Appendix 2

2025-11-18

Contents

So you wanna fit an integrated species distribution model? You've come to the right place!

NEED SOME SORT OF INTRO HERE.

- The first part of this sets up the range, imports species and covariate data, scales covariates, and sets everything up for NIMBLE.
- The second part fits the NIMBLE model
- The third part summarizes the NIMBLE output and creates output figures.

Let's first load the libraries we'll need to set up, visualize, fit, and summarize our data.

```
# Load libraries ----
library(tidyverse)
library(sf)
library(nimble)

## To install SpFut.flexiSDM or check for updates, use the commented code below:
# remotes::install_github("rileymummah/SpFut.flexiSDM", build_vignettes = T)
library(SpFut.flexiSDM)
```

Model specifications

To load data or fit a model, you must first create a .csv file that outlines the model specifications. We call ours `model-specs.csv`. Let's load the .csv and take a look at how we structured it.

```r
mods <- read.csv("code/model-specs.csv")

head(mods, n=2)
```

```
##   number sp.code  model mem.nim mem.sum year.start year.end buffer sp.auto
## 1      1    RACA tau0.1   30000   40000       1994     2025  50000    TRUE
## 2      2    RACA   tau1   30000   40000       1994     2025  50000    TRUE
##   zero_mean tau coarse.grid cont.grid    covs.PO covs.inat
## 1      TRUE 0.1       FALSE     FALSE traveltime      <NA>
## 2      TRUE   1       FALSE     FALSE traveltime      <NA>
##                                    covs.lin covs.quad check.covs
## 1 sqrtarea_small, sqrtarea_medium, footprint TRI, tmin      FALSE
## 2 sqrtarea_small, sqrtarea_medium, footprint TRI, tmin      FALSE
##   covs.int.factor reference covs.int.cont Bpriordist Bpriorvar1 Bpriorvar2
## 1              NA        NA            NA      dnorm          0          1
## 2              NA        NA            NA      dnorm          0          1
##   filter.region spat.bal coordunc coordunc_na.rm block.rows block.cols
## 1          TRUE     TRUE     1000           TRUE          5          5
## 2          TRUE     TRUE     1000           TRUE          5          5
##   block.folds   iter thin region.sub lon.hi lon.lo lat.hi lat.lo project
## 1           3 100000    5      FALSE     NA     NA     NA     NA      NA
## 2           3 100000    5      FALSE     NA     NA     NA     NA      NA
##   exclude.dataset
## 1              NA
## 2              NA
```

The following table describes the column names, data type, and description # FINISH TABLE

Column Name	Type	Description
number	numeric	Model number
sp.code	character	Four digit species code
model	character	Model name
mem.nim	numeric	Memory allocation (in MB) for NIMBLE
mem.sum	numeric	Memory allocation (in MB) for parallelized summarization. Could be omitted if not using an HPC.
year.start	numeric	Starting year of data inclusion
year.end	numeric	Ending year of data inclusion
buffer	numeric	Size of buffer around range edge (in km)
sp.auto	logical	Should a spatial model be included?
zero_mean	logical	Should a zero mean assumption be included in the spatial model
tau	numeric/character	Assign a fixed value or a prior distribution for precision (τ)
coarse.grid	logical	Should a coarse grid be used for the spatial model? Usually desired for large-range species
cont.grid	logical	Should the grid be restricted to continuous cells? (This will remove any groups of non-contiguous cells)
covs.PO	character	A comma-separated list of covariates for modeling PO sampling effort. These must match the covariate names in the data.
covs.inat	character	A comma-separated list of covariates for modeling iNaturalist sampling effort. These must match the covariate names in the data.

Column Name	Type	Description
covs.lin	character	A comma-separated list of linear covariates for modeling species relative abundance. These must match the covariate names in the data.
covs.quad	character	A comma-separated list of quadratic covariates for modeling species relative abundance. These must match the covariate names in the data.
check.covs	logical	Should covariate selection remove multicollinearity?
covs.int.factor		
reference		
covs.int.cont		
Bpriordist		
Bpriorvar1		
Bpriorvar2		
filter.region	logical	
spat.bal	logical	Should the PO data be spatially balanced?
coordunc	numeric	The level of acceptable coordinate uncertainty
coordunc_na.rm	logical	Should coordinates with uncertainty beyond `coordunc` be removed?
block.rows	numeric	The number of rows for cross-validation blocks
block.cols	numeric	The number of columns for cross-validation blocks
block.folds	numeric	The number of cross-validation block groups
iter	numeric	The number of iterations to run an MCMC chain
thin	numeric	Thin the MCMC chains by this parameter
region.sub	logical	Only use the centroid of the region with a radius equal to the buffer size?
lon.hi	numeric	High longitude value to restrict the range
lon.lo	numeric	Low longitude value to restrict the range
lat.hi	numeric	High latitude value to restrict the range
lat.lo	numeric	Low latitude value to restrict the range
project	numeric	The number of future projections (otherwise, NA)
exclude.dataset	character	A comma-separated list of datasets to exclude from analysis

We have set up the scripts in the flexiSDM workflow so that a minimum number of parameters needs to be changed among them. To run this script in full (using our file arrangement), you only need to edit the following parameters:

```
nums.do <- 2 # model number to run
block <- 'none' # block to run ('none', 1, 2, or 3)
local <- 1 # are you running the code locally (1) or on an HPC (0)?
a <- 1 # Fixed for indexing a vector below
```

For the purposes of this demonstration, we're going TO FIT X MODEL WITH THESE ASSUMPTIONS

Side note: High performance computing

We almost exclusively run these models on a high performance computing (HPC) cluster. We clone our repository directly to the HPC to ensure we have the same file structure and no version conflicts. We've set up the code so we can seamlessly move between running the workflow locally and on our HPC.

To run the first steps of the workflow via `01-flexiSDM.R`, we call the script `01-flexiSDM.sh`, which takes 3 inputs:

- Model number
- Block (defined by an array in the script)
- Local (1/0)

We have simplified this further by writing a wrapper called `01-HPC.sh`, which only takes one input: model number. Within the script, `local` is set to 0, indicating a run on an HPC.

The inputs provided to the `.sh` scripts are converted to inputs to the `01-flexiSDM.R` script and are interpreted below:

```
# Converts command arguments to something interpretable by R
args = commandArgs(trailingOnly = TRUE)

if (length(args) > 0) {

  # Model number to run
  nums.do = as.numeric(args[1])

  # Blocks can be run as single jobs or as arrays
  block = as.numeric(args[2])

  # The block input is numeric only. Convert 4 = 'none'
  if (block == 4) {
    block <- 'none'
  }

  # Running locally? Yes = 1
  local = as.numeric(args[3])

  # If running on an HPC, set the working directory to the project home directory
  if (local == 0) {
    setwd('/home/directory/')
  }
}
```

Load model specifications

Then we can use our model specifications to define a series of inputs for the setup script (found in `01-flexiSDM.R`).

```
## Set up model variables
mods <- filter(mods, number %in% nums.do)

## Create all the combinations of model number and cross-validation block
tmp <- expand.grid(block.out = block, number = unique(mods$number))
mods <- full_join(mods, tmp, by = c("number"))
# CAN WE SIMPLIFY THIS SO THE CODE ONLY RUNS ONE MODEL NUMBER-BLOCK COMBO?
# THEN WE COULD REMOVE 'A' BELOW AND FIX IT AT 1

## Get variables for model from model-specs.csv
number <- mods$number[a] # model number
sp.code <- mods$sp.code[a] # 4-digit species code
model <- mods$model[a] # model name
```

```r
sp.auto <- mods$sp.auto[a] # include spatial autocorrelation?
coarse.grid <- mods$coarse.grid[a] # use a coarse grid?
cont.grid <- mods$cont.grid[a] # restrict to only a continuous grid?
year.start <- mods$year.start[a] # start year for data
year.end <- mods$year.end[a] # end year for data
buffer <- mods$buffer[a] # buffer size (km)
filter.region <- mods$filter.region[a]
spat.bal <- mods$spat.bal[a] # include spatial balancing?
coordunc <- mods$coordunc[a] # level of coordinate uncertainty to include (km)
coordunc_na.rm <- mods$coordunc_na.rm[a] # filter by coordinate uncertainty?
block.folds <- mods$block.folds[a] # how many groups of blocks?
block.rows <- mods$block.rows[a] # how many rows of blocks?
block.cols <- mods$block.cols[a] # how many columns of blocks?
block.out <- mods$block.out[a] # IS THIS PART OF THE SIMPLIFICATION

if (block.out == "none") {
  blockname <- "full" # rename the block to 'full' if not doing cross-validation
} else {blockname <- block.out} # otherwise, keep the block number

# names of datasets to exclude (e.g., iNaturalist) - must match dataset naming
exclude.dataset <- unlist(str_split(mods$exclude.dataset[a], pattern = ", "))

## ICAR parameters
zero_mean <- mods$zero_mean[a] # zero mean assumption?
tau <- mods$tau[a] # precision (tau) can be a fixed value or a prior

## MCMC parameters
iter <- mods$iter[a] # number of MCMC iterations
thin <- mods$thin[a] # number to thin by
burnin <- floor(iter*0.75) # burnin to discard

## Change of region
region.sub <- mods$region.sub[a] # only estimate for a subregion?
lat.hi <- mods$lat.hi[a] # high value of latitude range
lat.lo <- mods$lat.lo[a] # low value of latitude range
lon.hi <- mods$lon.hi[a] # high value of longitude range
lon.lo <- mods$lon.lo[a] # low value of longitude range

## Future projections
project <- mods$project[a] # how many projections?
if (block.out != "none") {
  project <- 0 # no projections for cross-validation blocks
}

## Covariates
# list of PO covariates
covs.PO <- unlist(str_split(mods$covs.PO[a], pattern = ", "))
# list of iNaturalist covariates
covs.inat <- unlist(str_split(mods$covs.inat[a], pattern = ", "))
# list of linear covariates for state model
covs.lin <- unlist(str_split(mods$covs.lin[a], pattern = ", "))
# list of quadratic covariates for state model
covs.quad <- unlist(str_split(mods$covs.quad[a], pattern = ", "))
```

```
# SHOULD WE KEEP THIS IN? WE'RE NOT CURRENTLY USING IT BUT COULD??
# covs.int.factor <- unlist(str_split(mods$covs.int.factor[a], pattern = ", "))
# reference <- mods$reference[a]
# covs.int.cont <- unlist(str_split(mods$covs.int.cont[a], pattern = ", "))
check.covs <- mods$check.covs[a] # covariate selection to remove multicollinearity?

# Define the prior for the state model coefficients
Bpriordist <- mods$Bpriordist[a]
Bpriorvar1 <- mods$Bpriorvar1[a]
Bpriorvar2 <- mods$Bpriorvar2[a]
# WE COULD SIMPLIFY THIS TO LOOK LIKE TAU

process.intercept <- F # SHOULD RENAME TO state.intercept??
```

Now that we have our model specifications loaded, we're ready to start loading data and covariates. First, let's ensure that we have an output folder specific to the model we are fitting.

```
# Make output folder ----
out.dir <- paste0("outputs/", number, "_", sp.code, "_", model, "/")
if (dir.exists(out.dir) == F) {
  dir.create(out.dir)
}
```

Step 1: Define the region

We define the region of inference using the GAP and IUCN ranges. Other boundaries could be used to create the limits of the range. We saved the IUCN and GAP range boundaries in folders in `data/sp.code`, so that they can be called by the `get_range` function. We have also downloaded the US Census state boundary shapefile (`cb_2018_us_state_500k.shp`) to dictate the national and state borders.

Step 2: Set up cross-validation blocks (if using)

```
## Set up cross validation blocks
sb <- blockCV::cv_spatial(x = region$sp.grid,
                          rows_cols = c(block.rows, block.cols),
                          k = block.folds,
                          hexagon = F,
                          selection = "systematic",
                          biomod2 = F,
                          plot = F,
                          report = F)

## LANE CAN YOU GIVE A ONE SENTENCE AS TO WHAT THIS LINE IS DOING?
spatblocks <- make_CV_blocks(region, rows = block.rows, cols = block.cols, k = block.folds)

## Set up training and test sets based on cross validation blocks
if (block.out == "none") { # If fitting the full model, then there are
  test.i <- c()            # no test data
  train.i <- region$sp.grid$conus.grid.id
```

```
} else {
  block1 <- spatblocks %>% filter(folds == block.out)

  ## Find grid.ids for test block, everything else is train
  # test.i1 <- st_intersection(region$sp.grid, block1) DELETE???
  test.i <- st_intersection(region$sp.grid, block1) %>%
             pull(conus.grid.id) %>%
             unique()
  train.i <- filter(region$sp.grid, conus.grid.id %in% test.i == F) %>%
              pull(conus.grid.id)
}
```

Step 3: Load species data

```
## Compile the name of all files that have data for the species of interest
allfiles <- read.csv("data/dataset-summary-full.csv")

head(allfiles, 2)

tmp1 <- gsub("[|]", "$|^", sp.code.all) # THIS LINE WON'T RUN UNTIL THE SP.CODE.ALL IS CREATED - DO WE
tmp2 <- paste0("^", tmp1, "$")
allfiles <- allfiles[grep(tmp2, allfiles$species),] %>%
            select(name, file) %>%
            distinct() %>%
            filter(name %in% exclude.dataset == F)
```

Step 4: Plot species data

```
if (block == "none") {
  title <- ", full model"
} else {
  title <- paste0(", excluding block ", block)
}
```

Load covariate data