Appendix 2

Last updated: 2025-11-25

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)
library(SpFut.covariates)

## There are two custom functions that are needed to fit the data in NIMBLE
source("code/FXN-nimbleParallel.R")
```

01-flexiSDM.R

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.

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

head(mods)
```

```
##   number sp.code    model mem.nim mem.sum year.start year.end buffer sp.auto
## 1      1    RACA     iSDM   30000   40000       1994     2025  50000    TRUE
## 2      2    GPOR     iSDM   30000   60000       1994     2025  50000    TRUE
## 3      3    RACA subrange   30000   40000       1994     2025  50000    TRUE
##   zero_mean tau coarse.grid cont.grid    covs.PO covs.inat
## 1      TRUE   1       FALSE     FALSE traveltime      <NA>
## 2      TRUE   1        TRUE     FALSE traveltime       ORM
## 3      TRUE   1       FALSE     FALSE traveltime      <NA>
##                                   covs.lin covs.quad check.covs     Bprior
## 1 sqrtarea_small, sqrtarea_medium, footprint TRI, tmin      FALSE dnorm(0,1)
## 2              streamLength.km, prec, forest elevation      FALSE dnorm(0,1)
## 3 sqrtarea_small, sqrtarea_medium, footprint TRI, tmin      FALSE 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
```

```
## 3          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
## 3           3  10000    5       TRUE     NA     NA     NA     NA      NA
##   exclude.dataset
## 1              NA
## 2              NA
## 3              NA
```

The following table describes the column names, data type, and description found in `model-specs.csv`. Not all columns are required depending how you intend to fit the model (i.e., local computer vs. high performance computing cluster).

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. Could be omitted if not using an HPC.
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 in BUGS language for precision (τ; e.g., dgamma(5,5))
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.
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?
Bprior	character	Assign a prior distribution in BUGS lanaguage for the β coefficients in the state model (e.g., dnorm(1,0))
filter.region	logical	Should the region be spatially filtered?
spat.bal	logical	Should the PO data be spatially balanced?
coordunc	numeric	The level of acceptable coordinate uncertainty

Column Name	Type	Description
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 <- 3 # 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)?
```

For the purposes of this demonstration, we're going to fit a model to the centroid of the Cascades frog (RACA) range with a 50,000km buffer. This will allow us to easily visualize the datasets and spatial grid, while maintaining local computational capabilities.

Load model specifications

Now that we have loaded our model specifications, we can use them 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)

## Get variables for model from model-specs.csv
number <- mods$number[1] # model number
sp.code <- mods$sp.code[1] # 4-digit species code
model <- mods$model[1] # model name
sp.auto <- mods$sp.auto[1] # include spatial autocorrelation?
coarse.grid <- mods$coarse.grid[1] # use a coarse grid?
cont.grid <- mods$cont.grid[1] # restrict to only a continuous grid?
year.start <- mods$year.start[1] # start year for data
year.end <- mods$year.end[1] # end year for data
buffer <- mods$buffer[1] # buffer size (km)
filter.region <- mods$filter.region[1] # filter the region? LANE - FILTER BY WHAT?
spat.bal <- mods$spat.bal[1] # include spatial balancing?
coordunc <- mods$coordunc[1] # level of coordinate uncertainty to include (km)
coordunc_na.rm <- mods$coordunc_na.rm[1] # filter by coordinate uncertainty?
```

```
block.folds <- mods$block.folds[1] # how many groups of blocks?
block.rows <- mods$block.rows[1] # how many rows of blocks?
block.cols <- mods$block.cols[1] # how many columns of blocks?
block.out <- block # define the block to setup

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[1], pattern = ", "))

## ICAR parameters
zero_mean <- mods$zero_mean[1] # zero mean assumption?
tau <- mods$tau[1] # precision (tau) can be a fixed value or a prior

## MCMC parameters
iter <- mods$iter[1] # number of MCMC iterations
thin <- mods$thin[1] # number to thin by
burnin <- floor(iter*0.75) # burnin to discard

## Change of region
region.sub <- mods$region.sub[1] # only estimate for a subregion?
lat.hi <- mods$lat.hi[1] # high value of latitude range
lat.lo <- mods$lat.lo[1] # low value of latitude range
lon.hi <- mods$lon.hi[1] # high value of longitude range
lon.lo <- mods$lon.lo[1] # low value of longitude range

## Future projections
project <- mods$project[1] # 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$covs.PO[1], pattern = ", "))
# list of iNaturalist covariates
covs.inat <- unlist(str_split(mods$covs.inat[1], pattern = ", "))
# list of linear covariates for state model
covs.lin <- unlist(str_split(mods$covs.lin[1], pattern = ", "))
# list of quadratic covariates for state model
covs.quad <- unlist(str_split(mods$covs.quad[1], pattern = ", "))
check.covs <- mods$check.covs[1] # covariate selection to remove multicollinearity?

## Define the prior for the state model coefficients
Bprior <- mods$Bprior[1]

## Combine linear and quadratic distribution covariates into covs.z
covs.z <- c(covs.lin, covs.quad)
if ("" %in% covs.z) {
  covs.z <- covs.z[-which(covs.z == "")]
}
```

```r
if (NA %in% covs.z) {
  covs.z <- covs.z[-which(is.na(covs.z))]
}
```

Now we're ready to set up the region and load data and covariates. First, let's ensure that we have output and data folders specific to the model we are fitting.

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

## Make data folder
data.dir <- paste0("data/", number, "_", sp.code, "_", model, "/")
if (dir.exists(data.dir) == F) {
  dir.create(data.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. We have already overlaid a continental US hexbin grid and associated covariates to grid cells (`grid-covar`). Unfortunately, this file is too large to upload to GitHub, so we demonstrate how we create the region and provide the file in `data.dir/region.rds` to bypass this step.

```r
## Generating the region requires files that are too big for GitHub just read it in
if (file.exists(paste0(data.dir, "region.rds"))) {
  region <- read_rds(file = paste0(data.dir, "region.rds"))
} else {
  ## Load ranges
  range.path <- c(paste0("data/", sp.code, "/GAP/"),
                  paste0("data/", sp.code, "/IUCN/"))
  range.name <- c("GAP", "IUCN")
  rangelist <- get_range(range.path,
                         range.name,
                         crs = 4326)
  ## USA boundary
  exclude <- c("Alaska", "Hawaii", "Commonwealth of the Northern Mariana Islands",
               "American Samoa", "United States Virgin Islands", "Guam", "Puerto Rico")
  usa <- st_read("../species-futures/data/USA/maps/cb_2018_us_state_500k/cb_2018_us_state_500k.shp") %>%
          filter((NAME %in% exclude) == F) %>%
          st_union()
  ## CONUS grid
  load("../species-futures/data/USA/grid-covar.rdata") # Too big for GitHub

  ## Region
  region <- make_region(rangelist,
                        buffer = buffer,
```

```
                          crs = 3857,
                          sub = region.sub,
                          boundary = usa,
                          grid = conus.grid,
                          rm.clumps = T,
                          clump.size = 50)

  write_rds(region, file = paste0(data.dir, "region.rds"))
}
```

There is one additional step here if you intend to use the coarse spatial grid. We must redefine the spatial grid by grouping hexbins together in sets of seven via the `make_spatkey` function. Otherwise, `spatRegion` is set to NULL.

```
## If using the coarse spatial grid, redefine the grid and plot the resulting grid.
if (coarse.grid == T) {
  spatRegion <- suppressWarnings(make_spatkey(region$sp.grid))

  # Print spatial grid
  if (block.out == 'none') {
    pl <- ggplot(spatRegion$spat.grid) + geom_sf() + theme_bw()
    ggsave(pl, file = paste0(out.dir, "2_inputmap-e_spatGrid.jpg"), height = 8, width = 10)
  }
} else {
  spatRegion <- NULL
}
```

Step 2: Set up cross-validation blocks

We have set up the code so that this section needs to be run even if you do not intend to use cross-validation blocks. If you do not desire, cross-validation blocks, all of the data are considered "training data". Otherwise, the data are split into "test" and "training" data depending on the cross-validation fold (1, 2, or 3) that is selected to run.

```
## Set up cross validation blocks
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 in the training block
  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)
}
```

After the cross-validation blocks are assembled and the data are assigned to "training" and "test" sets, we make a `gridkey` that allows us to index the data in NIMBLE by assigning the data to one of two groups. MOVE THIS TO NIMBLE CODE AREA?

```
# Make gridkey ----
gridkey <- select(region$sp.grid, conus.grid.id) %>%
            st_drop_geometry() %>%
            mutate(grid.id = 1:nrow(.),
                   group = case_when(conus.grid.id %in% train.i ~ "train",
                                     conus.grid.id %in% test.i ~ "test"))
```

Step 3: Load species data

We're now ready to load the species data. We have assembled a full amphibian species list (`model-specieslist.csv`) from which we pull out the common name, any species code that could refer to the species of interest, and the scientific name. We demonstrate how we did this process, but these values could be set manually or pulled from a list. We additionally define the species type (e.g., Frog/Toad or Salamander) based off the genus name. This delineation is used later for covariate data.

```
## Define species codes
codeKey <- read.csv("data/model-specieslist.csv")

## Common name
common <- codeKey %>%
            filter(DS.code == sp.code) %>%
            pull(SSAR.common)
common
```

```
## [1] "Cascades Frog"
```

```
## All species codes used (some species are complexes or have subspecies so multiple codes exist)
sp.code.all <- codeKey %>%
                filter(DS.code == sp.code) %>%
                pull(all.codes)
sp.code.all
```

```
## [1] "RACA"
```

```
## Scientific name
sciname <- codeKey %>%
            filter(DS.code == sp.code) %>%
            pull(SSAR.scientific)
sciname
```

```
## [1] "Rana cascadae"
```

```
## Save genus name to assign Frog/Toad or Salamander
gen <- stringr::word(sciname, 1)

if (gen %in% c("Acris", "Anaxyrus", "Aquarana", "Ascaphus",
               "Craugastor", "Dendrobates", "Dryophytes", "Eleutherodactylus",
```

```
                "Gastrophryne", "Glandirana", "Hyla", "Hypopachus", "Incilius",
                "Leptodactylus", "Lithobates", "Osteopilus",
                "Pseudacris", "Rana", "Rhinella", "Rhinophrynus",
                "Scaphiopus", "Smilisca", "Spea", "Xenopus")) spp.type <- "Frog/Toad"
if (gen %in% c("Ambystoma", "Amphiuma", "Aneides", "Batrachoseps",
                "Cryptobranchus", "Desmognathus", "Dicamptodon",
                "Ensatina", "Eurycea", "Gyrinophilus", "Hemidactylium",
                "Hydromantes", "Necturus", "Notophthalmus",
                "Phaeognathus", "Plethodon", "Pseudobranchus",
                "Pseudotriton", "Rhyacotriton", "Siren",
                "Stereochilus", "Taricha", "Urspelerpes")) spp.type <- "Salamander"

gen; spp.type
```

```
## [1] "Rana"
```

```
## [1] "Frog/Toad"
```

```
# get all files that have data for that species
allfiles <- read.csv("data/00-data-summary-flexiSDM.csv") %>%
  filter(Species == sp.code) %>%
  rename(file.name = Data.Swamp.file.name,
         file.label = Name,
         covar.mean = Covar.mean,
         covar.sum = Covar.sum,
         data.type = Type.true) %>%
  select(file.name, file.label, covar.mean, covar.sum, data.type, PO.extent)

head(allfiles)
```

```
##                      file.name  file.label covar.mean          covar.sum
## 1 RACA_ARMIOSF_count_2010_2023  ARMI Count            duration, area_sqm
## 2        RACA_BLM_PO_1992_2021 BLM Surveys
## 3                RACA_iNat_PO iNaturalist
## 4               RACA_museum_PO      Museum
## 5       RACA_NCCN_PO_1984_2005        NCCN
## 6       RACA_NRIS_PO_1901_2022     FS NRIS
##   data.type PO.extent
## 1     Count      <NA>
## 2        PO        OR
## 3        PO     CONUS
## 4        PO     CONUS
## 5        PO        WA
## 6        PO     CONUS
```

NEED THESE TO LOAD SPECIES DATA NEED A LIST SAME LENGTH AS FILE VECTOR WHERE EACH ELEMENT IS A DETECTION COVARIATE

```
species.data <- load_species_data(sp.code,
                                  sp.code.all,
                                  file.info = allfiles,
                                  file.path = "data/data-ready/",
```

```
                                region = region,
                                filter.region = filter.region,
                                year.start = year.start,
                                year.end = year.end,
                                coordunc = coordunc,
                                coordunc_na.rm = coordunc_na.rm,
                                spat.thin = spat.bal,
                                keep.conus.grid.id = gridkey$conus.grid.id[which(gridkey$group == "tra
```

```
##
## File 1: ARMI Count....Loading file
## Removing 4000 observations that are outside of the study region
## No data for this analysis
##
## File 2: BLM Surveys....Loading file
## Removing 148 observations that are outside of the study region
## Removing 493 observations before 1994 or after 2025
## Removing 237 observations for spatial thinning
##
## File 3: iNaturalist....Loading file
## Removing 1674 observations that are outside of the study region
## Removing 79 observations whose coordinate uncertainty is above the threshold of 1000
## No data for this analysis
##
## File 4: Museum....Loading file
## Removing 16 observations that are outside of the study region
## No data for this analysis
##
## File 5: NCCN....Loading file
## Removing 52680 observations that are outside of the study region
## No data for this analysis
##
## File 6: FS NRIS....Loading file
## Removing 54500 observations that are outside of the study region
## Removing 168 observations before 1994 or after 2025
## Removing 118 observations for spatial thinning
##
## File 7: ORBIC....Loading file
## Removing 112 observations that are outside of the study region
## Removing 2 observations before 1994 or after 2025
## Removing 0 observations for spatial thinning
##
## File 8: USGS RACA assessment - current....Loading file
## Removing 390 observations that are outside of the study region
## Using  effort  as covariate(s)
##
## File 9: USGS RACA assessment - historic....Loading file
## Removing 91 observations that are outside of the study region
## Using  yday  as covariate(s)
##
## File 10: PNW Salamanders....Loading file
## Removing 177 observations that are outside of the study region
## Using  distance  as covariate(s)
```

```
##
## File 11: WDWF....Loading file
## Removing 1195 observations that are outside of the study region
## No data for this analysis
```

Step 4: Plot species data

WE PROBABLY DON'T NEED TO INCLUDE ALL OF THE FIGURES THAT WE MAKE, SO LET'S CHOOSE A FEW TO HIGHLIGHT

```
## Quick title to be used across figures
if (block == "none") {
  title <- ", full model"
} else {
  title <- paste0(", excluding block ", block)
}

## Plot data samples
pl <- map_species_data(region = region,
                       species.data = species.data,
                       year.start = year.start,
                       year.end = year.end,
                       plot = "samples",
                       plot.region = T,
                       details = F,
                       title = paste0(common, " (", sp.code, ")", title))
```

```
## Reading 'ne_10m_lakes.zip' from naturalearth...
```

```
## Save plot
ggsave(pl, file = paste0(out.dir, "2_inputmap-a_data-", blockname, ".jpg"),
       height = 8, width = 10)
```

```
## Cross-validation block-specific plots
if (block.out != "none") {

  pl <- map_species_data(region = region,
                         species.data = species.data,
                         year.start = year.start,
                         year.end = year.end,
                         plot = "samples",
                         plot.blocks = T,
                         blocks = spatblocks[which(spatblocks$folds == block),],
                         plot.region = T,
                         details = F,
                         title = paste0(common, " (", sp.code, ")", title))
  ggsave(pl, file = paste0(out.dir, "2_inputmap-c_blocks-", blockname, ".jpg"),
         height = 8, width = 10)
} else {
  # Plot
  pl <- map_species_data(region = region,
```

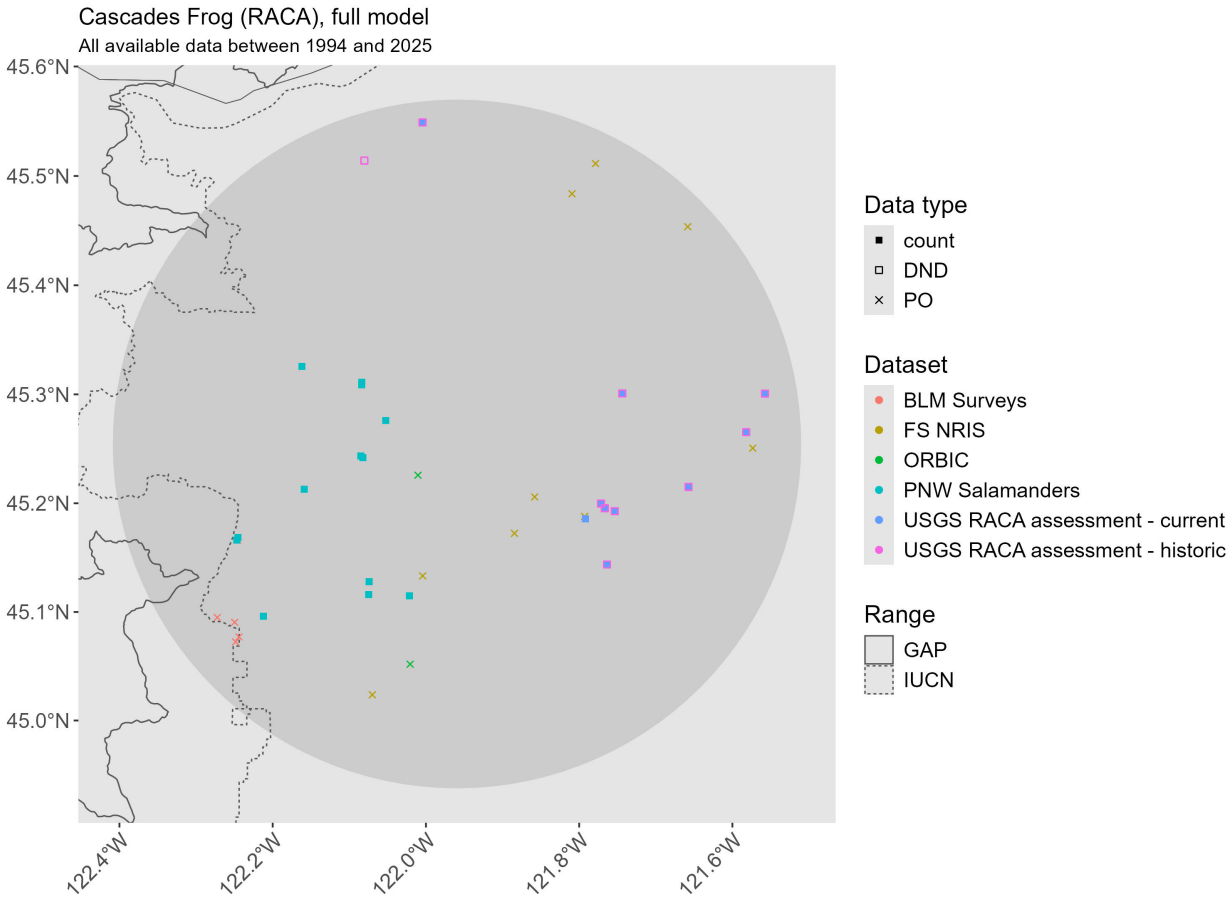

Figure 1: IMAGE

```
                          species.data = species.data,
                          year.start = year.start,
                          year.end = year.end,
                          plot = "samples",
                          plot.blocks = T,
                          blocks = spatblocks,
                          plot.region = T,
                          details = F,
                          title = paste0(common, " (", sp.code, ")", title))
  ggsave(pl, file = paste0(out.dir, "2_inputmap-c_blocks-", blockname, ".jpg"),
         height = 8, width = 10)
}
```

```
## Reading 'ne_10m_lakes.zip' from naturalearth...
```

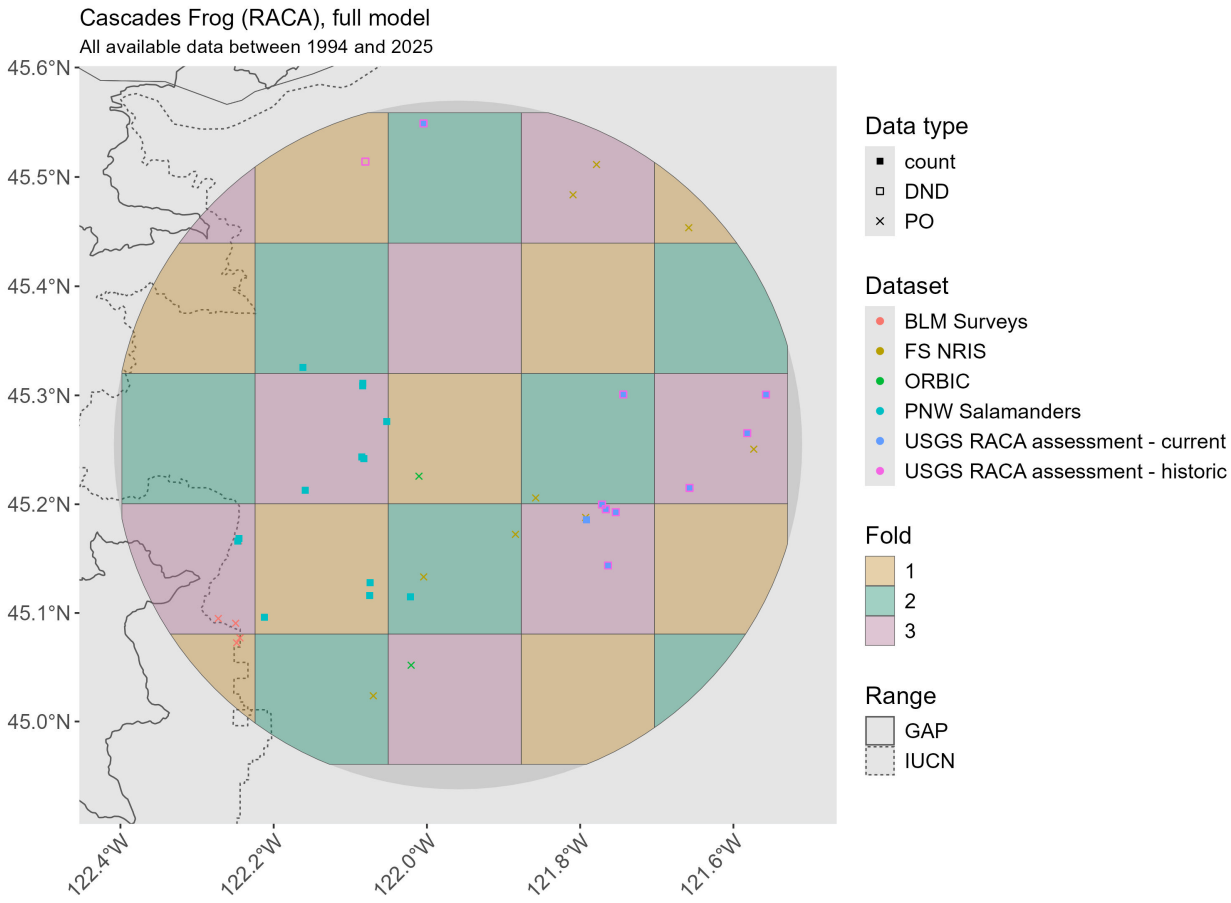

Figure 2: IMAGE

Step 5: Load covariate data

ONLY GOING TO SHOW COVARIATE DATA LOADING FOR RACA ADD INAT DATA FOR SIMILAR SPECIES Downloading and assembling the covariate data takes some time but should be This step takes awhile depending on the size of the range.

```
if (sp.code == "RACA") {

  if (file.exists(paste0(data.dir, "covariates.rds"))) {
    covar <- read_rds(paste0(data.dir, "covariates.rds"))
  } else {

    # Note that elevation data must be downloaded before running get_elevation()
    # We have downloaded the elevation and waterbody data outside of this repo
    # See documentation for details.
    tri <- get_elevation(locs = region$sp.grid, path = "../species-futures/data/USA/",
                         id.label = "conus.grid.id")
    waterbody <- get_waterbodies(locs = region$sp.grid, path = "../species-futures/data/USA/",
                                 id.label = "conus.grid.id")

    footprint <- get_footprint(locs = region$sp.grid, id.label = "conus.grid.id")
    climate <- get_climate(locs = region$sp.grid, id.label = "conus.grid.id")
    traveltime <- get_traveltime(locs = region$sp.grid, id.label = "conus.grid.id")

    covar <- full_join(tri, footprint, by = "conus.grid.id") %>%
              full_join(climate, by = "conus.grid.id") %>%
              full_join(waterbody, by = "conus.grid.id") %>%
              full_join(traveltime, by = "conus.grid.id") %>%
              mutate(sqrtarea_small = sqrt(area_small),
                     sqrtarea_medium = sqrt(area_medium)) %>%
              select(conus.grid.id, sqrtarea_small, sqrtarea_medium,
                     footprint, TRI, tmin, traveltime)

    covar <- covar[order(match(covar$conus.grid.id, region$sp.grid$conus.grid.id)),]
    write_rds(covar, file = paste0(data.dir, "covariates.rds"))

  }

}
```

Next we remove grid cells which do not have complete covariates

```
## Remove incomplete cases
rm <- which(complete.cases(covar[,covs.z]) == F)
if (length(rm) > 0) {
  covar <- covar[-rm,]
  region$sp.grid <- region$sp.grid[-rm,]
}
```

Then we scale all covariates to have a mean of 1 and a standard deviation of 0.

```
## Scale covariates
covar_unscaled <- covar
```

```
numcols <- sapply(covar, is.numeric)
numcols <- which(numcols)
covar[,numcols] <- sapply(covar[,numcols], scale_this)
```

Finally we create the quadratic covariates.

```
if (length(covs.quad) > 0 & paste0(covs.quad, collapse = "") != "") {
  for (c in 1:length(covs.quad)) {
    covar[,paste0(covs.quad[c], "2")] <- covar[,covs.quad[c]] * covar[,covs.quad[c]]
    covs.z <- c(covs.z, paste0(covs.quad[c], "2"))
  }
}
```

There's one additional step that could be done here. If removing collinearity is desired, the following piece of code would check and remove covariates that violate the 0.4 correlation threshold. If there are fewer than 3 covariates remaining after the procedure, then message will alert you.

```
# remove covariates that are correlated > 0.4
if (check.covs == T){

  threshold <- 0.4
  if (sp.code == "PJOR") threshold <- 0.45

  covs.rm <- select_covar(covs.z, threshold = threshold)
  covs.lin <- covs.lin[-which(covs.lin %in% covs.rm)]
  covs.quad <- covs.quad[-which(covs.quad %in% covs.rm)]

  covs.z <- c(covs.lin, covs.quad)
}

if (length(covs.z) < 3) {
  stop("There are fewer than 3 covariates remaining! This probably isn't a good model")
}
```

Step 6: Plot covariates

```
if (block.out == "none") {

  ## Distribution covariates

  ## Define covariate labels
  covlabs <- read.csv("data/covariate-labels.csv") %>% filter(covariate %in% covs.z)

  plot_covar(covar,
             region,
             cov.names = covlabs$covariate,
             cov.labels = covlabs$Label,
             out.path = out.dir,
             out.name = "1_covariates-a_process-map")

  cor_covar(covar,
```

```
            cov.names = covlabs$covariate,
            cov.labels = covlabs$Label,
            out.path = out.dir,
            out.name = "1_covariates-a_process-correlations",
            color.threshold = 0.25)

  ## iNat covariates
  if ("iNaturalist" %in% names(species.data$obs)) {
    ## Define covariate labels
    covlabs <- read.csv("data/covariate-labels.csv") %>%
                filter(covariate %in% covs.inat)

    plot_covar(covar,
               region,
               cov.names = covlabs$covariate,
               cov.labels = covlabs$Label,
               out.path = out.dir,
               out.name = "1_covariates-b_iNat-map")

    if (length(covs.inat) > 1) {
      cor_covar(covar,
                cov.names = covlabs$covariate,
                cov.labels = covlabs$Label,
                out.path = out.dir,
                out.name = "1_covariates-b_iNat-correlations",
                color.threshold = 0.25)
    }
  }

  ## PO covs

  ## Define covariate labels
  covlabs <- read.csv("data/covariate-labels.csv") %>%
              filter(covariate %in% covs.PO)

  plot_covar(covar,
             region,
             cov.names = covlabs$covariate,
             cov.labels = covlabs$Label,
             out.path = out.dir,
             out.name = "1_covariates-c_PO-map")

  if (length(covs.PO) > 1) {
    cor_covar(covar,
              cov.names = covlabs$covariate,
              cov.labels = covlabs$Label,
              out.path = out.dir,
              out.name = "1_covariates-c_PO-correlations",
              color.threshold = 0.25)
  }
}
```

```
## Ignoring unknown labels:
```

```
## * colour : "Correlation coefficient"
```

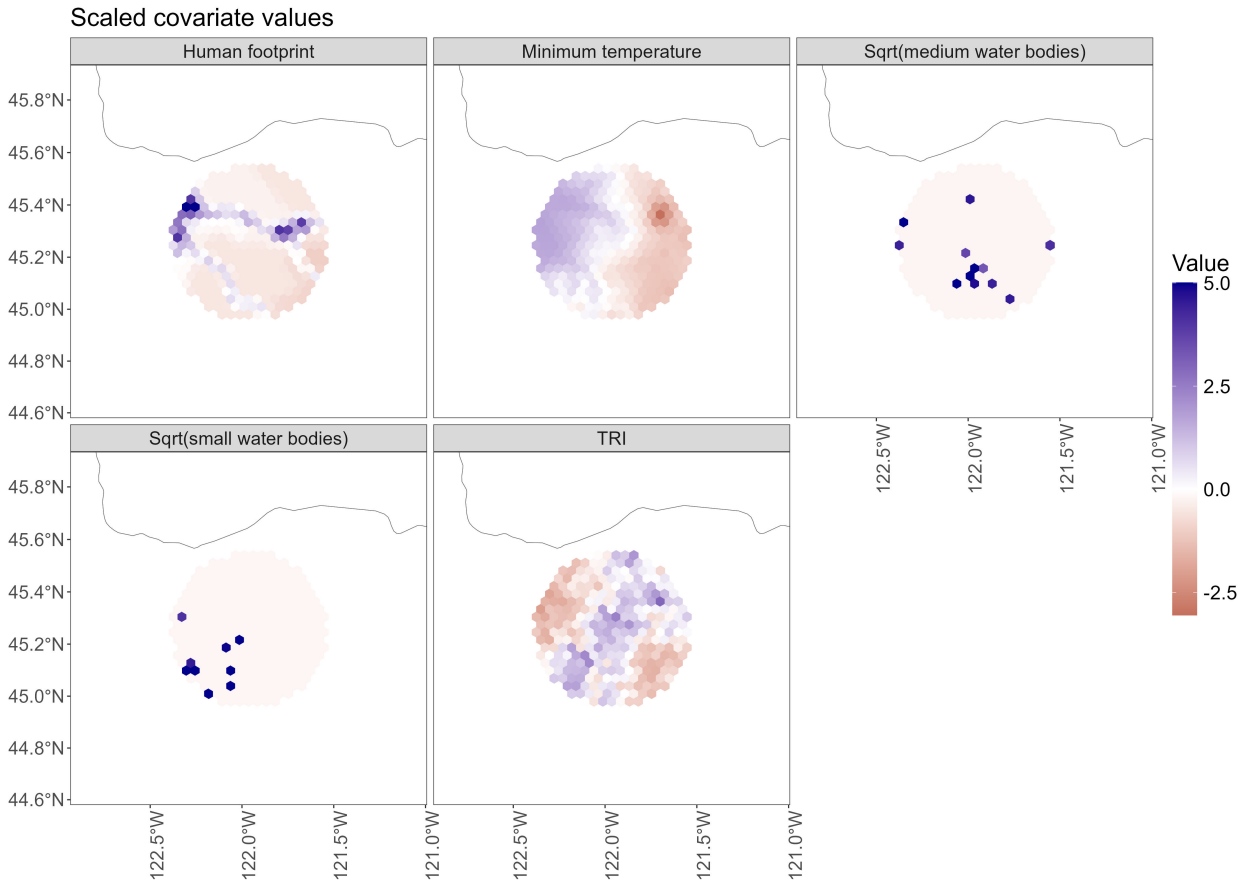

Figure 3: IMAGE

Step 7: NIMBLE

```
sp.data <- sppdata_for_nimble(species.data,
                              region,
                              file.info = allfiles,
                              covar = covar,
                              covs.inat = covs.inat,
```

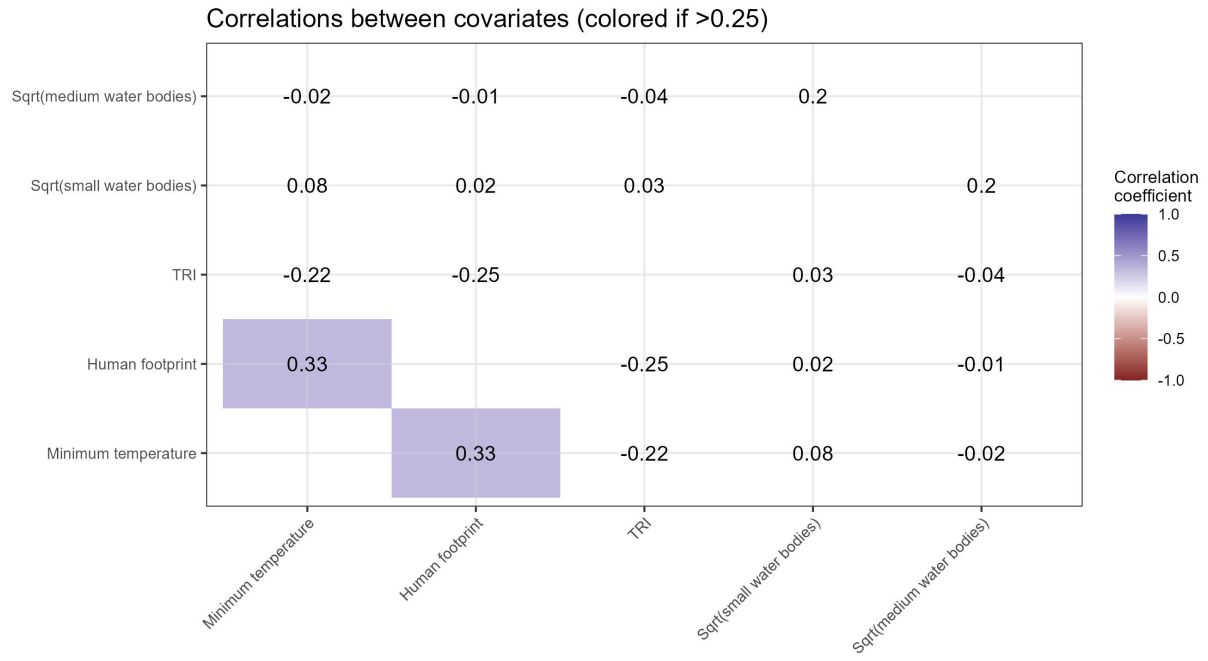

Figure 4: IMAGE

```
                                covs.PO = covs.PO,
                                DND.maybe = 1,
                                # Get rid of PO cells that are in the wrong grid cells
                                keep.conus.grid.id = gridkey$conus.grid.id[which(gridkey$group == "train"]
```

```
## No iNaturalist data for this species
##
## Loading PO
## Loading PO
## Loading DND
## Loading count
## Loading count
```

Data/constants

```
tmp <- data_for_nimble(sp.data, covar = covar, covs.z,
                       sp.auto = sp.auto, coarse.grid = coarse.grid, region = region,
                       process.intercept = F,
                       gridkey = gridkey, spatRegion = spatRegion)

data <- tmp$data
constants <- tmp$constants
```

STATE INDICATOR FOR GPOR

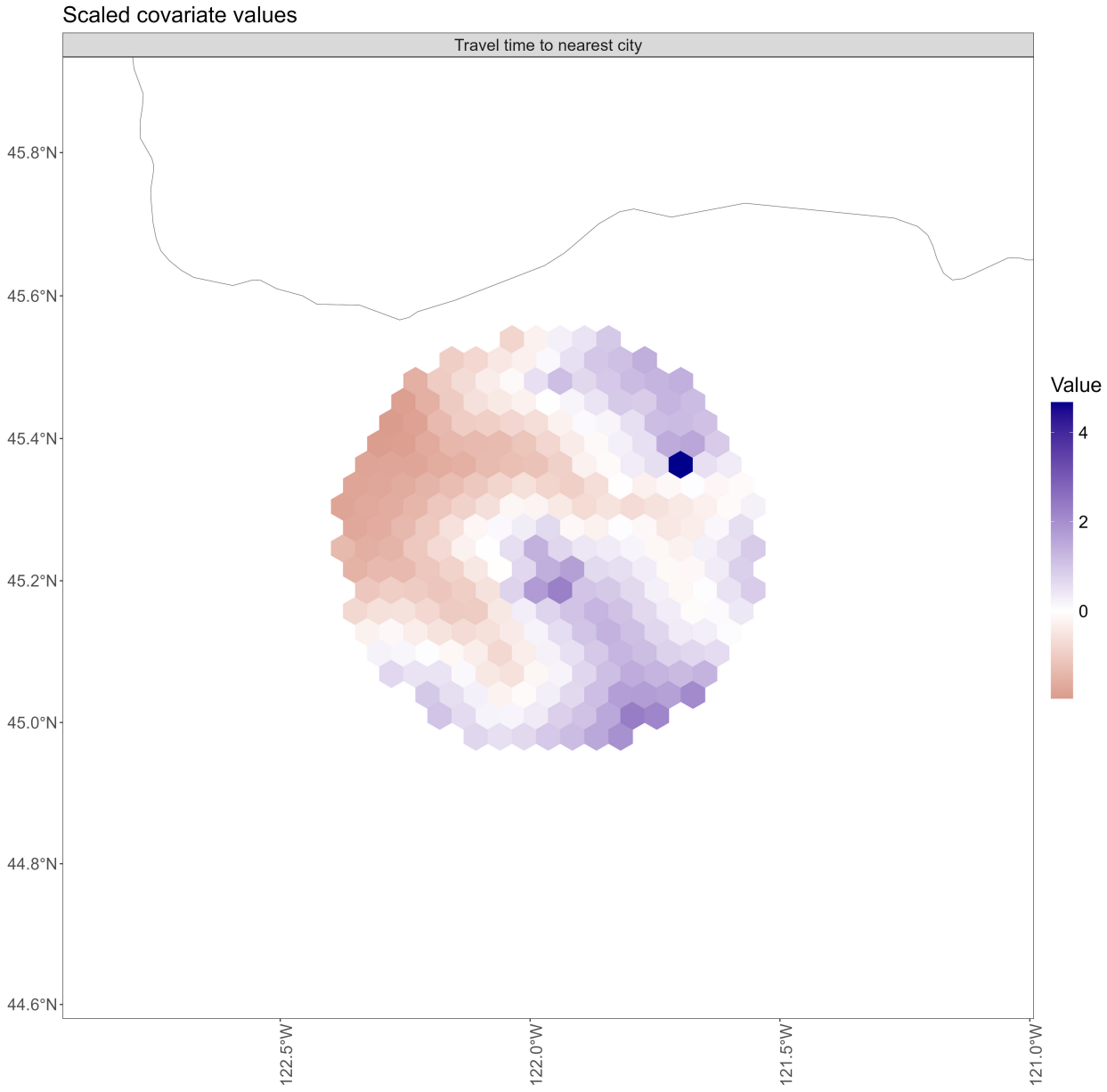

Figure 5: IMAGE

```
# Add state indicator variable for iNat data to indicate which states have taxon geoprivacy
# Add state indicator for multi-state PO to indicate which states have data
if (sp.code == "GPOR") obsc.state <- c("CT", "MS", "NJ", "RI")
constants <- add_state_ind(species.data,
                           region,
                           gridkey,
                           constants,
                           covs.inat = covs.inat,
                           obsc.state = obsc.state,
                           keep.conus.grid.id = gridkey$conus.grid.id[which(gridkey$group == "train")])
```

Code

```
code <- nimble_code(data,
                    constants,
                    path = out.dir,
                    sp.auto = sp.auto,
                    coarse.grid = coarse.grid,
                    Bprior = Bprior,
                    block.out = block.out,
                    min.visits.incl = 3,
                    zero_mean = zero_mean,
                    rm.state = F,
                    tau = tau)
```

```
##
## # Process Model
##
## # 279 cells
## for (i in 1:nCell) {
##
##     # log intensity
##     XB0[i] <- inprod(B[1:nCovZ], Xz[i,1:nCovZ])
##     log(lambda0[i]) <- XB0[i] + spat[i] # residual spatial effect
##
## }
##
##
## # ---------------------------------------
##
## # Observation Model 1: PO, FS NRIS
## # 279 cells
## for (j in 1:nW1) {
##
##   # Make dataset-specific lambda (and N and Z if needed)
##   lambdaD1[j] <- lambda0[Wcells1[j]] * alpha[1]
##
##   # Observation model
##   W1[j] ~ dpois(lambdaD1[j] * E1[j]) # Poisson
##
##   # Effort
```

```
##   log(E1[j]) <- A1[1] * Xw1[j,1]
##
##   # Prior for X imputation
##   Xw1[j, 1] ~ dnorm(0, 1)
## }
##
##
##
## # Observation Model 2: PO, BLM Surveys, ORBIC
## # 279 cells
## for (j in 1:nW2) {
##
##   # Make dataset-specific lambda (and N and Z if needed)
##   lambdaD2[j] <- lambda0[Wcells2[j]] * alpha[2]
##
##   # Observation model
##   W2[j] ~ dpois(lambdaD2[j] * E2[j]) # Poisson
##
##   # Effort
##   log(E2[j]) <- A2[1] * Xw2[j,1]
##
##   # Prior for X imputation
##   Xw2[j, 1] ~ dnorm(0, 1)
## }
##
##
##
## # Observation Model 3: DND, USGS RACA assessment - historic
## # 24 observations, 2.5 median visits per site
## for (j in 1:nV3) {
##
##   # Make dataset-specific lambda (and N and Z if needed)
##   lambdaD3[j] <- lambda0[Vcells3[j]] * alpha[3]
##
##   # Observation model
##   V3[j] ~ dbern(1-exp(-lambdaD3[j] * p3[j])) # Bernoulli
##
##   # Detection
##   log(p3[j]) <- inprod(D3[1:nCovV3], Xv3[j,1:nCovV3])
##
##   # Prior for X imputation
##   for (c in 1:nCovV3) {
##     Xv3[j,c] ~ dnorm(0, 1)
##   }
## }
##
##
##
## # Observation Model 4: count, USGS RACA assessment - current
## # 16 observations, 2 median visits per site
## for (j in 1:nY4) {
##
##   # Make dataset-specific lambda (and N and Z if needed)
##   lambdaD4[j] <- lambda0[Ycells4[j]] * alpha[4]
```

```
##
##   # Observation model
##   Y4[j] ~ dpois(lambdaD4[j] * p4[j]) # Poisson
##
##   # Detection
##   log(p4[j]) <- C4[1] * Xy4[j,1]
##
##   # Prior for X imputation
##   Xy4[j, 1] ~ dnorm(0, 1)
## }
##
##
##
## # Observation Model 5: count, PNW Salamanders
## # 19 observations, 1 median visits per site
## for (j in 1:nY5) {
##
##   # Make dataset-specific lambda (and N and Z if needed)
##   lambdaD5[j] <- lambda0[Ycells5[j]] * alpha[5]
##
##   # Observation model
##   Y5[j] ~ dpois(lambdaD5[j] * p5[j]) # Poisson
##
##   # Detection
##   log(p5[j]) <- C5[1] * Xy5[j,1]
##
##   # Prior for X imputation
##   Xy5[j, 1] ~ dnorm(0, 1)
## }
##
##
## # --------------------------------------
## # Process priors
## for (a in 1:nCovZ) {
##   B[a] ~ dnorm(0,1)
## }
##
## # Dataset intercepts
## for (a in 1:nD) {
##   alpha[a] <- exp(w[a])
##   w[a] ~ dnorm(0,1)
## }
##
## # Observation priors, PO 1: FS NRIS
## for (b in 1:nCovW1) {
##   A1[b] ~ dnorm(0,1)
## }
##
## # Observation priors, PO 2: BLM Surveys, ORBIC
## for (b in 1:nCovW2) {
##   A2[b] ~ dnorm(0,1)
## }
##
## # Observation priors, DND 3: USGS RACA assessment - historic
```

```
## for (b in 1:nCovV3) {
##   D3[b] ~ dnorm(0,1)
## }
##
## # Observation priors, count 4: USGS RACA assessment - current
## for (b in 1:nCovY4) {
##   C4[b] ~ dnorm(0,1)
## }
##
## # Observation priors, count 5: PNW Salamanders
## for (b in 1:nCovY5) {
##   C5[b] ~ dnorm(0,1)
## }
##
## tau <- 1
## spat[1:nCell] ~ dcar_normal(adj[1:L], weights[1:L], num[1:nCell], tau, zero_mean = 1)
```

Initial values

```
inits <- function(x){nimble_inits(data,
                                  constants,
                                  sp.auto = sp.auto,
                                  seed = x)}
```

Parameters

```
params <- nimble_params(data,
                        constants,
                        lambda = T,
                        XB = T,
                        sp.auto = sp.auto,
                        effort = T)
```

Step 8: Clean up and save

```
# Remove local and block in case the setup is run locally but the model is fit on the HPC.
# Remove other unnecessary files to reduce the size of setup_BLOCK.Rdata
rm(list=c('local','block','args','conus.covar.grid','conus.grid','usa','conus',
          'pl','centroid'))


# Save environment and full set up
save.image(paste0(out.dir, "setup_",block.out,".Rdata"))
```

02-flexiSDM.R

```
num <- 3
block <- "none"
sp.code <- "RACA"
model <- "subrange"
chain <- 1
local <- 1
```

```
# Load functions and packages
source("functions/FXN-nimbleParallel.R") # HOW DO YOU RUN THIS IN ISDM-FRAMEWORK?
library(SpFut.flexiSDM)
```

```
# Set output directory and load setup file
out.dir = paste0('outputs/',num,'_',sp.code,'_',model,'/')

load(paste0(out.dir,'setup_',block,'.Rdata'))
```

Fit NIMBLE model

```
if (local == 1) {
  start.nim <- Sys.time()
  samples <- nimbleParallel(code = code,
                            data = data,
                            constants = constants,
                            inits = inits,
                            param = params,
                            iter = iter,
                            burnin = burnin,
                            thin = thin)

  end.nim <- Sys.time() - start.nim
  print(end.nim)

  saveRDS(samples, paste0(out.dir,'samples_',block,'.rds'))
} else {

  info <- list(seed = chain,
               inits = inits(chain))

  start.nim <- Sys.time()
  samples <- run_nimbleMCMC(info = info,
                            code = code,
                            constants = constants,
                            data = data,
                            param = params,
                            iter = iter,
                            burnin = burnin,
                            thin = thin)
```

```
  end.nim <- Sys.time() - start.nim
  print(end.nim)

  saveRDS(samples, paste0(out.dir,'samples_',block,'_',chain,'.rds'))
}
```

–> –> –> –> –> –> –> –> –> –> –> –> –>

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/')
  }
}
```