Dynamic Occupancy Model

Matthew Rogan

2022-10-02

Running a multi-season occupancy analysis

Prepping odonate country checklists for integration with MOL

This script simulates data for a dynamic (multi-season) non-spatial occupancy study and then fits a series of models and evaluates them using an information criterion (AIC). For our purposes here, I am not including spatial autocorrelation or random effects in the model.

I simulated data using the *simDynocc()* function from R package *AHMbook* (https://cran.r-project.org/web/packages/AHMbook/AHMbook.pdf). It offers a tidy means of simulating all the data in one step with flexible levels of complexity.

I then fit a candidate set of models using maximum likelihood in R package *unmarked* (https://cran.r-project.org/web/packages/unmarked/vignettes/unmarked.html).

I will also use *ggplot2* to look at the outputs and some *tidyverse* packages (*tidyr, dplyr*) to manipulate some data for plotting. Only ggplot2 needs to be loaded but make sure the other packages are available before proceeding.

Step 1: Data simulation

I'll parameterize the simulation in three steps. First, I'll define the survey parameters: number of sites, number of seasons, and number of secondary occasions per season. Second, I'll specify true parameters of the population dynamics and detection process. Third, I'll specify covariate relationships.

For these purposes, I'll assume 100 sites sampled three times per year for ten years.

```
# load packages we need:
nSites <- 100
nYears <- 10
nOccas <- 3
```

Population parameters will be drawn from random uniform distributions, but I will choose limits consistent with a sparse, recolonizing population with moderate or high persistence and moderate detectability.

```
# mean probability of initial occupancy
psi_1 <- 0.2

# limits of uniform distribution for drawing persistence parameter
phi_lim <- c(0.6, 0.9)

# limits of uniform distribution for drawing colonization parameter
gamma_lim <- c(0.3, 0.5)

# limits of uniform distribution for drawing detection parameter
p_lim <- c(0.2, 0.5)
```

Now I will specify some covariate relationships for the four population parameters. I'll choose strong relationships for initial occupancy and persistence, a moderately relationship for colonization and no covariate or temporal effect for detection.

```
# strength of relationship for covariate of initial occupancy
beta_psi1 = 1

beta_phi = 1

beta_gam = 0.5

beta_p = 0
```

Now we're ready to simulate the data. *simDynocc()* returns a list with both true and simulated data components.

```
# seed for reproducibility
set.seed(102022)

sim_data <- AHMbook::simDynocc(
  nsites      = nSites,
  nyears      = nYears,
  nsurveys    = nOccas,
  mean.psi1   = psi_1,
  beta.Xpsi1  = beta_psi1,
  range.phi   = phi_lim,
  beta.Xphi   = beta_phi,
  range.gamma = gamma_lim,
  beta.Xgamma = beta_gam,
  range.p     = p_lim,
  beta.Xp     = beta_p,
  show.plots  = FALSE
)
```

Step 2: Fit occupancy models

Now that we have our data, I will reformat the simulated model inputs as an unmarked multiseason occupancy data format. This will consist of an observation matrix, a data frame of site covariates with one column per covariate, a list of matrices of annual site covariates, and the number of primary sampling periods. Here, I ignore

occasion-level predictors.

The simulation function returns observations as a three dimensional array but the unmarked occupancy format reads observations as a matrix where each row represents a site and each column is a sampling occasion such that the number of columns is equal to $nYearsXnOccas$.

```
# convert 3d observation array to 2d observation matrix
y3d <- sim_data$y

# create observation matrix of all zeros
Y <- matrix(0, nrow = nSites, ncol = nYears * nOccas)

# collapse primary and secondary occasions
for(i in 1:dim(y3d)[1]){
  Y[i,] <- c(y3d[i,,])
}
```

Now I'll format the covariate data and combine everything into the unmarked data object.

```
# site covariate - predictor variable for initial occupancy
site_covs = data.frame(xPsi = sim_data$Xpsi1)

# yearly site covariates - predictors for persistence and colonization
year_covs = list(xPhi = sim_data$Xphi,
                 xGam = sim_data$Xgamma)

# combine all data into unmarked data object
occuFrame <- unmarked::unmarkedMultFrame(y = Y,
                                         siteCovs = site_covs,
                                         obsCovs = NULL,
                                         yearlySiteCovs = year_covs,
                                         numPrimary = nYears)
```

Now we're ready to fit models. To keep things interesting, I'll specify a set of candidate models with a a number of univariate models, a "true" model, and an overfitted model that tests covariate effects on detection in addition to the state parameters. I'll first create a list of formulas then fit them in a separate step.

```
formulas <- list(
  list(~1, ~1, ~1, ~1),           # Null model - no covariate effects
  list(~xPsi, ~1, ~1, ~1),        # univariate effect on initial occupancy
  list(~1, ~xGam, ~1, ~1),        # univariate effect on colonization
  list(~1, ~1, ~xPhi, ~1),        # univariate effect on persistence
  list(~1, ~1, ~1, ~xPsi),        # univariate effect on detection using covariate for initial
 occupancy
  list(~xPsi, ~xGam, ~xPhi, ~1),  # "true" model
  list(~xPsi, ~xGam, ~xPhi, ~xPsi) # overfitted with non-existent covariate effect on detection
)

# name models
names(formulas) <- c("null",
                     "psi",
                     "gam",
                     "phi",
                     "p",
                     "true",
                     "global")
```

Now I'll fit the models as format them as a *fitList* object for easy comparison using AIC.

```
models <- unmarked::fitList(
  fits = lapply(formulas, function(terms, data){
    mod <- unmarked::colext(psiformula = terms[[1]],
                            gammaformula = terms[[2]],
                            epsilonformula = terms[[3]],
                            pformula = terms[[4]],
                            data = data,
                            se = TRUE)

    return(mod)
  },
  data = occuFrame)
)
```

Step 3: Evaluate models

First, let's look at model support using AIC.

```
unmarked::modSel(models)
```

```
##        nPars     AIC delta   AICwt cumltvWt
## true       7 2655.04  0.00 5.6e-01     0.56
## global     8 2655.49  0.45 4.4e-01     1.00
## phi        5 2687.33 32.29 5.4e-08     1.00
## psi        5 2690.18 35.14 1.3e-08     1.00
## gam        5 2712.60 57.56 1.8e-13     1.00
## null       4 2715.81 60.77 3.5e-14     1.00
## p          5 2717.72 62.68 1.4e-14     1.00
```

The good news is that the true model shows the most support, while the overfitted model shows nearly equal support. While I could average model parameters, when all supported models are nested, I prefer to select models based on parsimony or using a likelihood ratio test. In this particular case, however, the LR test is pointless since there's no decrease in AIC, but we can still run the test.

```
unmarked::LRT(models@fits$true, models@fits$global)
```

```
##      Chisq DF Pr(>Chisq)
## 1 1.547707  1  0.2134743
```

Predictably, there's no significant difference in the models so we are strongly justified in selecting the more parsimonious model. But just to be safe and make sure assumptions are met, let's run a goodness of fit test. The GOF function, parboot, does not seem to accept the model called from the fitList, so I will refit the model as a standalone object.

```
# refit selected model
mod <- unmarked::colext(psiformula = ~xPsi,
                        gammaformula = ~xGam,
                        epsilonformula = ~xPhi,
                        pformula = ~1,
                        data = occuFrame,
                        se = T)

# specify a set of test statistics
fitStats <- function(mod){
  observed <- unmarked::getY(mod@data)
  expected <- unmarked::fitted(mod)
  resids   <- unmarked::residuals(mod)
  sse      <- sum(resids^2)
  chisq    <- sum((sqrt(observed) - sqrt(expected))^2)

  out <- c(SSE = sse, Chisq = chisq)
  return(out)

}

gof <- unmarked::parboot(mod,
                         fitStats,
                         parallel = FALSE,
                         nsim = 100,
                         seed = 102022)

gof
```

```
##
## Call:
## unmarked::parboot(object = mod, statistic = fitStats, nsim = 100, seed = 102022, parallel = F
ALSE)
##
## Parametric Bootstrap Statistics:
##        t0 mean(t0 - t_B) StdDev(t0 - t_B) Pr(t_B > t0)
## SSE   440           7.54             18.8        0.386
## Chisq 608           6.81             20.6        0.406
##
## t_B quantiles:
##        0% 2.5% 25% 50% 75% 97.5% 100%
## SSE   366  394 423 434 445   461  472
## Chisq 526  558 590 603 615   633  639
##
## t0 = Original statistic computed from data
## t_B = Vector of bootstrap samples
```

Fit is adequate. So let's look at the actual model estimates…

```
print(mod@estimates)
```

```
## Initial:
##             Estimate    SE     z  P(>|z|)
## (Intercept)    -2.12 0.592 -3.59 0.000333
## xPsi            2.13 0.660  3.22 0.001285
##
## Colonization:
##             Estimate    SE     z P(>|z|)
## (Intercept)   -0.459 0.145 -3.17 0.00151
## xGam           0.392 0.143  2.74 0.00616
##
## Extinction:
##             Estimate    SE     z  P(>|z|)
## (Intercept)   -1.128 0.237 -4.77 1.88e-06
## xPhi          -0.923 0.199 -4.64 3.51e-06
##
## Detection:
##  Estimate     SE     z  P(>|z|)
##    -0.683 0.0803 -8.51 1.76e-17
```

Now let's plot the true occupancy versus observed and estimated occupancy. For estimated occupancy, we'll use the smoothed expected proportion, multiplied by 100 to get the expected number of occupied sites. To derive standard error, I'll use non-parametric bootstrapping. To simplify plotting, I combine everything into one data frame.

```
library(tidyr, quietly = T)
```

```
## Warning: package 'tidyr' was built under R version 4.1.3
```

```
library(dplyr, quietly = T)
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#non-parametric bootstrapping
bootSamp <- unmarked::nonparboot(mod,
                                 B = 50)

site_occu <- data.frame(year     = 1:10,
                        True     = colSums(sim_data$z), # actual number of occupied sites
                        Observed = colSums(apply(sim_data$y, c(1,3), max)), #observed number of
 occupied sites
                        Expected = unmarked::smoothed(mod)[2,]*100,
                        SE       = bootSamp@smoothed.mean.bsse[2,]*100) %>% # expected number of
occupied sites
  # Convert to long format
  pivot_longer(cols = c("True", "Observed", "Expected"),
               values_to = "Sites",
               names_to = "Type") %>%
  # remove SE for True and Observed site occupancy
  mutate(SE = if_else(Type == "Expected",
                      SE,
                      as.numeric(NA)))

# we also need a dataframe with just the expected and SE
```

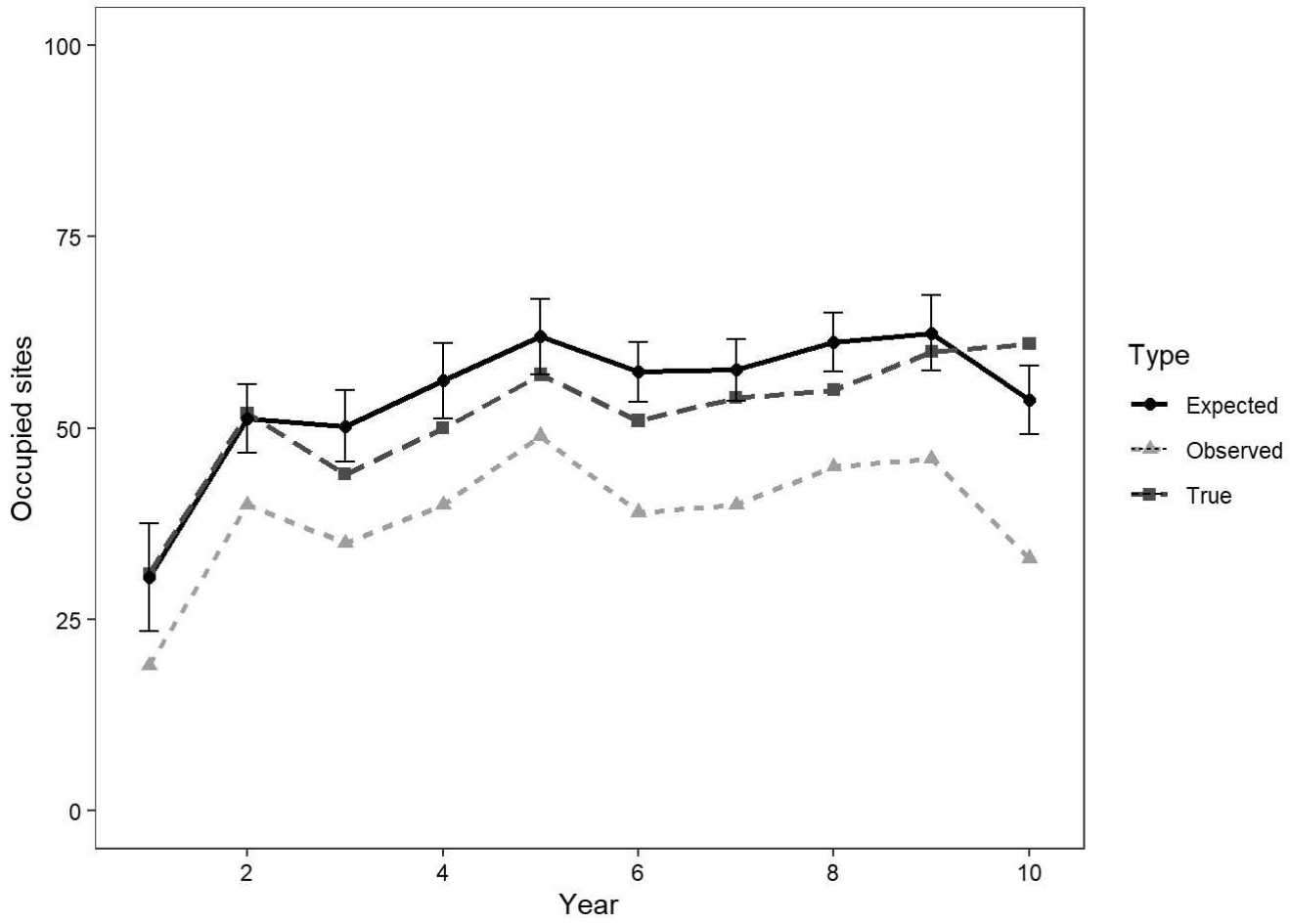
Now let's plot:

```
library(ggplot2, quietly = T)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
plot <- ggplot(site_occu, aes(x = year, y = Sites, color = Type, linetype = Type)) +
  geom_line(size = 1) +
  geom_point(aes(shape = Type), size = 2) +
  geom_errorbar(aes(ymin = Sites - SE, ymax = Sites + SE), width = 0.2, color = "black") +
  ylab("Occupied sites") +
  xlab("Year") +
  scale_color_manual(values = c("black", "dark orange", "red")) +
  scale_x_continuous(breaks = seq(2, 10, 2)) +
  ylim(0, 100) +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.text = element_text(color = "black"))

plot
```



So we see that we track the initial increase in occupancy well, but then marginally overestimate occupancy for most of the remainder of the sampling period by about one standard error.