Appendix 2

Last updated: 2026-02-12

Contents

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

NEED SOME SORT OF INTRO HERE.

Our workflow is separated into four parts.

- First, the data needs to be in the correct format. We do not provide a script for this, as each dataset begins in a unique format. Instead, we provide a description of the format that the analysis requires.
- The first script sets up the range, imports species and covariate data, scales covariates, and sets everything up for NIMBLE.
- The second script fits the NIMBLE model.
- The third script 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)
library(rnaturalearth)

## 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)
# remotes::install_git('https://code.usgs.gov/eastern-ecological-science-center/nearmi/SpFut-covariates
library(SpFut.covariates)

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

# External path for large covariate datasets
ext.path <- '~/GitHub/species-futures/data/USA/'
```

Part 0: Data formatting

Data must be in the following structure to enter models. This structure is suitable for Presence-only (PO), Detection/non-detection (DND), and count (Count) data. Other data types can usually be reduced into one of these types (e.g., capture-mark-recapture can be reduced to Count or DND). In anticipation of adding a specific observation model for CMR and time-to-detection data in the future, we include `individual.id` and `time.to.detection` as required columns in the data format.

We use a long format, where there is a unique row for each age (if recorded) of each species in each pass of each sampling event (defined as a visit to a site). There are 13 required columns. Any other information recorded in the field that relates to detection probability (e.g., duration of survey, area surveyed) should be included as additional columns. There should be no empty cells. Use NA (not NULL, -999, etc.) to indicate missing data.

Column Name	Format	Appropriate values	Notes
site.id	character	alphanuermic string	Each site.id is associated with exactly one set of coorinates
lat	numeric	WGS84	Each set of coordinates is associated with exactly one site.id
lon	numeric	WGS84	Each set of coordinates is associated with exactly one site.id
day	numeric	1-31	If day was not recorded, use NA
month	numeric	1-12	If month was not recorded, use NA
year	numeric	4 digit year	If year was not recorded, use NA
survey.conducted	numeric	0, 1	Indicates whether or not a survey was conducted (e.g., if there was no water at an intended survey location, survey.conducted = 0). For all PO records, survey.conducted = 1
survey.id	numeric	any number	All observations from a single visit to a single site have the same survey.id
data.type	character	`PO`, `DND`, `count`	data.type may vary across species or age class (e.g. adults are count but larvae are DND)
pass.id	numeric	any number	Within a survey.id (i.e., a visit to a site), each pass should have a unique pass.id. All observations within a pass should have the same pass.id
species	character	species identifier	We use 4- or 6-digit codes. Any value can be used, as long as it is consistent across all datasets
age	character	`egg`, `egg mass`, `larva`, `juvenile`, `metamorph`, `adult`, `unknown`, `NR`	Use NR if age classes were not recorded. Do not use NA
individual.id	character	alphanumeric string	If no individuals were seen (count = 0), then individual.id = NA. If individuals were seen (count = 1) but not marked (too small, escaped, etc.), then individual.id = 0.
time.to.detect	numeric	any number	Use NA if time to detection was not recorded

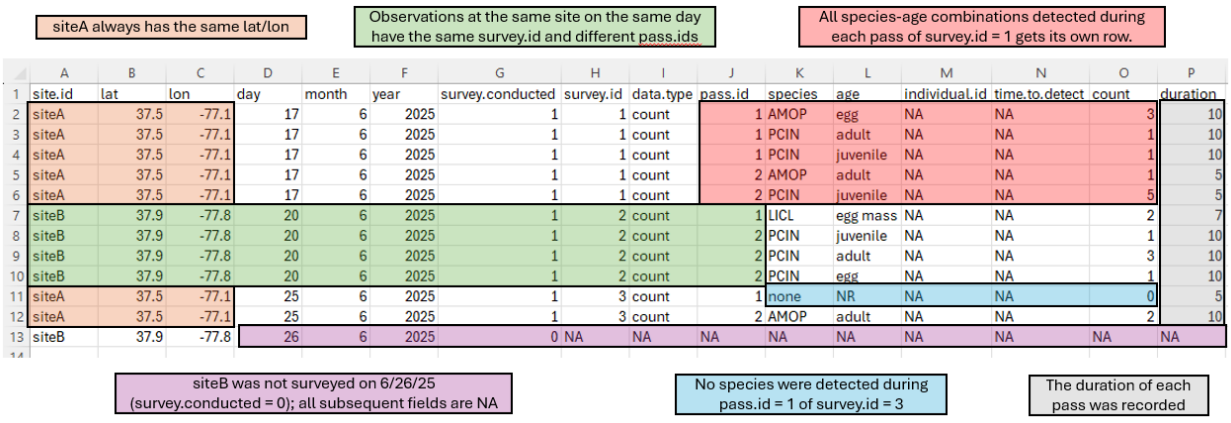
Column Name	Format	Appropriate values	Notes
count	numeric	If count: 0-Inf; If DND or CMR: 0/1/2; If PO: 1	Must be a specific value, not range of numbers (e.g., 10-50); for DND and CMR, 0 = not detected, 1 = detected, 2 = maybe detected

A Note on survey.id and pass.id: To make sure all data enter the model correctly, we need to know which records were part of the same sampling event. A sampling event is a specific visit to a specific site. Sampling events are identified with `survey.id`. During a sampling event, multiple observations ("passes") may be made, e.g., multiple observers, multiple passes by a single observer, multiple audio recordings, multiple water samples for eDNA. Generally, site.id, day, month, and year define a unique sampling event. The exception is if passes occur overnight: passes that occur after midnight should be given the same survey.id as passes that occur before midnight.

All passes within a sampling event get the same survey.id. All passes within a sampling event get a unique pass.id.

Example: a double observer visual encounter survey. Both observations at Site A on Day 1 get survey.id = 1, and the observations by each observer get a pass.id that is unique within the survey.id (1 and 2). All observations at site B on Day 1 get survey.id = 2, and each observation gets a unique pass.id (1 and 2). Site A is revisited on Day 2, and all observations get survey.id = 3, and each observation gets a unique pass.id (1 and 2).

Example: five audio recordings are taken at each visit to a site. All five recordings at Site A on Day 1 get survey.id = 1, and each recording gets a pass.id from 1 to 5. All recordings at Site B on Day 1 get survey.id = 2, and each recording gets a pass.id from 1 to 5. Site A is revisited on Day 2, and all recordings get survey.id = 3, and each recording gets a pass.id from 1 to 5.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	site.id	lat	lon	day	month	year	survey.conducted	survey.id	data.type	pass.id	species	age	individual.id	time.to.detect	count	duration
2	siteA	37.5	-77.1	17	6	2025	1	1	count	1	AMOP	egg	NA	NA	3	10
3	siteA	37.5	-77.1	17	6	2025	1	1	count	1	PCIN	adult	NA	NA	1	10
4	siteA	37.5	-77.1	17	6	2025	1	1	count	1	PCIN	juvenile	NA	NA	1	10
5	siteA	37.5	-77.1	17	6	2025	1	1	count	2	AMOP	adult	NA	NA	1	5
6	siteA	37.5	-77.1	17	6	2025	1	1	count	2	PCIN	juvenile	NA	NA	5	5
7	siteB	37.9	-77.8	20	6	2025	1	2	count	1	LICL	egg mass	NA	NA	2	7
8	siteB	37.9	-77.8	20	6	2025	1	2	count	2	PCIN	juvenile	NA	NA	1	10
9	siteB	37.9	-77.8	20	6	2025	1	2	count	2	PCIN	adult	NA	NA	3	10
10	siteB	37.9	-77.8	20	6	2025	1	2	count	2	PCIN	egg	NA	NA	1	10
11	siteA	37.5	-77.1	25	6	2025	1	3	count	1	none	NR	NA	NA	0	5
12	siteA	37.5	-77.1	25	6	2025	1	3	count	2	AMOP	adult	NA	NA	2	10
13	siteB	37.9	-77.8	26	6	2025	0	NA	NA	NA	NA	NA	NA	NA	NA	NA

Figure 1: Example of data structure

Part 1: 01-flexiSDM.R

Load and define model specifications

Each model requires a variety of specifications. Rather than enter them manually in the code, we find it easier to manage multiple models by storing all specifications in a .csv file that is read in to provide input

values. We call our csv file `model-specs.csv`. Each row represents a model, so you can easily see and edit the parameters used for each model. 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 region.sub buffer cont.grid coarse.grid lon.lo lon.hi
## 1      1    RACA     iSDM      FALSE  50000     FALSE       FALSE     NA     NA
## 2      2    GPOR     iSDM      FALSE  50000     FALSE        TRUE     NA     NA
## 3      3    RACA subrange       TRUE  50000     FALSE       FALSE     NA     NA
## 4      4    GPOR subrange       TRUE 100000     FALSE       FALSE     NA     NA
##   lat.lo lat.hi year.start year.end filter.region spat.thin coordunc
## 1     NA     NA       1994     2025          TRUE      TRUE     1000
## 2     NA     NA       1994     2025          TRUE      TRUE     1000
## 3     NA     NA       1994     2025          TRUE      TRUE     1000
## 4     NA     NA       1994     2025          TRUE      TRUE     1000
##   coordunc_na.rm    covs.PO  covs.inat
## 1           TRUE traveltime       <NA>
## 2           TRUE traveltime        ORM
## 3           TRUE traveltime       <NA>
## 4           TRUE traveltime traveltime
##                                  covs.lin covs.quad check.covs     Bprior
## 1 sqrtarea_medium, sqrtarea_small, footprint TRI, tmin      FALSE dnorm(0,1)
## 2              streamLength.km, prec, forest elevation      FALSE dnorm(0,1)
## 3 sqrtarea_medium, sqrtarea_small, footprint TRI, tmin      FALSE dnorm(0,1)
## 4              streamLength.km, prec, forest elevation      FALSE dnorm(0,1)
##   sp.auto zero_mean tau   iter thin project block.rows block.cols block.folds
## 1    TRUE      TRUE   1 100000    5       0          5          5           3
## 2    TRUE      TRUE   1 100000    5       0          5          5           3
## 3    TRUE      TRUE   1  10000    5       0          5          5           3
## 4    TRUE      TRUE   1  10000    5       0          5          5           3
##   mem.nim mem.sum
## 1   30000   40000
## 2   30000   60000
## 3   30000   40000
## 4   30000   40000
```

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	Recommended value	Description
number	numeric	1	Model number
sp.code	character	species identifier	Species code, must match the species code used in the species data
model	character	short descriptor	Model name
region.sub	logical	F, unless troubleshooting	Only use the centroid of the region with a radius equal to the buffer size?
buffer	numeric	50000	Size of buffer around range edge or centroid (in m)

Column Name	Type	Recommended value	Description
cont.grid	logical	T	Should the grid be restricted to continuous cells? (This will remove any groups of non-contiguous cells)
coarse.grid	logical	T, unless small region	Should a coarse grid be used for the spatial model?
lon.lo	numeric	empty, unless troubleshooting	Low longitude value to restrict the range
lon.hi	numeric	empty, unless troubleshooting	High longitude value to restrict the range
lat.lo	numeric	empty, unless troubleshooting	Low latitude value to restrict the range
lat.hi	numeric	empty, unless troubleshooting	High latitude value to restrict the range
year.start	numeric	1994	Starting year of data to include
year.end	numeric	current year	Ending year of data to include
filter.region	logical	F to see if species is recorded outside of range	Should data outside of the region be removed? Note that this is only for mapping purposes; data that are outside of the region are never used to fit the model
spat.thin	logical	T	Should the PO data be spatially thinned?
coordunc	numeric	1000	The level of acceptable coordinate uncertainty (meters, for PO datasets that record coordinate uncertainty)
coordunc_na.rm	logical	T	Should PO data with coordinate uncertainty = NA be removed? (for PO datasets that record coordinate uncertainty)
covs.PO	character	human density	A comma-separated list of covariates for modeling non-iNaturalist PO sampling effort. These must match the covariate names in the data.
covs.inat	character	human density	A comma-separated list of covariates for modeling iNaturalist sampling effort. These must match the covariate names in the data.
covs.lin	character	based on species ecology	A comma-separated list of linear covariates for modeling species relative abundance. These must match the covariate names in the data.
covs.quad	character	based on species ecology	A comma-separated list of quadratic covariates for modeling species relative abundance. These must match the covariate names in the data.
check.covs	logical	T	Should covariate selection remove multicollinearity?
Bprior	numeric or character	dnorm(0, 1)	Assign a prior distribution in BUGS language for the β coefficients in the distribution model (e.g., dnorm(1,0))
sp.auto	logical	T	Should a spatial model be included?
zero_mean	logical	T	Should a zero mean assumption be included in the spatial model?
tau	numeric or character	1	Assign a fixed value or a prior distribution in BUGS language for precision (τ; e.g., dgamma(5,5))

Column Name	Type	Recommended value	Description
iter	numeric	depends on computational power	The number of iterations to run an MCMC chain (the first 75% will be discarded as a burn-in)
thin	numeric	5	Thin the MCMC chains by this parameter
project	numeric	0	The number of future projections (otherwise, NA)
block.rows	numeric	5	The number of rows for cross-validation blocks
block.cols	numeric	5	The number of columns for cross-validation blocks
block.folds	numeric	3	The number of folds to group cross-validation blocks into
mem.nim	numeric	30000	Memory allocation (in MB) for NIMBLE. Can be omitted if not using an HPC.
mem.sum	numeric	40000	Memory allocation (in MB) for parallelized summarization. Can be omitted if not using an HPC.

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
fold <- 'none' # CV fold to exclude ('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 model 3, which uses the centroid of the Cascades frog (RACA) range with a 50km buffer. This will allow us to easily visualize the datasets and spatial grid, while maintaining local computational capabilities. You can see all of the specifications for this model in the third row of `mods`. We have provided an additional example, model 4, which uses the centroid of the Spring salamander (GPOR) range with a 100km buffer.

Now that we have loaded our model specifications, we can use them to define a series of inputs for the setup script.

```
## 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] # species code
model <- mods$model[1] # model name
sp.auto <- mods$sp.auto[1] # include spatial autocorrelation?
coarse.grid <- mods$coarse.grid[1] # use a coarse grid for the spatial effect?
cont.grid <- mods$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$buffer[1] # buffer size (m)
filter.region <- mods$filter.region[1] # filter data to the region?
spat.thin <- mods$spat.thin[1] # include spatial thinning for PO data?
coordunc <- mods$coordunc[1] # level of coordinate uncertainty to include (m)
coordunc_na.rm <- mods$coordunc_na.rm[1] # include coordinate uncertainty = NA?
block.folds <- mods$block.folds[1] # how many groups of blocks?
```

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

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

## 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$region.sub[1] # only use the centroid + buffer?
lat.hi <- mods$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 (fold.out != "none") {
  project <- 0 # no projections for cross-validation models
}

## Covariates
# non-iNaturalist PO covariates
covs.PO <- unlist(str_split(mods$covs.PO[1], pattern = ", "))
# iNaturalist covariates
covs.inat <- unlist(str_split(mods$covs.inat[1], pattern = ", "))
# linear covariates for distribution model
covs.lin <- unlist(str_split(mods$covs.lin[1], pattern = ", "))
# quadratic covariates for distribution model
covs.quad <- unlist(str_split(mods$covs.quad[1], pattern = ", "))
check.covs <- mods$check.covs[1] # covariate selection to remove multicollinearity? (T/F)

## Define the prior for the distribution 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 == "")]
}
if (NA %in% covs.z) {
  covs.z <- covs.z[-which(is.na(covs.z))]
}
```

We're almost ready to set up the region and load data. Let's first ensure that we have output and data folders specific to the model we are fitting to store some of these objects.

```
## 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 (`conus.grid`). Unfortunately, this file is too large to upload to GitHub, so we demonstrate how we created the region and provide the file in `data.dir/region.rds` to bypass this step.

```
## 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)
  # We use GAP and IUCN ranges, but you can use any polygons. Just enter the path to where you store th

  ## USA boundary
  exclude <- c("Alaska", "Hawaii", "Commonwealth of the Northern Mariana Islands",
               "American Samoa", "United States Virgin Islands", "Guam", "Puerto Rico")
  usa <- st_read(paste0(ext.path,"maps/cb_2018_us_state_500k/cb_2018_us_state_500k.shp")) %>%
          filter((NAME %in% exclude) == F) %>%
          st_union()
  ## CONUS grid
  load(paste0(ext.path,"grid-and-huc.rdata"))

  ## Region
  region <- make_region(rangelist,
                        buffer = buffer,
                        sub = region.sub,
                        boundary = usa, # crop study region to this boundary
                        grid = conus.grid, # You can use any grid to delineate spatial units
                        rm.clumps = T, # remove small clumps of cells?
```

```
                          clump.size = 50, # size of clumps to remove, if rm.clumps = T
                          continuous = cont.grid)

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

There is an additional step here if you intend to use a coarse spatial grid for the spatial model. Using a coarse grid greatly reduces computation time, which may be of concern for large-range species. We must redefine the spatial grid by grouping neighboring 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 (fold.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
}
```

Finally, identify which cells in the grid fall into which states. Add a column to `region$sp.grid` indicating whether each cell falls into one or more states. These will be used later.

```
statemap <- ne_states(country = c("Canada", "Mexico", "United States of America"),
                      returnclass = "sf")
stategrid <- get_state_grid(region, statemap)

xstate <- stategrid %>%
          group_by(conus.grid.id) %>%
          summarize(nstate = n()) %>%
          filter(nstate > 1) %>%
          pull(conus.grid.id)
region$sp.grid <- region$sp.grid %>%
  mutate(nstate = case_when(conus.grid.id %in% xstate ~ "multi",
                            T ~ "single"))
```

Step 2: Designate training and testing data

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

```
spatblocks <- make_CV_blocks(region, rows = block.rows, cols = block.cols,
                             k = block.folds)

if (fold.out == "none") {
  # If fitting the full model, then there are no test data
```

```
  test.i <- c()
  train.i <- region$sp.grid$conus.grid.id

} else {

  ## Set up cross validation blocks
  block1 <- spatblocks %>% filter(folds == fold.out)

  ## Find grid.ids for test fold, everything else is in the training fold
  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, the cells are assigned to "train" and "test" groups in a `gridkey`. This `gridkey` also contains `grid.id`, which allows us to index the data in NIMBLE.

```
# 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 do this with our list, but these values could be set manually or pulled from a different list. We additionally define the species type (e.g., Frog/Toad or Salamander) based off the genus name. This delineation is used later in the covariate data section.

```
## 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
spp.type <- codeKey %>%
  filter(DS.code == sp.code) %>%
  pull(spp.type)
spp.type
```

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

Next we will create a dataframe containing information about each data source. We define ours in a CSV file (`00-data-summary-flexiSDM.csv`) because we have hundreds of datasets to filter through for different species. This process could also be done by manually creating a dataframe with the same column names. The necessary columns are:

`-file.name` (how the file is named in the source folder) `-file.label` (the label for the data source) `-data.type` (Count, DND, or PO) `-PO.extent` (does the presence-only span the continent or a particular state; only required for PO) `-covar.mean` (detection covariates that should be averaged across passes) `-covar.sum` (detection covariates that should be summed across passes)

For additional clarification, sometimes it is more appropriate to summarize detection covariates by averaging than summing and vice versa. For example, the number of survey minutes should be summed across passes but the temperature during the survey should be averaged.

```
# 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
```

We can now use `allfiles` to load all of the species data sources. We have developed the function `load_species_data()` which takes the species code(s), `allfiles`, the region, and information about filtering, start/end dates, and spatial uncertainty to load and filter the data appropriately. The number of observations removed from each data source and the reasons for removal are provided as console output to the user.

```
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.thin,
                  keep.conus.grid.id = gridkey$conus.grid.id[which(gridkey$group == "train")])
```

```
##
## File 1: ARMI Count....Loading file
## Starting with 4500 observations of RACA
## Removing 4000 observations that are outside of the study region
## No data for this analysis
##
## File 2: BLM Surveys....Loading file
## Starting with 883 observations of RACA
## 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
## Starting with 1753 observations of RACA
## 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
## Starting with 16 observations of RACA
## Removing 16 observations that are outside of the study region
## No data for this analysis
##
## File 5: NCCN....Loading file
## Starting with 52680 observations of RACA
## Removing 52680 observations that are outside of the study region
## No data for this analysis
```

```
##
## File 6: FS NRIS....Loading file
## Starting with 54795 observations of RACA
## 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
## Starting with 116 observations of RACA
## 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
## Starting with 495 observations of RACA
## Removing 390 observations that are outside of the study region
## Using  effort  as covariate(s)
##
## File 9: USGS RACA assessment - historic....Loading file
## Starting with 115 observations of RACA
## Removing 91 observations that are outside of the study region
##
## File 10: PNW Salamanders....Loading file
## Starting with 291 observations of RACA
## Removing 177 observations that are outside of the study region
## Using  distance  as covariate(s)
##
## File 11: WDFW....Loading file
## Starting with 1195 observations of RACA
## Removing 1195 observations that are outside of the study region
## No data for this analysis
```

Step 4: Plot species data

Next let's plot the species data to visualize where the data occur in the subrange.

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

## Plot data samples
out <- 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))
```

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

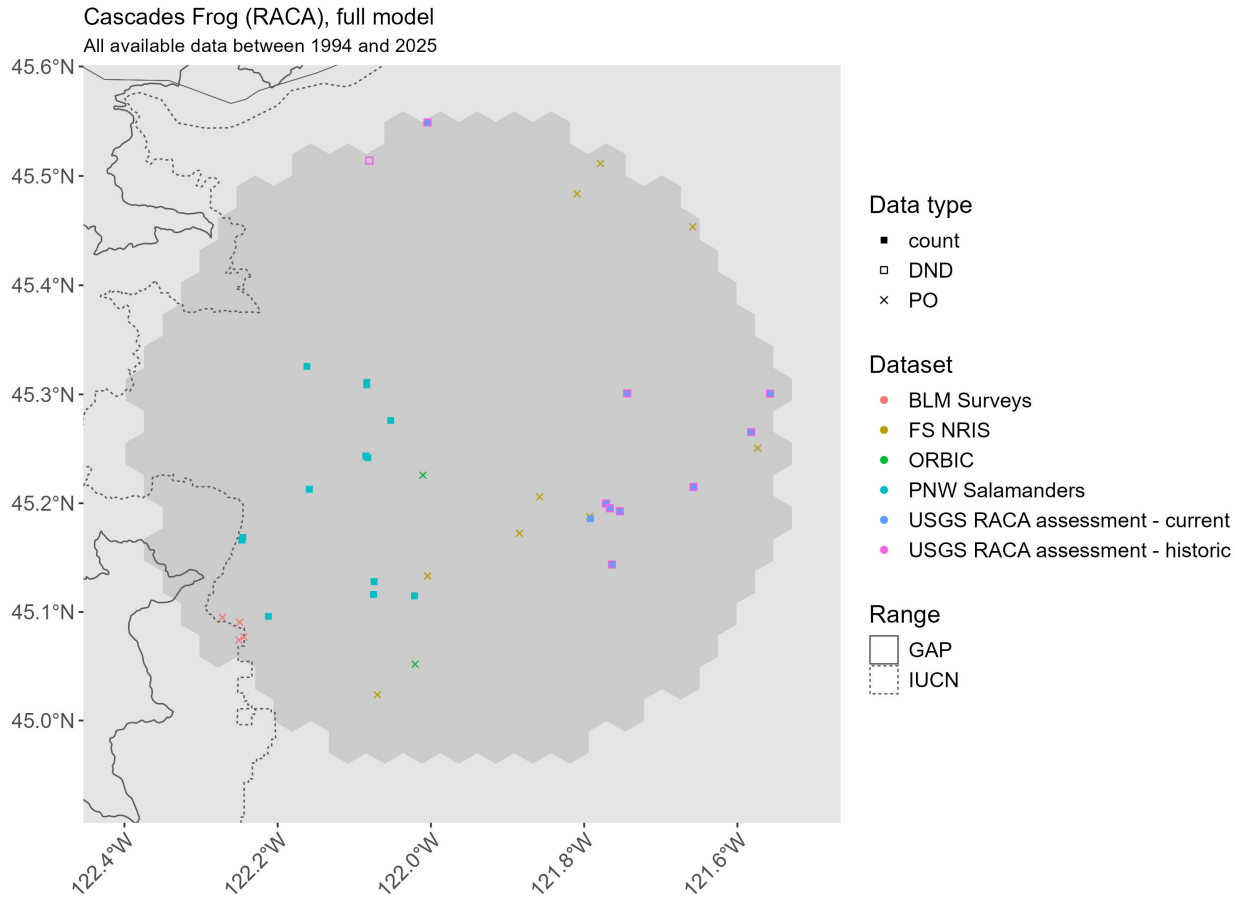


Figure 2: Distribution of species data across the subrange

We can also take a look at the distribution of data across the cross-validation folds.

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

  out <- map_species_data(region = region,
                          species.data = species.data,
                          year.start = year.start,
                          year.end = year.end,
                          plot = "samples",
                          blocks = spatblocks[which(spatblocks$folds == fold),],
                          plot.region = T,
                          details = F,
                          title = paste0(common, " (", sp.code, ")", title))
```

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

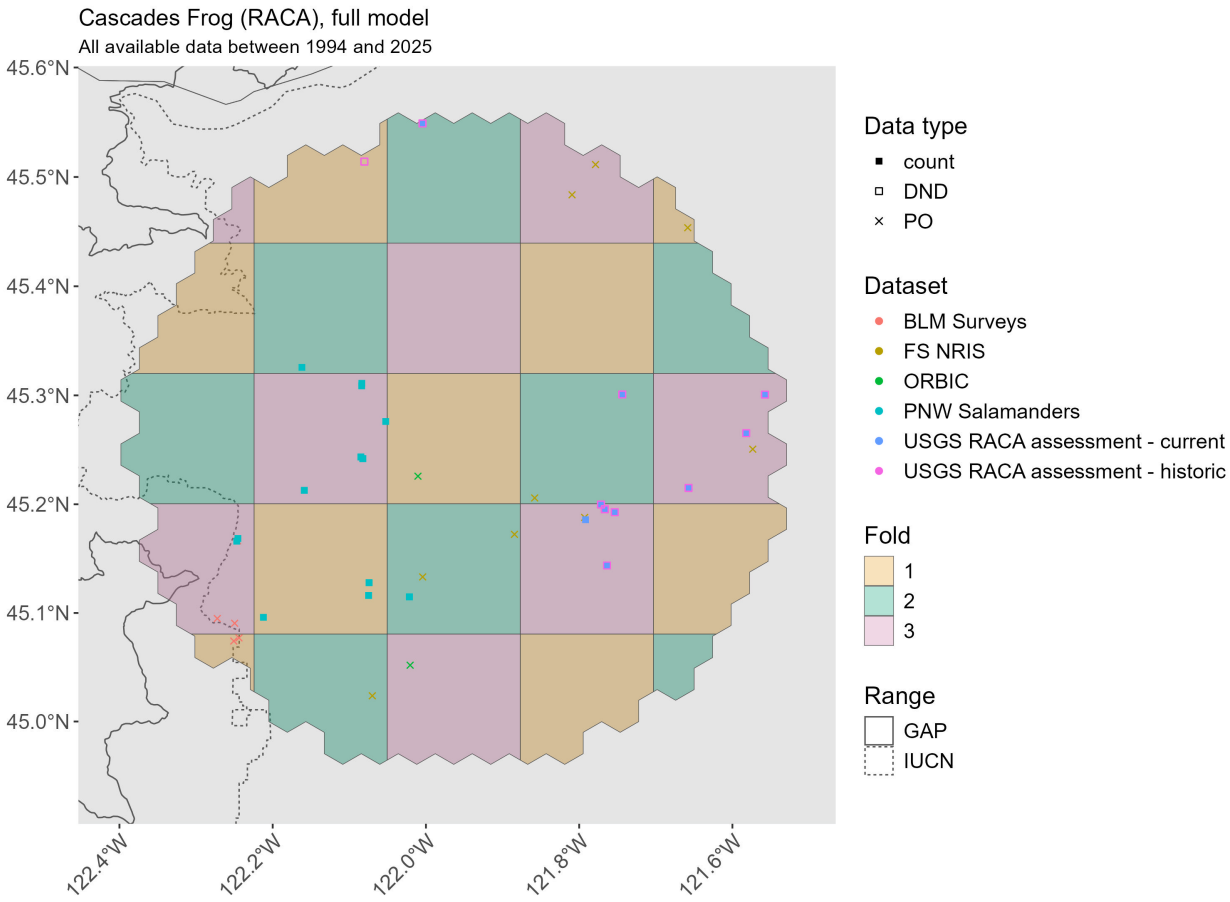


Figure 3: Distribution of data with overlaid cross-validation blocks

Step 5: Load covariate data

We're now ready to set up the covariate data. The dataframe `covar` needs to contain all covariates needed for the distribution model (`covs.z`), as well as all covariates needed for the PO effort models (`covs.PO` and `covs.inat`). These covariates vary across species. Any method can be used to derive the covariates, as long as it is aggregated to the spatial unit and contains a column for `conus.grid.id`.

We download and process covariates using functions from the `SpFut.covariates` package (ISSUE - cite). These functions consist of wrappers for other data sources (e.g., National Hydrography Database and the `geodata` R package (ISSUE - cite)) that download and format the data for our purposes. See the documentation for the `SpFut.covariates` package for more information on how to use the functions properly.

Note: Downloading and assembling the covariate data can take significant time depending on the size of the range and your internet connection. We demonstrate how we downloaded the data below but provide the `covariates.rds` file to proceed to the next step.

```
# ISSUE - re-download covariates with CRAN package
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$sp.grid,
                         path = "~/GitHub/species-futures/data/USA/",
                         id.label = "conus.grid.id")
    waterbody <- get_waterbodies(locs = region$sp.grid,
                                 path = "~/GitHub/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"))

  }

}


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

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

    # Note that elevation and flowlines data must be downloaded before running
    # get_elevation() and get_flowlines(). See documentation for details.
    tri <- get_elevation(locs = region$sp.grid,
                         path = "~/GitHub/species-futures/data/USA/",
                         id.label = "conus.grid.id")
    stream <- get_flowlines(locs = region$sp.grid,
                            path = "~/GitHub/species-futures/data/USA/",
                            id.label = "conus.grid.id")
    landcover <- get_landcover(locs = region$sp.grid,
                               path = "~/GitHub/species-futures/data/USA/",
                               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")

    # Combine
    covar <- full_join(tri, stream, by = "conus.grid.id") %>%
      full_join(landcover, by = "conus.grid.id") %>%
      full_join(climate, by = "conus.grid.id") %>%
      full_join(traveltime, by = "conus.grid.id") %>%
      select(conus.grid.id, streamLength.km, prec, forest, elevation, 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 that 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 (as defined in `model-specs.csv`).

```
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 multicollinearity is desired, the following piece of code would check and remove covariates that violate a certain correlation threshold (we use 0.4). If there are fewer than 3 covariates remaining after the procedure, then a message will alert you.

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

  threshold <- 0.4

  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

Now that we've downloaded covariate data, let's take a look at the distribution of each covariate. If there are multiple covariates, we can look at the correlations between them.

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

  ## Distribution covariates

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

  out <- plot_covar(covar,
                    region,
                    cov.labs = covlabs,
                    scaled = T)
  ggsave(out$plot, height = 12, width = 12,
         file = paste0(out.dir, "1_covariates-a_process-map.jpg"))

  out <- cor_covar(covar,
                   cov.labs = covlabs,
                   color.threshold = 0.25)
  ggsave(out$plot, height = 12, width = 12,
         file = paste0(out.dir, "1_covariates-a_process-cors.jpg"))
```

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

    out <- plot_covar(covar,
                      region,
                      cov.labs = covlabs,
                      scaled = T)
    ggsave(out$plot, height = 12, width = 12,
           file = paste0(out.dir, "1_covariates-b_POinat-map.jpg"))

    if (length(covs.inat) > 1) {
      out <- cor_covar(covar,
                       cov.labs = covlabs,
                       color.threshold = 0.25)
      ggsave(out$plot, height = 12, width = 12,
             file = paste0(out.dir, "1_covariates-b_POinat-cors.jpg"))
    }
  }

  ## PO covs

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

  out <- plot_covar(covar,
                    region,
                    cov.labs = covlabs,
                    scaled = T)
  ggsave(out$plot, height = 12, width = 12,
         file = paste0(out.dir, "1_covariates-c_PO-map.jpg"))

  if (length(covs.PO) > 1) {
    out <- cor_covar(covar,
                     cov.labs = covlabs,
                     color.threshold = 0.25)
    ggsave(out$plot, height = 12, width = 12,
           file = paste0(out.dir, "1_covariates-c_PO-cors.jpg"))
  }
}
```

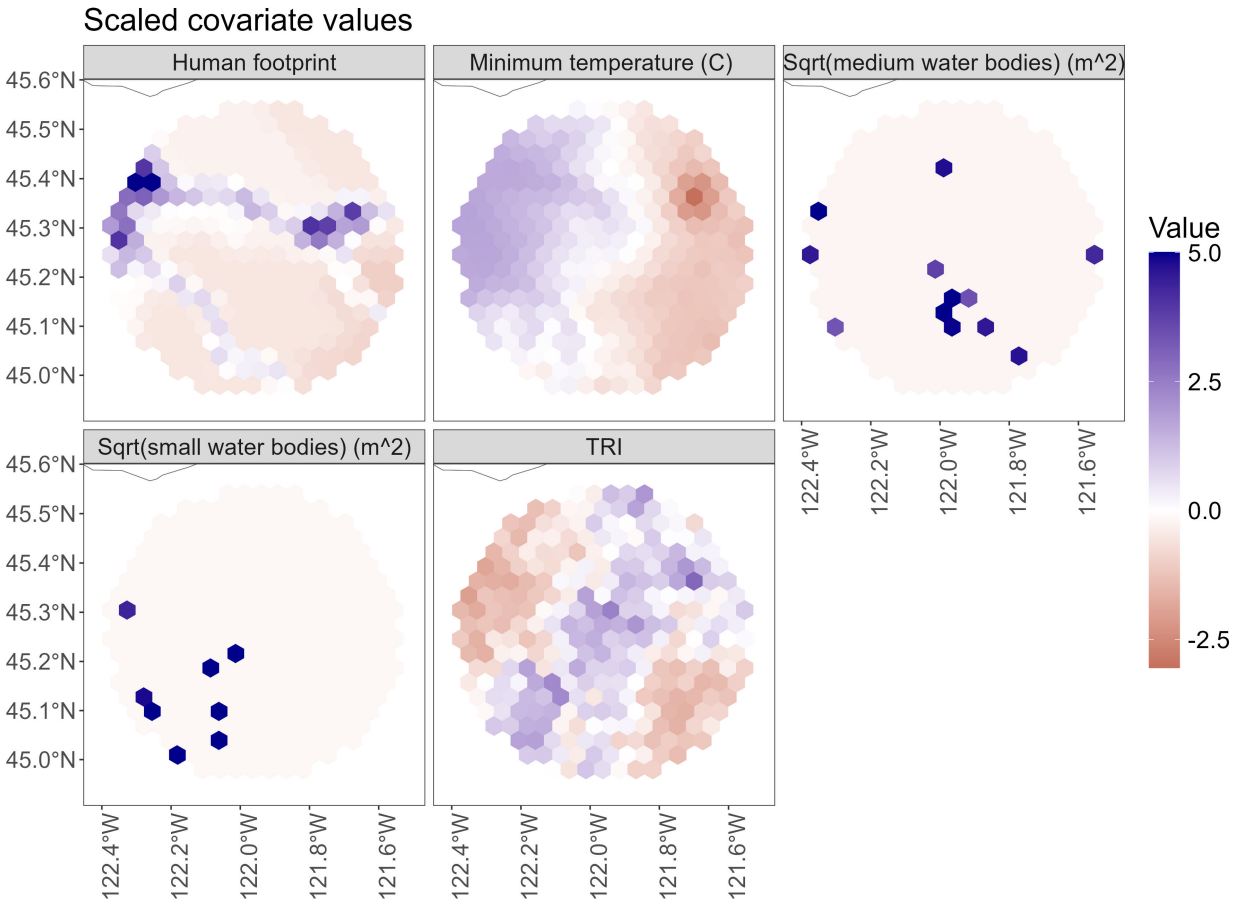


Figure 4: Distribution of distribution-level covariates across the subrange

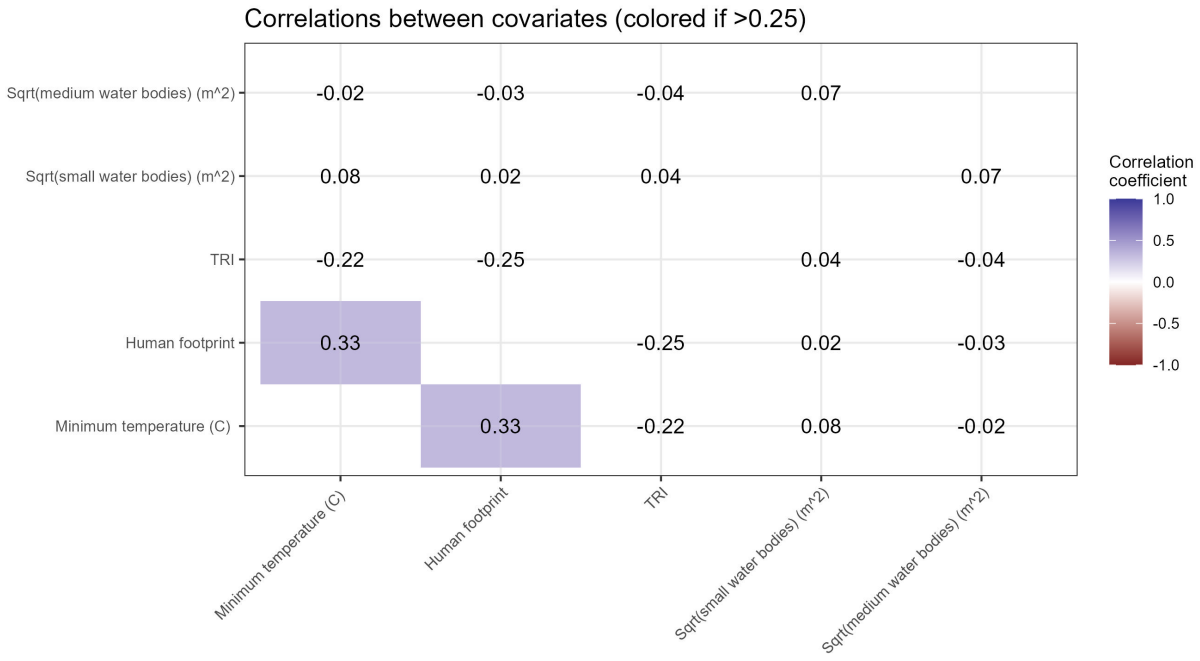


Figure 5: Correlation between distribution-level covariates

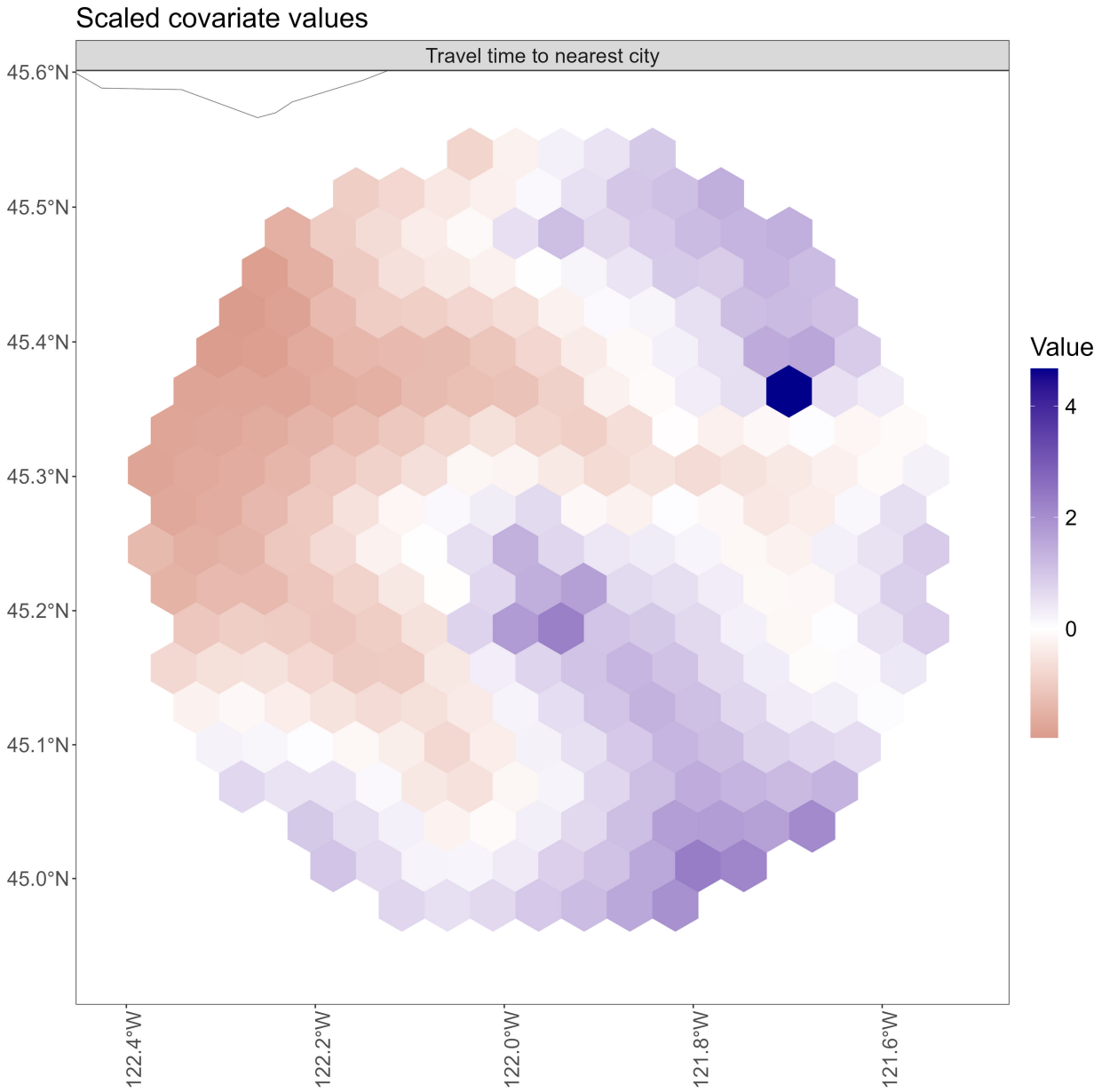


Figure 6: Distribution of presence-only (PO) effort covariates

Step 7: NIMBLE

To load the data into NIMBLE, the data must be formatted in a specific way. We have wrapped this process in a function that combines and formats species datasets (`allfiles`) and their associated covariates (contained in `allfiles`, `covs.inat`, and `covs.PO`)

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

```
  covs.inat = covs.inat,
  covs.PO = covs.PO,
  DND.maybe = 1,  # treat "maybe" detections as 1 or 0?
  # Only keep PO cells that training 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
```

Next, the distribution-level covariates (`covar`) are combined with the formatted species data. This combined data is then broken into `data` and `constants` which are the inputs for the NIMBLE model. As their names imply, the `data` list object contains values which vary over space or time (e.g., species data, distribution-level covariates, and detection covariates). The `constants` list object contains fixed values such as the number of cells in the grid, grid indices for specific datasets, names of datasets, and parameters for the spatial model.

```
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
```

The model structure allows for an indicator variable to indicate whether a cell is in a state with PO effort or not. There are generally two cases where this is useful:

- If a state agency has collected PO only from within its state, by definition there is effort within that state but not outside of it. This information is already stored in `constants`, which pulled it from `allfiles$POextent`
- In iNaturalist data, if a species has taxon geoprivacy (i.e., obscured locations due to listing status) within a state, all records from that state will be removed by `load_species_data()` due to high coordinate uncertainty. The effort in that state is therefore effectively 0.

The following code adds the state indicator variable to `constants`.

```
# GPOR has taxon geoprivacy in four states
if (sp.code == "GPOR") obsc.state <- c("CT", "MS", "NJ", "RI")

# RACA has no iNat data
if (sp.code == "RACA") obsc.state <- NA

constants <- add_state_ind(
  species.data,
  region,
  gridkey,
  constants,
  stategrid = stategrid,
```

```
  obsc.state = obsc.state,
  keep.conus.grid.id = gridkey$conus.grid.id[which(gridkey$group == "train")])
```

Once `data` and `constants` are created for NIMBLE, the function `nimble_code()` writes the NIMBLE code using the information contained in those objects as well several other specificiations defined earlier in the code. The NIMBLE code is saved as a .R file in the location defined by the `path` argument, and in the object `code`.

```
code <- nimble_code(data,
                    constants,
                    path = out.dir,
                    sp.auto = sp.auto,
                    coarse.grid = coarse.grid,
                    Bprior = Bprior,
                    block.out = fold.out,
                    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(eff2[j]) <- A2[1] * Xw2[j,1]
##   E2[j] <- eff2[j] * S2[j]
##
##   # 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)
```

Next, set up the initial values, also based on the information in `data` and `constants`.

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

Finally, identify for which parameters the model should save chains. By default, distribution model parameters (β) and dataset intercepts (α) are saved. Parameters that are estimated for each hexbin (λ, XB, effort) require more storage, so there are options to disable them. If `sp.auto = T`, then the spatial random effect and τ are also saved.

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

Step 8: Clean up and save

Remove objects that are large and are not required for further steps, save the workspace, and move on to `02-flexiSDM.R`.

```
# Remove local and fold 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
suppressWarnings(rm(list=c('local','fold','args','conus.covar.grid','conus.grid', 'conus.huc',
                           'usa','conus', 'pl','centroid')))


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

Part 2: 02-flexiSDM.R

The goal of this script is solely to fit the model via NIMBLE. The code is set up to allow for parallelization of the MCMC chains so they can be fit simultaneously on either a local computer or a high performance computing system.

Because this script can be run independently of `01-flexiSDM.R`, we set some key parameters manually and load the setup file created previously.

```
num <- 3 # model number
fold <- "none" # CV fold

mods <- read.csv("code/model-specs.csv") %>% filter(number %in% num) # get species code and model name
sp.code <- mods$sp.code[1] # species code
model <- mods$model[1] # model name

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


local <- 1 # are you fitting the model locally (1) or on an HPC (0)?
chain <- 1 # chain number (only needed if fitting as an array job on an HPC)
```

```
# Load functions and packages
source("code/FXN-nimbleParallel.R")
library(SpFut.flexiSDM)
```

Fit NIMBLE model

If you intend to fit the model locally, the function `nimbleParallel()` will create a local cluster on your computer with a default of 3 cores (this can be changed using the `cores` argument). It then configures, compiles, and runs the MCMC in parallel for the number of chains you have defined. The output is then saved as a single output object.

If you intend to use an HPC to fit the model, please see the section below on high performance computing.

```
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_',fold,'.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_',fold,'_',chain,'.rds'))
}
```

```
## Jack be NIMBLE...
## Jack be quick...
## Jack jumped over the candlestick!
## Time difference of 1.454409 mins
```

Part 3: 03-flexiSDM.R

The goal of this script is to summarize the chains from the NIMBLE MCMC model and visualize the output in a series of figures.

Because this script can be run independently of `02-flexiSDM.R`, we set load packages, set some key parameters manually, and load the setup file created in the first script before proceeding.

```
## load packages ----
library(tidyverse, quietly = T)
library(magrittr, quietly = T)
library(sf, quietly = T)
library(SpFut.flexiSDM)
```

```
num <- 3 # model number
fold <- "none" # CV fold

mods <- read.csv("code/model-specs.csv") %>% filter(number %in% num) # get species code and model name
sp.code <- mods$sp.code[1] # species code
model <- mods$model[1] # model name

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

local <- 1 # are you fitting the model locally (1) or on an HPC (0)?
maxchain <- 3 # the total number of chains that were fit
```

First we'll load the MCMC samples created by `02-flexiSDM.R`, assuming the model was fit on your local computer. Instructions for models fit with a high performance computing are below. To save time in fitting the model and reduce the size of the samples.rds file, we removed all derived quantities (e.g., occupancy probability (ψ) and projections) from the NIMBLE code. We can calculate those derived quantities now by using the `get_derived()` function, which will create a derived quantity for each step in the MCMC chain.

```
# Load chains (samples) ----
samples <- readRDS(paste0(out.dir,'samples_',fold,'.rds'))

## Calculate derived quantities ----
samples <- lapply(samples, get_derived, data = data, project = project,
                  coarse.grid = coarse.grid, spatRegion = spatRegion)
```

Our next step is to summarize the MCMC chains using the `summarize_samples` function. This function also creates a local cluster on your machine to parallelize the summarization. In this step, you will be summarizing a matrix of

```
## Summarize chains ----
out <- summarize_samples(samples,
                         data,
                         constants,
                         project = project,
                         coarse.grid = coarse.grid,
                         block.out = fold.out,
                         gridkey = gridkey,
                         spatkey = spatRegion$spatkey,
                         effort = T,
                         SLURM = ifelse(local == 1, F, T))

save(out, file = paste0(out.dir, "data", foldname, ".rdata"))
```

All plotting functions all produce a list with two objects: `plot`, a ggplot object with our preset formatting, and `dat`, a dataframe containing the data visualized in `plot`.