, K_j$ replicates at each site j. We envision the detection-nondetection data as arising from a Bernoulli process conditional on the true latent occurrence process:

$$\begin{aligned} &y_{j,k} \sim \text{Bernoulli}(p_{j,k} \cdot z_j), \\ &\text{logit}(p_{j,k}) = \boldsymbol{v}_{j,k}' \cdot \boldsymbol{\alpha}, \end{aligned} \tag{2}$$

where $p_{j,k}$ is the probability of detecting a species at site j during replicate k (given it is present at site j), which is a function of site and replicate specific covariates $\boldsymbol{V}$ and a vector of regression coefficients ($\boldsymbol{\alpha}$).

To complete the Bayesian specification of the model, we assign multivariate normal priors for the occurrence ($\boldsymbol{\beta}$) and detection ($\boldsymbol{\alpha}$) regression coefficients. To yield an efficient implementation of the occupancy model using a logit link function, we use Polya-Gamma data augmentation (Polson, Scott, and Windle 2013), which is described in depth in the Gibbs sampler vignette (**P**olya-**G**amma is where the `PG` comes from in all `spOccupancy` model fitting functions).

0.3.2 Fitting single species occupancy models with `PGOcc`

The `PGOcc` function fits single species occupancy models using Polya-Gamma latent variables, which should make it faster than other implementations of occupancy models using a logit link function (Clark and Altwegg 2019, @polson2013). `PGOcc` has the following arguments:

```r
PGOcc(occ.formula, det.formula, data, starting, n.samples, priors,
      n.omp.threads = 1, verbose = TRUE, n.report = 100,
      n.burn = round(.10 * n.samples), n.thin = 1, ...)
```

The first two arguments, `occ.formula` and `det.formula`, use standard R model syntax to denote the covariates included in the occurrence and detection portions of the model, respectively. Only the right hand side of the formulas are included. Random intercepts can be included in both the occurrence and detection portions of the occupancy model using `lme4` syntax (Bates et al. 2015). The names of variables given in the formulas should correspond to those found in `data`, which is a list consisting of the following tags: `y` (detection-nondetection data), `occ.covs` (occurrence covariates), `det.covs` (detection covariates). `y` should be stored as a sites x replicate matrix, `occ.covs` as a matrix or data frame with site-specific covariate values, and `det.covs` as a list with each list element corresponding to a covariate to include in the detection portion of the model. Covariates on detection can vary by site and/or survey, and so these covariates may be specified as a site by survey matrix for survey-level covariates or as a one-dimensional vector for survey level covariates. The `ovenHBEF` list is already in the required format. Here we will model OVEN occurrence as a function of linear and quadratic elevation and will include three observational covariates (linear and quadratic day of survey, time of day of survey) on the detection portion of the model. We specify the formulas below

```r
oven.occ.formula <- ~ Elevation + Elevation.2
oven.det.formula <- ~ day + tod + day.2
# Make sure format of ovenHBEF is correct.
str(ovenHBEF)
```

```
List of 4
 $ y       : num [1:373, 1:3] 1 1 0 1 0 0 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:373] "1" "2" "3" "4" ...
  .. ..$ : chr [1:3] "1" "2" "3"
 $ occ.covs: num [1:373, 1:2] -0.889 -0.765 -0.413 -0.14 -0.13 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "Elevation" "Elevation.2"
 $ det.covs:List of 3
  ..$ day  : num [1:373, 1:3] -1.62 -1.62 -1.62 -1.62 -1.62 ...
  ..$ tod  : num [1:373, 1:3] -1.565 -1.378 -1.084 -0.79 -0.549 ...
  ..$ day.2: num [1:373, 1:3] 2.61 2.61 2.61 2.61 2.61 ...
 $ coords  : num [1:373, 1:2] 280000 280000 280000 280001 280000 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:373] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "Easting" "Northing"
```

Next, we specify the starting values for the MCMC sampler in `starting`. `PGOcc` (and all other `spOccupancy` model fitting functions) will set starting values by default (and report what these values are), but here we will do this explicitly. Starting values are specified in a list with the following tags: `z` (latent occurrence values), `alpha` (detection regression coefficients), and `beta` (occurrence regression coefficients). Below we set all initial values of the regression coefficients to 0, and set starting values for `z` based on the detection-nondetection data matrix.

```r
# Number of detection and occupancy parameters
# + 1 is needed to count the intercept
```

```
p.det <- length(ovenHBEF$det.covs) + 1
p.occ <- ncol(ovenHBEF$occ.covs) + 1
oven.starting <- list(alpha = rep(0, p.det),
                      beta = rep(0, p.occ),
                      z = apply(ovenHBEF$y, 1, max, na.rm = TRUE))
```

We next specify the priors for the occurrence and detection regression coefficients. The Polya-Gamma data augmentation algorithm employed by `spOccupancy` assumes normal priors for both the detection and occurrence regression coefficients. These priors are specified in a list with tags `beta.normal` for occurrence and `alpha.normal` for detection parameters. Each list element is then itself a list, with the first element of the list consisting of the hypermeans for each coefficient to be estimated and the second element of the list consisting of the hypervariances for each coefficient. By default, `spOccupancy` will set the hypermeans to 0 and the hypervariances to 2.72, which corresponds to a relatively flat prior on the probability scale (0, 1) (Broms, Hooten, and Fitzpatrick 2016). We will use these default priors here, but we specify them explicitly below for clarity

```
oven.priors <- list(alpha.normal = list(mean = rep(0, p.det),
                                        var = rep(2.72, p.det)),
                    beta.normal = list(mean = rep(0, p.occ),
                                       var = rep(2.72, p.occ)))
```

Our last step is to specify the number of samples to run the MCMC (`n.samples`), the amount of burn-in (`n.burn`), and how often we want to thin the posterior samples (`n.thin`). For a simple single species occupancy model, we shouldn't need to many samples and will only need a small amount of burn-in and very minimal thinning (if any).

```
n.samples <- 4000
n.burn <- 1000
n.thin <- 1
```

We are now set to run the occupancy model. Single species occupancy models are fast, and so we set `n.omp.threads = 1` to indicate we won't use multiple threads to run the model. For more time consuming models, we can set `n.omp.threads` to a number greater than 1 and smaller than the number of threads on the computer you are using. Note this argument will only use multiple threads if `spOccupancy` was compiled for OpenMP support. We set `verbose = TRUE` and `n.report = 1000` to report progress after every 1000th MCMC iteration.

```
out <- PGOcc(occ.formula = oven.occ.formula,
             det.formula = oven.det.formula,
             data = ovenHBEF,
             starting = oven.starting,
             n.samples = n.samples,
             priors = oven.priors,
             n.omp.threads = 1,
             verbose = TRUE,
             n.report = 1000,
             n.burn = n.burn,
             n.thin = n.thin)
```

```
----------------------------------------
	Preparing the data
----------------------------------------
----------------------------------------
	Model description
----------------------------------------
Occupancy model with Polya-Gamma latent
```

```
variable fit with 373 sites.

Number of MCMC samples: 4000
Burn-in: 1000
Thinning Rate: 1
Total Posterior Samples: 3000


Source compiled with OpenMP support and model fit using 1 thread(s).

Sampling ...
Sampled: 1000 of 4000, 25.00%
-------------------------------------------------
Sampled: 2000 of 4000, 50.00%
-------------------------------------------------
Sampled: 3000 of 4000, 75.00%
-------------------------------------------------
Sampled: 4000 of 4000, 100.00%
```

```
str(out)
```

```
List of 16
 $ beta.samples : 'mcmc' num [1:3000, 1:3] 2.65 2.61 2.36 2.44 2.71 ...
  ..- attr(*, "mcpar")= num [1:3] 1 3000 1
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:3] "(Intercept)" "Elevation" "Elevation.2"
 $ alpha.samples: 'mcmc' num [1:3000, 1:4] 0.673 0.813 0.739 0.688 0.646 ...
  ..- attr(*, "mcpar")= num [1:3] 1 3000 1
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:4] "(Intercept)" "day" "tod" "day.2"
 $ z.samples    : 'mcmc' num [1:3000, 1:373] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "mcpar")= num [1:3] 1 3000 1
 $ psi.samples  : 'mcmc' num [1:3000, 1:373] 0.949 0.947 0.933 0.938 0.956 ...
  ..- attr(*, "mcpar")= num [1:3] 1 3000 1
 $ y.rep.samples: int [1:3000, 1:373, 1:3] 0 1 0 1 1 0 0 0 1 1 ...
 $ run.time     : 'proc_time' Named num [1:5] 2.951 0.278 2.731 0 0
  ..- attr(*, "names")= chr [1:5] "user.self" "sys.self" "elapsed" "user.child" ...
 $ X            : num [1:373, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:373] "1" "2" "3" "4" ...
  .. ..$ : chr [1:3] "(Intercept)" "Elevation" "Elevation.2"
  ..- attr(*, "assign")= int [1:3] 0 1 2
 $ X.p          : num [1:1108, 1:4] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1108] "1" "2" "3" "4" ...
  .. ..$ : chr [1:4] "(Intercept)" "day" "tod" "day.2"
  ..- attr(*, "assign")= int [1:4] 0 1 2 3
 $ y            : num [1:373, 1:3] 1 1 0 1 0 0 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:373] "1" "2" "3" "4" ...
  .. ..$ : chr [1:3] "1" "2" "3"
 $ n.samples    : int 4000
 $ call         : language PGOcc(occ.formula = oven.occ.formula, det.formula = oven.det.formula, data =
 $ n.post       : int 3000
 $ n.thin       : int 1
 $ n.burn       : int 1000
 $ pRE          : logi FALSE
 $ psiRE        : logi FALSE
 - attr(*, "class")= chr "PGOcc"
```

You may see a slightly different message depending on whether or not your computer compiled `spOccupancy` with OpenMP support. We see `PGOcc` returns a list of class `PGOcc` with a suite of different objects, most of them being `coda::mcmc` objects of posterior samples. Notice the "Preparing the data" printed section doesn't have any information shown in it. `spOccupancy` functions will present warnings when preparing the data for the model in this section, or will print out the default priors or starting values used when they are not specified in the function call. Here we specified everything explicitly so no information was reported.

For a nice summary of the regression parameters we can use the `summary` function on the resulting `PGOcc` object.

```r
summary(out)
```

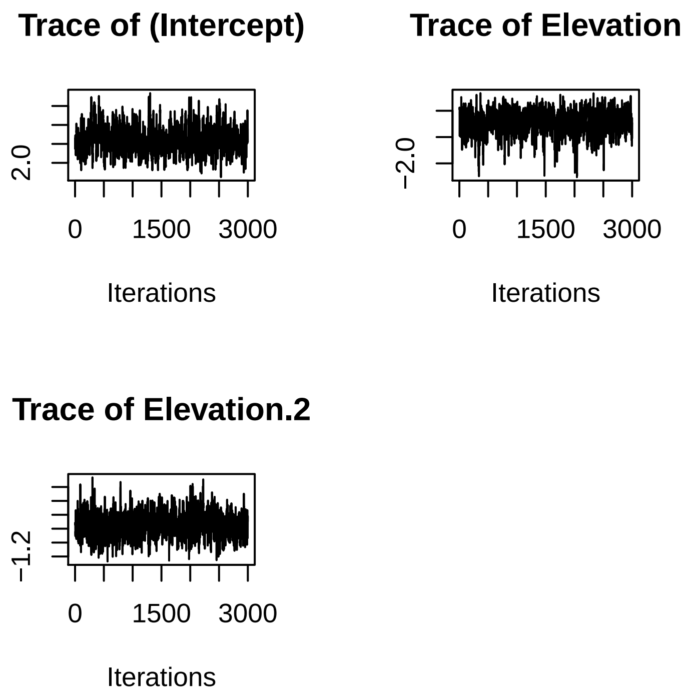
```

Call:
PGOcc(occ.formula = oven.occ.formula, det.formula = oven.det.formula,
    data = ovenHBEF, starting = oven.starting, n.samples = n.samples,
    priors = oven.priors, n.omp.threads = 1, verbose = TRUE,
    n.report = 1000, n.burn = n.burn, n.thin = n.thin)

Chain Information:
Total samples: 4000
Burn-in: 1000
Thin: 1
Total Posterior Samples: 3000

Occurrence:
                2.5%     25%     50%     75%   97.5%
(Intercept)  1.9936  2.3750  2.5926  2.8236  3.3553
Elevation   -1.7598 -1.3560 -1.2142 -1.0796 -0.8600
Elevation.2 -1.0679 -0.8606 -0.7481 -0.6280 -0.3945

Detection:
                2.5%     25%     50%     75%   97.5%
(Intercept)  0.3784  0.5375  0.6227  0.7105  0.8717
day         -0.1583 -0.0757 -0.0292  0.0168  0.0999
tod         -0.2662 -0.1771 -0.1310 -0.0830  0.0018
day.2       -0.3157 -0.2122 -0.1584 -0.1071 -0.0029
```

We see OVEN is fairly prominent in the forest given the large intercept value, and the negative linear and quadratic terms for `Elevation` suggest occurrence probability peaks at mid-elevations.

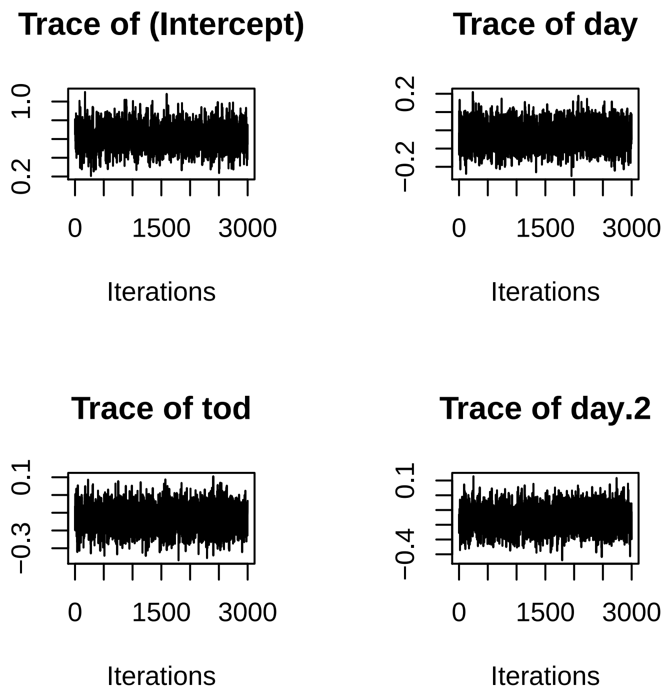
0.3.3 Convergence diagnostics

The posterior samples in the `PGOcc` object are `coda::mcmc` objects, which we can quickly assess for convergence visually using trace plots.

```r
plot(out$beta.samples, density = FALSE)
```



```r
plot(out$alpha.samples, density = FALSE)
```



For a complete analysis (i.e., in a peer-reviewed manuscript), we will likely want to more formally check for convergence, perhaps using the Gelman-Rubin R-hat diagnostic. This requires running multiple chains at largely different starting values for the regression parameters. For a single species non-spatial occupancy model, we can accomplish this by running multiple chains sequentially (since they run really fast) with different starting values, then combining the output into a `coda::mcmc.list` object for use the `coda::gelman.diag` function. Notice below we set `verbose = FALSE` to suppress the messages printed by `PGOcc`.

```r
oven.starting <- list(alpha = rep(2, p.det),
                      beta = rep(2, p.occ),
                      z = apply(ovenHBEF$y, 1, max, na.rm = TRUE))
out.2 <- PGOcc(occ.formula = oven.occ.formula,
               det.formula = oven.det.formula,
               data = ovenHBEF,
               starting = oven.starting,
               n.samples = n.samples,
               priors = oven.priors,
               n.omp.threads = 1,
               verbose = FALSE,
               n.report = 1000,
           n.burn = n.burn,
           n.thin = n.thin)
oven.starting <- list(alpha = rep(-2, p.det),
                      beta = rep(-2, p.occ),
                      z = apply(ovenHBEF$y, 1, max, na.rm = TRUE))
out.3 <- PGOcc(occ.formula = oven.occ.formula,
               det.formula = oven.det.formula,
               data = ovenHBEF,
               starting = oven.starting,
               n.samples = n.samples,
               priors = oven.priors,
               n.omp.threads = 1,
               verbose = FALSE,
               n.report = 1000,
               n.burn = n.burn,
               n.thin = n.thin)
# beta convergence
gelman.diag(mcmc.list(out$beta.samples, out.2$beta.samples,
              out.3$beta.samples))
```

```
Potential scale reduction factors:

            Point est. Upper C.I.
(Intercept)       1.00       1.01
Elevation         1.01       1.02
Elevation.2       1.01       1.02

Multivariate psrf

1.01
```

```r
# alpha convergence
gelman.diag(mcmc.list(out$alpha.samples, out.2$alpha.samples,
              out.3$alpha.samples))
```

```
Potential scale reduction factors:

            Point est. Upper C.I.
(Intercept)          1       1.01
```

```
day                   1       1.00
tod                   1       1.00
day.2                 1       1.00

Multivariate psrf

1
```

0.3.4 Posterior predictive checks

The function `ppcOcc` performs a posterior predictive check on all `spOccupancy` model objects as a Goodness of Fit (GoF) assessment. The fundamental idea of a posterior predictive check is as follows: our model should generate data that closely align with the observed data. If there are drastic differences in the true data from the model generated data, our model likely is not very useful (Hobbs and Hooten 2015). GoF assessments are more complicated using binary data, like detection-nondetection used in occupancy models, as standard approaches are not valid assessments for binary data (Broms, Hooten, and Fitzpatrick 2016, @mccullagh2019). Thus, any approach to assess model fit for detection-nondetection data must bin the raw values in some manner, and then perform a model fit assessment on the binned values. There are numerous ways we could envision binning the raw detection-nondetection values (Kéry and Royle 2015).

The resulting `PGOcc` model object is sent as input to the `ppcOcc` function, along with a fit statistic (`fit.stat`) and numeric value indicating how to group the data (`group`). Currently supported fit statistics include the Freeman-Tukey statistic and the Chi-Square statistic (`freeman-tukey` or `chi-square`, respectively, Kéry and Royle (2015)). Currently, `ppcOcc` allows the user to group the data by row (site; `group = 1`) or column (replicate; `group = 2`). `ppcOcc` will then return a set of posterior samples for the fit statistic (or discrepancy measure) using the observed data (`fit.y`) and model generated data set (`fit.y.rep`), summed across all data points. These values can be used with the `summary` function to generate a Bayesian p-value. Bayesian p-values are sensitive to individual values, so we should also explore the discrepancy measures for each "grouped" data point. `ppcOcc` returns a matrix of posterior quantiles for the fit statistic for both the observed (`fit.y.group.quants`) and model generated data (`fit.y.rep.group.quants`) for each "grouped" data point.

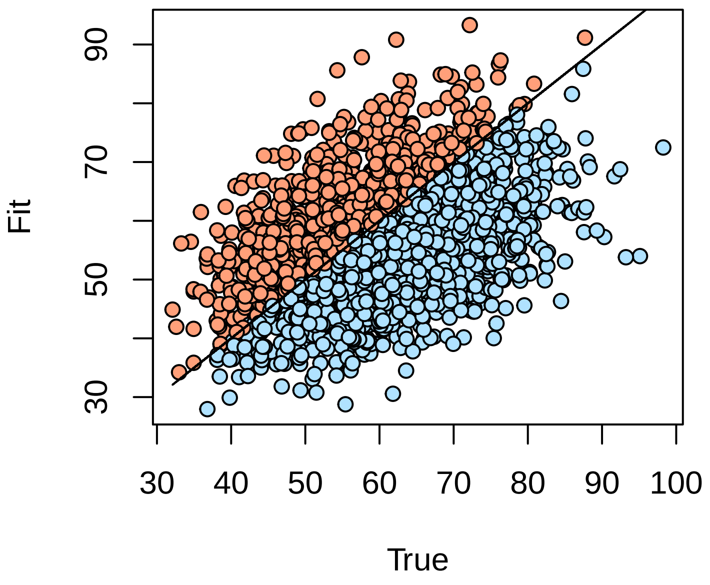
We next perform a posterior predictive check using the Freeman-Tukey statistic grouping the data by sites. We summarize the posterior predictive check with the `summary` function, which reports a Bayesian p-value. A Bayesian p-value that hovers around 0.5 indicates adequate model fit, while values less than 0.1 or greater than 0.9 suggest our model does not fit the data well (Hobbs and Hooten 2015).

```r
ppc.out <- ppcOcc(out, fit.stat = 'freeman-tukey', group = 1)
summary(ppc.out)
```

```

Call:
ppcOcc(object = out, fit.stat = "freeman-tukey", group = 1)

Chain Information:
Total samples: 4000
Burn-in: 1000
Thin: 1
Total Posterior Samples: 3000

Bayesian p-value:  0.3523333
Fit statistic:  freeman-tukey
```

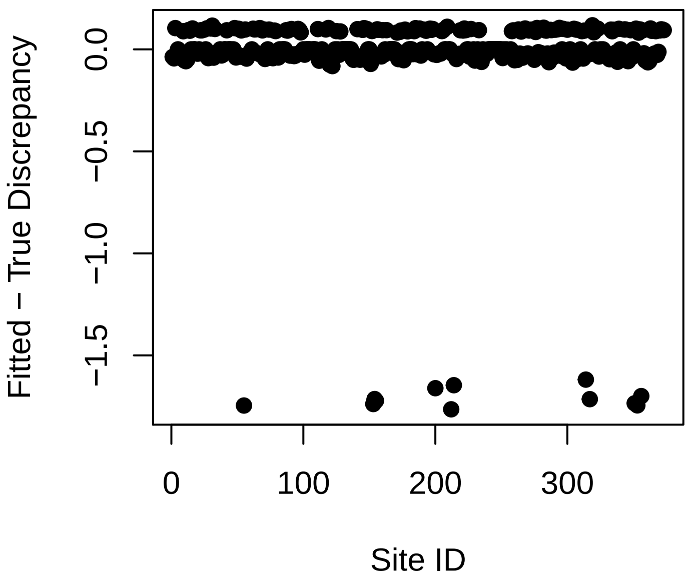
The Bayesian p-value is the proportion of posterior samples of the fit statistic of the model generated data that are greater than the corresponding fit statistic of the true data, summed across all "grouped" data points. We can create a visual representation of the Bayesian p-value as follows, which is highly motivated by Kéry and Royle (2015).

```
ppc.df <- data.frame(fit = ppc.out$fit.y,
                     fit.rep = ppc.out$fit.y.rep,
                     color = 'lightskyblue1')
ppc.df$color[ppc.df$fit.rep > ppc.df$fit] <- 'lightsalmon'
plot(ppc.df$fit, ppc.df$fit.rep, bg = ppc.df$color, pch = 21,
     ylab = 'Fit', xlab = 'True')
lines(ppc.df$fit, ppc.df$fit, col = 'black')
```



Our Bayesian p-value indicates no lack of fit for the OVEN model. However, relying solely on the Bayesian p-value as an assessment of model fit is not always a great option, as individual data points can have an overbearing influence on the resulting summary value. Instead of summing across all data points for a single discrepancy measure, `ppcOcc` also allows us to explore discrepancy measures on a "grouped" point by point basis. The resulting `ppcOcc` object will contain the objects `fit.y.group.quants` and `fit.y.rep.group.quants`, which contain quantiles of the posterior distributions for the discrepancy measures of each grouped data point. Below we plot the difference in the discrepancy measure between the fitted and true data across each of the sites.

```
diff.fit <- ppc.out$fit.y.rep.group.quants[3, ] - ppc.out$fit.y.group.quants[3, ]
plot(diff.fit, pch = 19, xlab = 'Site ID', ylab = 'Fitted - True Discrepancy')
```



We see there are a few sites where the true discrepancy is much larger than the discrepancy under the fitted data. Here we will ignore this, but in a real analysis we would explore these sites further to see what could explain this pattern (e.g., are the sites close together in space?).

0.3.5 Model selection using WAIC

Posterior predictive checks allow us to assess how well our model fits the data, but they are not very useful if we want to compare multiple competing models and ultimately select a final model based on some criterion. Bayesian model selection is very much a constantly changing field, especially in the ecological and environmental sciences. See Hooten and Hobbs (2015) for an accessible overview of Bayesian model selection for ecologists.

For Bayesian hierarchical models like occupancy models, the most common Bayesian model selection criterion, DIC, is not applicable (Hooten and Hobbs 2015). Instead, we can use the Widely Applicable Information Criterion (Watanabe 2010) to compare a set of models and select the best performing model according to the WAIC for final analysis. When focused primarily on predictive performance, an out-of-sample validation approach is another attractive (but more computationally intensive) alternative to compare a series of models, especially since WAIC may not be a reliable metric for all models (**???**). We will show how to perform out of sample model validation using `spOccupancy` model objects in a subsequent vignette.

The WAIC is calculated for all `spOccupancy` model objects using the function `waicOcc`. We calculate the WAIC as

$$\text{WAIC} = -2 \times (\text{elpd} - \text{pD}),$$

where elpd is the expected log pointwise predictive density and PD is the effective number of parameters. We calculate elpd by calculating the likelihood for each posterior sample, taking the mean of these likelihoods, taking the log of the mean of the likelihoods, and summing these values across all sites. We calculate the effective number of parameters by calculating the variance of the log likelihood for each site taken over all posterior samples, and then summing these values across all sites.

We calculate the WAIC using `waicOcc` for our OVEN model below.

```r
waicOcc(out)
```

```
      elpd          pD        WAIC
-697.92206    5.88771 1407.61955
```

Next we rerun the OVEN model, but this time we assume occurrence is constant across the HBEF, and subsequently compare the WAIC value to the full model

```r
p.occ <- 1
oven.starting <- list(alpha = rep(0, p.det),
                      beta = rep(0, p.occ),
                      z = apply(ovenHBEF$y, 1, max, na.rm = TRUE))
oven.priors <- list(alpha.normal = list(mean = rep(0, p.det),
                                        var = rep(2.72, p.det)),
                    beta.normal = list(mean = rep(0, p.occ),
                                       var = rep(2.72, p.occ)))
out.small <- PGOcc(occ.formula = ~ 1,
           det.formula = oven.det.formula,
           data = ovenHBEF,
           starting = oven.starting,
           n.samples = n.samples,
           priors = oven.priors,
           n.omp.threads = 1,
           verbose = FALSE,
           n.burn = n.burn,
           n.thin = n.thin)
waicOcc(out.small)
```

```
       elpd           pD         WAIC
-766.028630    3.903127 1539.863513
```

Smaller values of WAIC indicate models with better performance. We see the WAIC for the model with elevation is smaller than the intercept only model, indicating elevation is an important predictor for OVEN occurrence in HBEF.

0.3.6 Prediction

All resulting model objects from `spOccupancy` model functions can be used with the `predict` function to generate a series of posterior predictive samples at non-sampled locations, given the values of all covariates used in the model fitting process. The object `hbefElev` contains elevation values at a 30x30m resolution from the National Elevation Dataset across the entire HBEF. The values are standardized using the mean and standard deviation of the elevation values used to fit the model. We load the data below

```r
data(hbefElev)
str(hbefElev)
```

```
'data.frame':   46091 obs. of  3 variables:
 $ val     : num  2.07 2.08 2.09 2.1 2.12 ...
 $ Easting : num  276269 276291 276314 276336 276358 ...
 $ Northing: num  4871425 4871425 4871425 4871425 4871425 ...
```

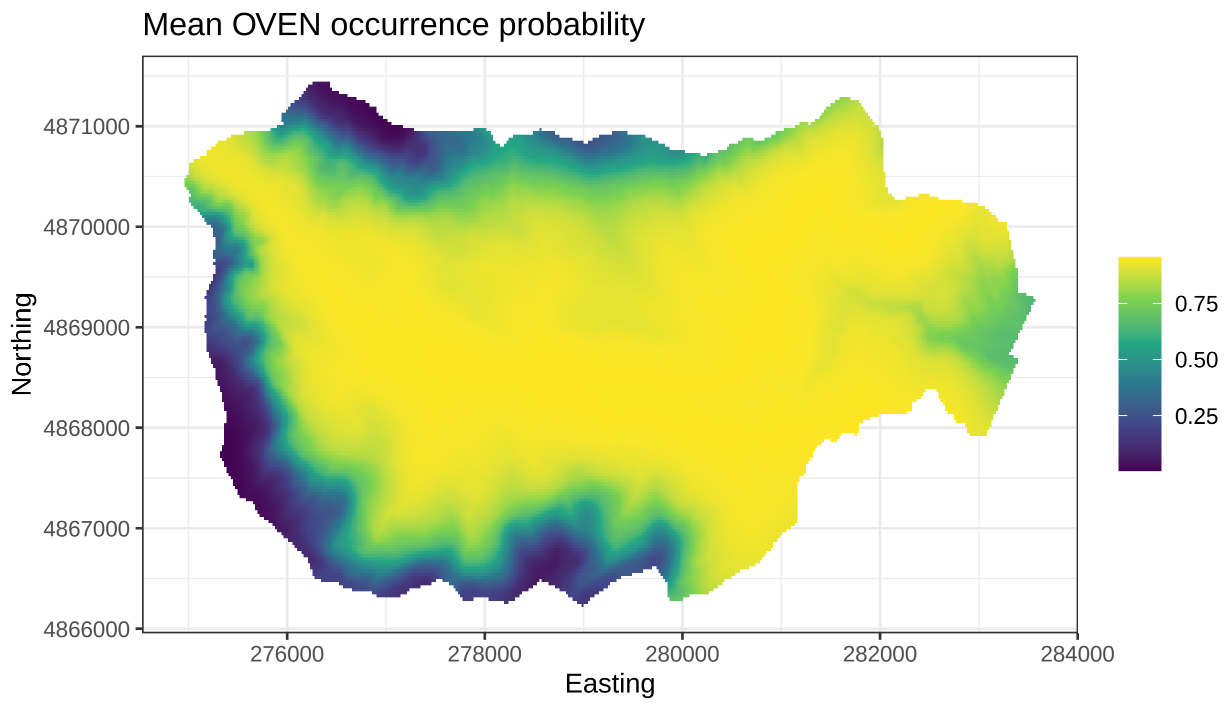
The column `val` contains the standardized elevation values, while `Easting` and `Northing` contain the spatial coordinates that we will use for plotting. We can obtain posterior predictive samples for the occurrence probabilities at these sites by using the `predict` function and our `PGOcc` model object.

```
X.0 <- cbind(1, hbefElev$val, hbefElev$val^2)
out.pred <- predict(out, X.0)
```

For `PGOcc` objects, the `predict` function takes two arguments: (1) the `PGOcc` model object; and (2) a matrix or data frame consisting of the design matrix for the prediction locations (including an intercept). The resulting object consists of posterior predictive samples for the latent occurrence probabilities (`psi.0.samples`) and latent occurrence values (`z.0.samples`). The beauty of the Bayesian paradigm is that these predictions all have fully propagated uncertainty. We can use these values to create plots of the predicted mean occurrence values, as well as their standard deviation.

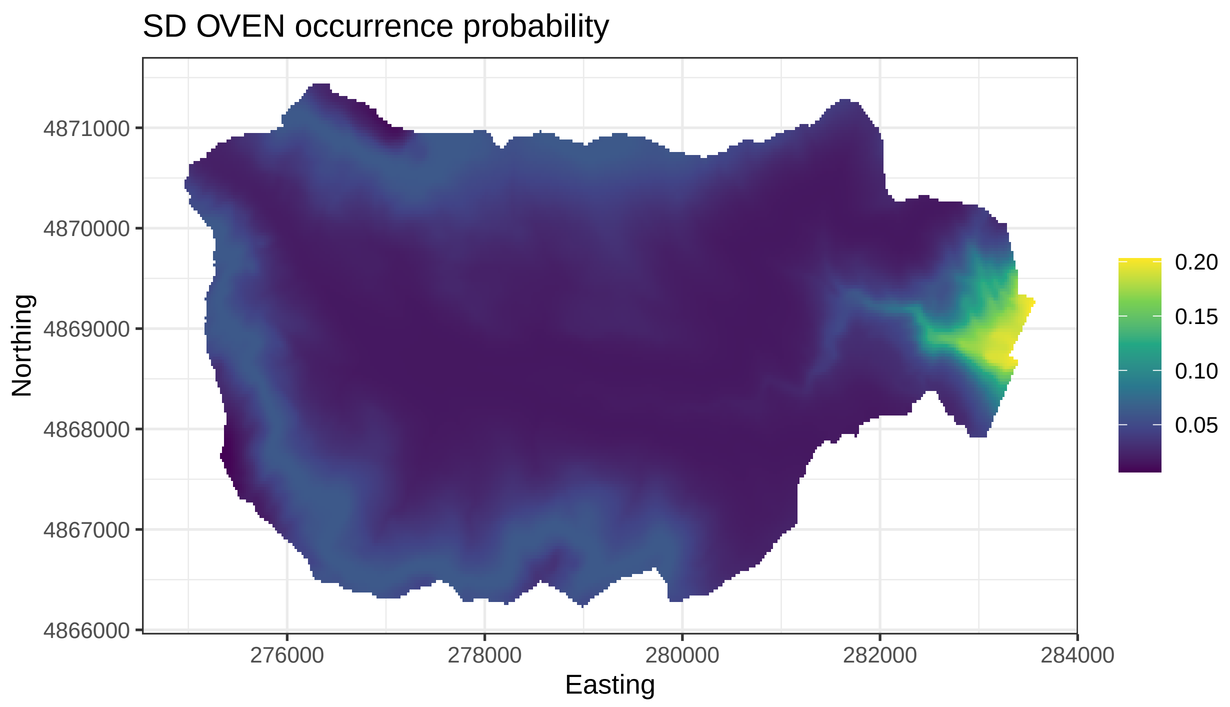
```
plot.dat <- data.frame(x = hbefElev$Easting,
                       y = hbefElev$Northing,
                       mean.psi = apply(out.pred$psi.0.samples, 2, mean),
                       sd.psi = apply(out.pred$psi.0.samples, 2, sd))

dat.stars <- st_as_stars(plot.dat, dims = c('x', 'y'))
ggplot() +
  geom_stars(data = dat.stars, aes(x = x, y = y, fill = mean.psi)) +
  scale_fill_viridis_c(na.value = NA) +
  labs(x = 'Easting', y = 'Northing', fill = '',
       title = 'Mean OVEN occurrence probability') +
  theme_bw()
```

```
Warning: Removed 19143 rows containing missing values (geom_raster).
```



```
ggplot() +
  geom_stars(data = dat.stars, aes(x = x, y = y, fill = sd.psi)) +
  scale_fill_viridis_c(na.value = NA) +
  labs(x = 'Easting', y = 'Northing', fill = '',
       title = 'SD OVEN occurrence probability') +
  theme_bw()
```

```
Warning: Removed 19143 rows containing missing values (geom_raster).
```



0.4 Single species spatial occupancy models

0.5 Basic model description

We extend the basic single species occupancy model to incorporate a spatial Gaussian Process that accounts for unexplained spatial variation in species occurrence across a region of interest. The species-specific occurrence probability at site j, ψ_j, now takes the form

$$\text{logit}(\psi_j) = \boldsymbol{x}_j' \cdot \boldsymbol{\beta} + \text{w}_j, \tag{3}$$

where w_j is a realization from a zero-mean spatial Gaussian Process, i.e.,

$$\mathbf{w} \sim N(\mathbf{0}, \boldsymbol{\Sigma}(\boldsymbol{s}, \boldsymbol{s}', \boldsymbol{\theta})). \tag{4}$$

We define $\boldsymbol{\Sigma}(\boldsymbol{s}, \boldsymbol{s}', \boldsymbol{\theta})$ as a $J \times J$ covaraince matrix that is a function of the distances between any pair of site coordinates $\boldsymbol{s}$ and $\boldsymbol{s}'$ and a set of parameters $(\boldsymbol{\theta})$ that govern the spatial process. The vector $\boldsymbol{\theta}$ is equal to $\boldsymbol{\theta} = \{\sigma^2, \phi, \nu\}$, where σ^2 is a spatial variance parameter, ϕ is a spatial decay parameter, and ν is a spatial smoothness parameter. ν is only specified when using a Matern correlation function.

The detection portion of the occupancy model remains unchanged from the non-spatial occupancy model and follows Equation (2). Formulation of Polya-Gamma latent variables is also exactly analogous to the nonspatial model (Equation (**??**)), with all references to ψ_j now including the latent spatial random effects in addition to the site-level covariates.

0.6 spPGOcc: single species spatial occupancy models

When working across large spatial domains, accounting for residual spatial autocorrelation in species distributions can often improve predictive performance, leading to more accurate species distribution maps (Guélat and Kéry 2018, lany2020). The function spPGOcc fits single species spatial occupancy models using Polya-Gamma latent variables, where spatial autocorrelation is accounted for using a spatial Gaussian Process.

Perhaps one of the reasons why spatially-explicit occupancy models are not widely used is due to the large run times required when running occupancy models across large regions. This problem is known as the "big N" problem in the spatial statistics literature (Heaton et al. 2019). When the number of sites is greater than say 1000, spatial Gaussian process models can be drastically slow as a result of needing to take the inverse of the spatial covariance matrix at each MCMC iteration. Numerous approximation methods exist to reduce this computational cost (Heaton et al. 2019). spPGOcc fits spatial occupancy models using either a full Gaussian process or a Nearest Neighbor Gaussian Process (NNGP) (Datta et al. 2016, @finley2019efficient), which provides nearly identical results to the full Gaussian process at a fraction of the computational cost. See Datta et al. (2016) for statistical details on the NNGP, as well as Finley, Datta, and Banerjee (2020) for details on using NNGPs with Polya-Gamma latent variables.

We will fit the same occupancy model for OVEN that we fit previously using PGOcc, but we will now make the model spatially explicit by incorporating a spatial process with spPGOcc. First, let's take a look at the arguments for spPGOcc:

```r
spPGOcc(occ.formula, det.formula, data, starting, n.batch,
        batch.length, accept.rate = 0.43, priors,
        cov.model = "exponential", tuning, n.omp.threads = 1,
        verbose = TRUE, NNGP = FALSE, n.neighbors = 15,
        search.type = "cb", n.report = 100,
        n.burn = round(.10 * n.batch * batch.length),
        n.thin = 1, ...)
```

We will walk through each of the arguments to spPGOcc in the context of our Ovenbird example. The occupancy (occ.formula) and detection (det.formula) formulas, as well as the list of data (data), take the same form as with PGOcc. Notice the coords matrix in the ovenHBEF list of data. We did not use this for PGOcc but specifying the spatial coordinates in data is required for all spatially explicit models in spOccupancy.

```r
oven.occ.formula <- ~ Elevation + Elevation.2
oven.det.formula <- ~ day + tod + day.2
str(ovenHBEF) # coords is required for spPGOcc.
```

The starting values (starting) are again specified in a list. Valid tags for starting values now additionally include the parameters associated with the spatial random effects. These include: sigma.sq (spatial variance parameter), phi (spatial range parameter), w (the latent spatial random effects at each site), and nu (spatial smoothness parameter). nu is only specified if using a Matern covariance function (i.e., cov.model = 'matern'). spOccupancy supports four spatial covariance models (exponential, spherical, gaussian, and matern), which are specified in the cov.model argument. Here we will use an exponential covariance model. As a starting value for the spatial range parameter phi, we compute the mean distance between points in HBEF and then set it equal to 3 divided by this mean distance. When using an exponential covariance function, $\frac{3}{\phi}$ is the effective range, or the distance at which the residual spatial correlation between two sites is 0.05. Thus our initial guess for this effective range is the average distance betweeen sites across HBEF.

```r
# Distances between sites
dist.hbef <- dist(ovenHBEF$coords)
# Number of detection and occupancy regression parameters
p.det <- length(ovenHBEF$det.covs) + 1
p.occ <- ncol(ovenHBEF$occ.covs) + 1
# Exponential covariance model
```

```r
cov.model <- "exponential"
oven.starting <- list(alpha = rep(0, p.det),
                      beta = rep(0, p.occ),
                      z = apply(ovenHBEF$y, 1, max, na.rm = TRUE),
                      sigma.sq = 2,
                      phi = 3 / mean(dist.hbef),
                      w = rep(0, nrow(ovenHBEF$y)))
```

The next three arguments (`n.batch`, `batch.length`, and `accept.rate`) are all related to the Adaptive MCMC sampler we use to fit the model. Updates for the spatial range parameter (and smoothness parameter if `cov.model = matern`) require the use of a Metropolis Hastings algorithm. We implement an adaptive Metropois-Hastings algorithm discussed in Roberts and Rosenthal (2009). This algorithm adjusts the tuning values for each parameter that requires a Metropolis-Hastings update within the sampler itself. This process results in a more efficient sampler than if we were to fix the tuning parameters prior to fitting the model. The parameter `accept.rate` is the target acceptance rate for each parameter, and the algorithm will adjust the tuning parameters to hover around this value. The default value is 0.43, which we suggest leaving as is unless you have a good reason to change it. The tuning parameters are updated after a single "batch". We must specify the total `n.batch` batches, where each "batch" consists of `batch.length` MCMC samples. Thus, the total number of MCMC samples is `n.batch * batch.length`. Typically, we set `batch.length = 25` and then play around with `n.batch` until convergence is reached. Here we set `n.batch = 400` for a total of 10000 MCMC samples. We will additionally specify a burn-in period of 2000 samples and a thinning rate of 4. We also need to specify an initial value for the tuning parameters for the spatial decay and smoothness parameters (if applicable). These values are sent as input in the form of a list with tags `phi` and `nu`. The initial tuning value can be any value greater than 0, but we recommend starting the value out around 0.5. After some initial runs of the model, if you notice the final acceptance rate of a parameter is much larger than the target acceptance rate (`accept.rate`), you can then change the initial tuning value to get closer to the target rate.

```r
batch.length <- 25
n.batch <- 400
n.burn <- 2000
n.thin <- 4
oven.tuning <- list(phi = 0.5)
```

Priors are again specified in a list in the argument `priors`. We assume an inverse gamma prior for the spatial variance parameter `sigma.sq` (tag is `sigma.sq.ig`), and uniform priors for the spatial decay parameter `phi` and smoothness parameter `nu` (if Matern), with the associated tags `phi.unif` and `nu.unif`. The hyperparameters of the inverse Gamma are passed as a vector of length two, with the first and second elements corresponding to the shape and scale, respectively. The lower and upper bounds of the uniform distribution are passed in as a two-element vector for the uniform priors.

The priors for the spatial parameters in a spatially-explicit model must be at least weakly informative for the model to converge (Banerjee, Carlin, and Gelfand 2003). For the inverse-Gammma prior on the spatial variance, we typically set the shape parameter to 2 and the scale parameter equal to our best guess of the spatial variance. For the spatial decay parameter, we determine the bounds of the uniform distribution by computing the smallest distance between sites and the largest distance between sites. We then set the lower bound of the uniform to `3/max` and the upper bound to `3/min`, where min and max correspond to the predetermined distances between sites.

```r
# Minimum value is 0, so need to grab second smallest
min.dist <- sort(unique(dist.hbef))[2]
max.dist <- max(dist.hbef)
oven.priors <- list(beta.normal = list(mean = rep(0, p.occ),
                                       var = rep(2.72, p.occ)),
                    alpha.normal = list(mean = rep(0, p.det),
                                        var = rep(2.72, p.det)),
                    sigma.sq.ig = c(2, 2),
                    phi.unif = c(3/max.dist, 3/min.dist))
```

The argument `n.omp.threads` specifies the number of threads to use for parallelization, while `verbose` specifies whether or not to print the progress of the sampler. We *highly* recommend setting `verbose = TRUE` for all spatial models to ensure the adaptive MCMC is working as you want. The argument `n.report` specifies the interval to report the Metropolis sampler acceptance. Note that `n.report` is specified in terms of batches, not the overall number of samples. Below we set `n.report = 100`, which will result in information on the acceptance rate and tuning parameters every 100th batch.

```
n.omp.threads <- 1
verbose <- TRUE
n.report <- 100
```

The remaining parameters (`NNGP`, `n.neighbors` and `search.type`) relate to whether or not you want to fit the model with a Gaussian Process (GP) or NNGP. The argument `NNGP` is a logical value indicating whether to fit the model with an NNGP (`TRUE`) or a regular GP (`FALSE`). For data sets that have more than 1000 locations, using an NNGP will have substantial increases in run time. Even for more modest size data sets (like the HBEF data set), using an NNGP will be quite a bit faster. Unless you are concerned about the NNGP approximation for some reason, we recommend setting `NNGP = TRUE`. The argument `n.neighbors` and `search.type` specify the number of neighbors used in the NNGP and the nearest neighbor search algorithm, respectively, to use for the NNGP model. Generally, the default values of these arguments will be adequate. Datta et al. (2016) showed that setting `n.neighbors = 15` is usually sufficient, although for certain data sets a good approximation can be achieved with as small as five neighbors, which could substantially decrease run time. We generally recommend leaving `search.type = "cb"`, as this results in a fast code book nearest neighbor search algorithm. However, details on when you may want to change this are described in Finley, Datta, and Banerjee (2020). We will run an NNGP model using the default values for `n.neighbors` and `search.type`, and so we won't explicitly specify them in the model call.

We now fit the model and summarize the results using `summary`.

```
out.sp <- spPGOcc(occ.formula = oven.occ.formula,
          det.formula = oven.det.formula,
          data = ovenHBEF,
          starting = oven.starting,
          n.batch = n.batch,
          batch.length = batch.length,
          priors = oven.priors,
          cov.model = cov.model,
          NNGP = TRUE,
          tuning = oven.tuning,
          n.report = n.report,
          n.burn = n.burn,
          n.thin = n.thin)
str(out.sp)
summary(out.sp)
```

We see `spPGOcc` returns a list of class `spPGOcc` and consists of posterior samples for all parameters. Note that posterior samples for spatial parameters are stored in the list element `theta.samples`. The `summary` function reveals model results generally align with those found using the non-spatial model.

0.6.1 Convergence diagnostics

Convergence diagnostics, posterior predictive checks, model selection, and out-of-sample prediction all proceed analogously to what we saw with the non-spatial occupancy model using `PGOcc`.

```r
plot(out.sp$beta.samples, density = FALSE)
plot(out.sp$alpha.samples, density = FALSE)
plot(out.sp$theta.samples, density = FALSE)
```

We might want to run the chain for a bit longer to ensure convergence of the spatial parameters, but we'll resist doing so for now. Convergence can be more formally assessed using the Gelman-Rubin diagnostic as done for the nonspatial model.

0.6.2 Posterior predictive checks

For our posterior predictive check, we send the `spPGOcc` model object to the `ppcOcc` function.

```r
ppc.sp.out <- ppcOcc(out.sp, fit.stat = 'freeman-tukey', group = 1)
summary(ppc.sp.out)
```

The Bayesian p-value does not suggest any lack of fit.

0.6.3 Model selection using WAIC

We next use the `waicOcc` function to compute the WAIC, which we can compare to the non-spatial model to assess the benefit of incorporating the spatial random effects.

```r
waicOcc(out.sp)
# Compare to non-spatial model
waicOcc(out)
```

We see the WAIC value for the spatial model is marginally smaller than that of the nonspatial model, indicating that incorporation of the spatial random effects may not be necessary. This is not all that surprising, as we expect elevation to soak up most of the spatial variation in OVEN occurrence across the forest.

0.6.4 Prediction

Finally, we can perform out of sample prediction using the `predict` function just as before. Out of sample prediction for spatial models is more computationally intensive than non-spatial models, and so the `predict` function for `spPGOcc` class objects also has options for parallelization (`n.omp.threads`) and printing sampler progress (`verbose` and `n.report`). Note that for `spPGOcc`, you also need to supply the coordinates of the out of sample prediction locations in addition to the covariate values.

```r
coords.0 <- as.matrix(hbefElev[, c('Easting', 'Northing')])
out.sp.pred <- predict(out.sp, X.0, coords.0, verbose = TRUE, verbose = FALSE)
plot.dat <- data.frame(x = hbefElev$Easting,
                y = hbefElev$Northing,
                mean.psi = apply(out.sp.pred$psi.0.samples, 2, mean),
                sd.psi = apply(out.sp.pred$psi.0.samples, 2, sd))
test <- rasterFromXYZ(plot.dat)
plot(test[[1]], main = 'Mean OVEN occurrence probability', xlab = 'Easting',
     ylab = 'Northing')
plot(test[[2]], main = 'SD OVEN occurrence probability', xlab = 'Easting',
     ylab = 'Northing')
```

The mean predictions look nearly identical to the non-spatial model. The standard deviations for the spatial predictions are larger.

0.7 `msPGOcc`: Multispecies occupancy models

`spOccupancy` uses nearly identical syntax for fitting multispecies models as it does for single species models and provides the same functionality for posterior predictive checks, GoF assessments using WAIC, and out of sample prediction. The `msPGOcc` function fits nonspatial multispecies occupancy models using Polya-Gamma latent variables, which results in substantial increases in run time compared to standard implementations of logit link multispecies occupancy models. `msPGOcc` has exactly the same arguments as `PGOcc`:

```r
msPGOcc(occ.formula, det.formula, data, starting, n.samples, priors,
    n.omp.threads = 1, verbose = TRUE, n.report = 100,
    n.burn = round(.10 * n.samples), n.thin = 1, ...)
```

We will again use the Hubbard Brook data in `hbef2015` as an example data set, but we will now model occurrence for all 12 species in the community. Below we reload the `hbef2015` data set to get a fresh copy.

```r
data(hbef2015)
```

We will model occurrence for all species as a function of linear and quadratic elevation, and detection as a function of linear and quadratic day of survey as well as the time of day the survey occurred. These models are specified in the `occ.formula` and `det.formula` as before, which reference variables stored in the `data` list. For multispecies models, the multispecies detection-nondetection data `y` is now a three-dimensional array with dimensions corresponding to species, sites, and replicates. This is how the data are provided in the `hbef2015` object, so we don't need to do any additional prep.

```r
occ.ms.formula <- ~ Elevation + Elevation.2
det.ms.formula <- ~ day + tod + day.2
str(hbef2015)
```

Next we specify the starting values in `starting`. For multispecies occupancy models, we need to supply starting values for community-level and species-level parameters. In `msPGOcc`, we will supply starting values for the following parameters: `alpha.comm` (community level detection coefficients), `beta.comm` (community level occurrence coefficients), `alpha` (species level detection coefficients), `beta` (species level occurrence coefficients), `tau.beta` (community level occurrence variance parameters), `tau.alpha` (community level detection variance parameters, `z` (latent occurrence values for all species). These are all specified in a single list. Starting values for community level parameters are vectors of length corresponding to the number of community-level detection or occurrence parameters in the model (including the intercepts), while starting values for species level parameters are matrices with the number of rows indicating the number of species, and each column corresponding to a different regression parameter. The starting values for the latent occurrence matrix are specified as a matrix with N rows corresponding to the number of species and J columns corresponding to the number of sites.

```r
N <- dim(hbef2015$y)[1]
p.det <- length(hbef2015$det.covs) + 1
p.occ <- ncol(hbef2015$occ.covs) + 1
ms.starting <- list(alpha.comm = rep(0, p.det),
                    beta.comm = rep(0, p.occ),
                    beta = matrix(0, N, p.occ),
                    alpha = matrix(0, N, p.det),
                    tau.beta = rep(1, p.occ),
                    tau.alpha = rep(1, p.det),
                    z = apply(hbef2015$y, c(1, 2), max, na.rm = TRUE))
```

In multispecies models, we specify priors on the community-level coefficients rather than the species-level effects. For nonspatial models, these priors are specified with the following tags: `beta.comm.normal` (normal prior on the community level occurrence mean effects), `alpha.comm.normal` (normal prior on the community level detection mean effects), `tau.beta.ig` (inverse-Gamma prior on the community level occurrence variance parameters), `tau.alpha.ig` (inverse-Gamma prior on the community level detection variance parameters).

Each tag consists of a list with elements corresponding to the mean and variance for normal priors and scale and shape for inverse-Gamma priors.

Below we specify normal priors to be relatively non-informative on the probability scale with a mean of 0 and variance of 2.72, and specify vague inverse gamma priors on the community level variance parameters setting both the shape and scale parameters to 0.1.

```r
ms.priors <- list(beta.comm.normal = list(mean = rep(0, p.occ),
                                          var = rep(2.72, p.occ)),
                  alpha.comm.normal = list(mean = rep(0, p.det),
                                           var = rep(2.72, p.det)),
                  tau.beta.ig = list(a = rep(0.1, p.occ),
                                     b = rep(0.1, p.occ)),
                  tau.alpha.ig = list(a = rep(0.1, p.det),
                                      b = rep(0.1, p.det)))
```

All that's left to do is specify the number of threads to use (`n.omp.threads`), the number of MCMC samples (`n.samples`), the amount of samples to discard as burn-in (`n.burn`), the the thinning rate (`n.thin`), and arguments to control the display of sampler progress (`verbose`, `n.report`).

```r
out.ms <- msPGOcc(occ.formula = occ.ms.formula,
          det.formula = det.ms.formula,
          data = hbef2015,
          starting = ms.starting,
          n.samples = 20000,
          priors = ms.priors,
          n.omp.threads = 1,
          verbose = TRUE,
          n.report = 5000,
          n.burn = 10000,
          n.thin = 4)
out.ms$run.time
```

We see `msPGOcc` took less than 3 minutes to run the multispecies occupancy model with 373 sites and 12 species for a total of 20,000 iterations. The resulting object `out.ms` is a list of class `msPGOcc` consisting primarily of posterior samples of all community and species level parameters, as well as some additional objects that are used for summaries, prediction, and model fit evaluation. We can display a nice summary of these results using the `summary` function. For multispecies objects, when using summary we need to specify the level of parameters we want to summarize. We do this using the argument `level`, which takes values `community`, `species`, or `both` to print results for community-level parameters, species-level parameters, or all parameters.

```r
summary(out.ms, level = 'both')
```

Looking at the community level variance parameters, we see large variability in the average occurrence (the intercept) for the twelve species, as well as substantial variability in the effect of elevation across the community. There appears to be less variability across species in the detection portion of the model. We can look directly at the species-specific effects to confirm this.

0.7.1 Convergence diagnostics

The resulting posterior samples in the `msPGOcc` object are `coda:mcmc` samples, and so convergence diagnostics can proceed as before.

```r
plot(out.ms$beta.comm.samples, density = FALSE)
# Look at the first few species-specific occurrence intercepts
```

```r
plot(out.ms$beta.samples[, 1:4], density = FALSE)
```

Looking at the species-specific intercepts, we would probably want to run the model a bit longer. Formal assessments of convergence using the Gelman-Rubin diagnostic can be accomplished following the steps shown for `PGOcc` using the `gelman.diag` function.

0.7.2 Posterior predictive checks

We can use the `ppcOcc` function to perform a posterior predictive check, and summarize the check with a Bayesian p-value using the `summary` function. The `summary` function again requires the `level` argument to specify if you want an overall Bayesian p-value for the entire community (`level = 'community'`), each individual species (`level = 'species'`), or both (`level = 'both'`).

```r
ppc.ms.out <- ppcOcc(out.ms, 'chi-square', group = 1)
summary(ppc.ms.out, level = 'both')
```

The Bayesian p-value for the overall community suggests an adequate model fit, but looking closer at each individual species reveals certain species (the very common species) that our model may not be doing a great job. We should explore this further in a complete analysis (and also of course run the model longer to ensure convergence, as this may be contributing to many of the extreme values).

0.7.3 Model selection using WAIC

We can compute the WAIC for comparison with alternative models using the `waicOCC` function.

```r
waicOcc(out.ms)
```

0.7.4 Prediction

Out-of-sample prediction with `msPGOcc` objects is exactly analogous to what we saw with `PGOcc`. We can use the `predict` function along with a data frame of covariates at new locations. We predict across the entire HBEF for all twelve species using the elevation data stored in `hbefElev`. Instead of predicting for all 30 x 30 m cells across the HBEF, below we predict values at every 50th cell. We predict only for OVEN for comparison to predictions using single species occupancy models.

```r
X.0.ms <- cbind(1, hbefElev$val, hbefElev$val^2)
X.0.ms <- X.0.ms[seq(1, nrow(X.0.ms), by = 50), ]
out.ms.pred <- predict(out.ms, X.0.ms)
psi.pred.oven <- out.ms.pred$psi.0.samples[, which(sp.names == 'OVEN'), ]
hist(apply(psi.pred.oven, 2, mean), xlab = 'Mean Occurrence Probability')
```

The histogram supports our results from the single species model that indicated OVEN was common throughout most of the HBEF.

0.8 `spMsPGOcc`: Multispecies spatial occupancy models

Residual spatial autocorrelation may perhaps be more prominent in multispecies occupancy models compared to single species models, as a single set of covariates is used to explain occurrence probability across a region of interest for all species. Given the large variety individual species show in habitat requirements, this may result in important drivers of occurrence probability not being included for certain species, resulting in many species having high residual spatial autocorrelation. The function `spMsPGOcc` fits spatially explicit multispecies occupancy models. Similar to single species models using `spPGOcc`, models can be fit using either a full Gaussian Process (GP) or a Nearest Neighbor Gaussian Process (NNGP). `msSpPGOcc` fits a separate spatial process for each species. The syntax for `msSpPGOcc` is analogous to the syntax for single species spatially-explicit models using `spPGOcc`.

```
spMsPGOcc(occ.formula, det.formula, data, starting, n.batch,
          batch.length, accept.rate = 0.43, priors,
          cov.model = "exponential", tuning, n.omp.threads = 1,
          verbose = TRUE, NNGP = TRUE, n.neighbors = 15,
          search.type = "cb", n.report = 100,
          n.burn = round(.10 * n.batch * batch.length), n.thin = 1, ...)
```

We will again display the model using the HBEF foliage-gleaning bird data set, with the same predictors in our occurrence and detection models

```
occ.ms.sp.formula <- ~ Elevation + Elevation.2
det.ms.sp.formula <- ~ day + tod + day.2
```

Our starting values in the `starting` argument will look analagous to what we specified for the nonspatial multispecies occupancy model using `msPGOcc`, but we will also include additional starting values for the parameters controlling the spatial processes: `sigma.sq` is the species-specific spatial variance parameter, `phi` is the species specific spatial decay parameter, and `w` is the latent spatial proccess for each species at each site. We will use an exponential covariance model, but when using a Matern covariance model we must also specify starting values for `nu`, the species-specific spatial smoothness parameter. Note that all species-specific spatial parameters are independent of each other. We do not share any information from one spatial process to another. This is something we plan to incorporate for future `spOccupancy` development. Starting values for `phi`, `sigma.sq`, and `nu` (if applicable) are specified as vectors with N elements (the number of species being modeled), while the starting values for the latent spatial processes are specified as a matrix with N rows (i.e., species) and J columns (i.e., sites). Here we set the starting value for the spatial variances equal to 2 for all species and set the starting values for the spatial decay parameter to yield an effective range of the average distance between sites across the HBEF.

```
# Number of species
N <- dim(hbef2015$y)[1]
# Distances between sites
dist.hbef <- dist(hbef2015$coords)
# Number of detection and occupancy regression parameters
p.det <- length(hbef2015$det.covs) + 1
p.occ <- ncol(hbef2015$occ.covs) + 1
# Exponential covariance model
cov.model <- "exponential"
ms.starting <- list(alpha.comm = rep(0, p.det),
                    beta.comm = rep(0, p.occ),
                    beta = matrix(0, N, p.occ),
                    alpha = matrix(0, N, p.det),
                    tau.beta = rep(1, p.occ),
                    tau.alpha = rep(1, p.det),
                    z = apply(hbef2015$y, c(1, 2), max, na.rm = TRUE),
            sigma.sq = rep(2, N),
            phi = rep(3 / mean(dist.hbef), N),
            w = matrix(0, N, dim(hbef2015$y)[2]))
```

We next specify the priors in the `priors` argument. The priors are the same as those we specified for the non-spatial multispecies model, with the addition of priors for the parameters controlling the species-specific spatial processes. We assume independent priors for all spatial parameters across the different species. For each species, we assign an inverse gamma prior for the spatial varaince parameter `sigma.sq` (tag is `sigma.sq.ig`) and uniform priors for the spatial decay parameter `phi` and smoothness parameter `nu` (if `cov.model = 'matern'`), with the associated tags `phi.unif` and `nu.unif`. All priors are specified as lists with two elements. For the inverse-Gamma prior, the first element is a length N vector of shape parameters for each species, and the second element is a length N vector of scale parameters for each species. For the uniform priors, the first element is a length N vector of the lower bounds for each species, and the second element is a length N vector of upper bounds for each species. For the inverse-Gamma prior on the spatial variances, here we set the shape parameter to 2 and the scale parameter equal to 2. For a more formal analysis, we would likely want to do some exploratory data analysis to obtain a better guess for the spatial variance for each species, and then replace the scale parameter with this estimated guess for each species. For the spatial decay parameter, we determine the bounds of the uniform distribution by computing the smallest distance between sites and the largest distance between sites. We then set the lower bound of the uniform to `3/max` and the upper bound to `3/min`, where `min` and `max` correspond to the predetermined distances between sites.

```
# Minimum value is 0, so need to grab second element.
min.dist <- sort(unique(dist.hbef))[2]
max.dist <- max(dist.hbef)
ms.priors <- list(beta.comm.normal = list(mean = rep(0, p.occ),
                                          var = rep(2.72, p.occ)),
                  alpha.comm.normal = list(mean = rep(0, p.det),
                                           var = rep(2.72, p.det)),
                  tau.beta.ig = list(a = rep(0.1, p.occ),
                                     b = rep(0.1, p.occ)),
                  tau.alpha.ig = list(a = rep(0.1, p.det),
                                      b = rep(0.1, p.det)),
          sigma.sq.ig = list(a = rep(2, N),
                     b = rep(2, N)),
          phi.unif = list(a = rep(3/max.dist, N),
                  b = rep(3/min.dist, N)))
```

We next set the parameters controlling the Adaptive MCMC algorithm (see `spPGOcc` section for details). Notice our specification of the starting tuning values is exactly the same as for `spPGOcc`. We assume the same initial tuning value for all species. However, the adaptive algorithm will allow for species specific tuning parameters, so these will be adjusted in the algorithm as needed.

```
batch.length <- 25
n.batch <- 400
n.burn <- 2000
n.thin <- 4
ms.tuning <- list(phi = 0.5)
n.omp.threads <- 1
# Values for reporting
verbose <- TRUE
n.report <- 50
```

Spatially explicit multispecies occupancy models are currently the most computationally intensive models fit by `spOccupancy`. Even for modest sized data sets, we encourage the use of NNGPs instead of full GPs when fitting spatially-explicit models to ease the computational burden of fitting these models. We fit the model with an NNGP below and summarize it using the `summary` function, where we specify that we want to summarize both species and community level parameters.

```
out.sp.ms <- spMsPGOcc(occ.formula = occ.ms.sp.formula,
               det.formula = det.ms.sp.formula,
               data = hbef2015,
               starting = ms.starting,
               n.batch = n.batch,
               batch.length = batch.length,
               accept.rate = 0.43,
               priors = ms.priors,
               cov.model = cov.model,
               tuning = ms.tuning,
               n.omp.threads = n.omp.threads,
               verbose = TRUE,
               NNGP = TRUE,
               n.report = n.report,
               n.burn = n.burn,
               n.thin = n.thin)
summary(out.sp.ms, level = 'both')
```

The resulting object `out.sp.ms` is a list of class `spMsPGOcc` consisting primarily of posterior samples of all community and species-level parameters, as well as some additional objects that are used for summaries, predictions, and model fit evaluation.

0.8.1 Convergence diagnostics

Convergence diagnostics proceed as we have seen with all previous `spOccupancy` model objects. Posterior samples are returned as `coda::mcmc` objects, so we can use functions like `plot` and `gelman.diag` to assess convergence.

```
plot(out.sp.ms$beta.comm.samples, density = FALSE)
# Species-specific effects have yet to converge
plot(out.sp.ms$beta.samples[, 1:4], density = FALSE)
```

0.8.2 Posterior predictive checks

We perform posterior predictive checks to assess Goodness of Fit using `ppcOcc` just as we have previously seen.

```
ppc.sp.ms.out <- ppcOcc(out.sp.ms, 'freeman-tukey', group = 2)
summary(ppc.sp.ms.out, level = 'both')
```

We see all Bayesian p-values are quite large, which is probably at least partly due to the fact that the chains have yet to converge.

0.8.3 Model selection using WAIC

Below we compute the WAIC using `waicOcc` and compare it to the WAIC for the non-spatial multispecies occupancy model.

```
waicOcc(out.sp.ms)
waicOcc(out.ms)
```

The WAIC for the spatial model is much larger than that for the nonspatial model, potentially indicating we don't need the additional complexities brought in by the species-specific spatial processes. However, we should not place a large emphasis on this since neither of the models has completely converged.

0.8.4 Prediction

Out-of-sample prediction with `spMsPGOcc` objects again uses the `predict` function given a set of covariates and spatial coordinates of unobserved locations. Here we predict values for all 12 species at every 50th cell of the total cells. We show results only for OVEN for comparison to results with the nonspatial multispecies occupancy model.

```r
X.0.ms <- cbind(1, hbefElev$val, hbefElev$val^2)
X.0.ms <- X.0.ms[seq(1, nrow(X.0.ms), by = 50), ]
coords.0 <- hbefElev[seq(1, nrow(hbefElev), by = 50), 2:3]
out.sp.ms.pred <- predict(out.sp.ms, X.0.ms, coords.0)
psi.pred.oven <- out.sp.ms.pred$psi.0.samples[, which(sp.names == 'OVEN'), ]
hist(apply(psi.pred.oven, 2, mean), xlab = 'Mean Occurrence Probability')
```

0.9 `intPGOcc`: single species integrated occupancy models

0.9.1 Convergence diagnostics

0.9.2 Posterior predictive checks

0.9.3 Model selection using WAIC

0.9.4 Prediction

0.10 `spIntPGOcc`: single species spatial integrated occupancy models

0.10.1 Convergence diagnostics

0.10.2 Posterior predictive checks

0.10.3 Model selection using WAIC

0.10.4 Prediction

References

Banerjee, Sudipto, Bradley P Carlin, and Alan E Gelfand. 2003. *Hierarchical Modeling and Analysis for Spatial Data.* Chapman; Hall/CRC.

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software* 67 (1): 1–48. https://doi.org/10.18637/jss.v067.i01.

Broms, Kristin M, Mevin B Hooten, and Ryan M Fitzpatrick. 2016. "Model Selection and Assessment for Multi-Species Occupancy Models." *Ecology* 97 (7): 1759–70.

Clark, Allan E, and Res Altwegg. 2019. "Efficient Bayesian Analysis of Occupancy Models with Logit Link Functions." *Ecology and Evolution* 9 (2): 756–68.

Datta, Abhirup, Sudipto Banerjee, Andrew O Finley, and Alan E Gelfand. 2016. "Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets." *Journal of the American Statistical Association* 111 (514): 800–812.

Doser, Jeffrey W., Wendy Leuenberger, T. Scott Sillett, Michael T. Hallworth, and Elise F. Zipkin. 2021. "Integrated Community Occupancy Models: A Framework to Assess Occurrence and Biodiversity Dynamics Using Multiple Data Sources." *arXiv Preprint arXiv:2109.01894*.

Finley, Andrew O, Abhirup Datta, and Sudipto Banerjee. 2020. "SpNNGP R Package for Nearest Neighbor Gaussian Process Models." *arXiv Preprint arXiv:2001.09111*.

Finley, Andrew O, Abhirup Datta, Bruce D Cook, Douglas C Morton, Hans E Andersen, and Sudipto Banerjee. 2019. "Efficient Algorithms for Bayesian Nearest Neighbor Gaussian Processes." *Journal of Computational and Graphical Statistics* 28 (2): 401–14.

Guélat, Jérôme, and Marc Kéry. 2018. "Effects of Spatial Autocorrelation and Imperfect Detection on Species Distribution Models." *Methods in Ecology and Evolution* 9 (6): 1614–25.

Heaton, Matthew J, Abhirup Datta, Andrew O Finley, Reinhard Furrer, Joseph Guinness, Rajarshi Guhaniyogi, Florian Gerber, et al. 2019. "A Case Study Competition Among Methods for Analyzing Large Spatial Data." *Journal of Agricultural, Biological and Environmental Statistics* 24 (3): 398–425.

Hobbs, N Thompson, and Mevin B Hooten. 2015. *Bayesian Models*. Princeton University Press.

Hooten, Mevin B, and N Thompson Hobbs. 2015. "A Guide to Bayesian Model Selection for Ecologists." *Ecological Monographs* 85 (1): 3–28.

Kéry, Marc, and J Andrew Royle. 2015. *Applied Hierarchical Modeling in Ecology: Volume 1: Prelude and Static Models*. Elsevier Science.

McCullagh, Peter. 2019. "Generalized Linear Models."

Polson, Nicholas G, James G Scott, and Jesse Windle. 2013. "Bayesian Inference for Logistic Models Using Pólya–Gamma Latent Variables." *Journal of the American Statistical Association* 108 (504): 1339–49.

Roberts, Gareth O, and Jeffrey S Rosenthal. 2009. "Examples of Adaptive Mcmc." *Journal of Computational and Graphical Statistics* 18 (2): 349–67.

Watanabe, Sumio. 2010. "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory." *Journal of Machine Learning Research* 11 (12).