

Appendix to Conspecific density dependence in plant communities: a theory-based toolkit for empirical studies

Authors here—

2023-10-31

Table of contents

1	Preface	4
2	Calculating neighborhood densities	5
2.1	Overview	5
2.2	Load libraries and data	6
2.3	Data format explanation	6
2.4	Define exponential decay function	8
2.4.1	Determine which trees are at edge of plot	10
2.5	Calculate distances among focal individual and neighbors	12
2.6	Calculate neighborhood density	15
2.7	Model mortality as a function of neighborhood density	17
2.8	Summarize model fits	21
2.9	Selecting optimum decay parameter values across all species	24
2.10	Selecting optimum decay parameter values separately for each species	27
3	Calculating Marginal Effects	31
3.1	Overview	31
3.2	Load libraries	31
3.3	Load data	32
3.4	Handling “data deficient” species	34
3.5	Function for fitting models	35
3.6	Fit models	38
3.7	Summarize model fits	41
3.8	Plotting results	42
3.9	AMEs Absolute and Relative	43
3.9.1	Settings for AMEs	43
3.9.2	Functions to calculate AMEs and rAME	44
3.9.3	Calculating Absolute Average Marginal Effect (AMEs)	49
3.9.4	Calculating Relative Average Marginal Effect (rAMEs)	50
3.10	Saving results	52
4	Using Meta-regressions to Compare CDD across Species or Sites	53
4.1	Overview	53
4.2	Load libraries and data	54
4.3	Reformat data for model fitting	55

4.4	Fit meta-regression model	56
4.5	Print model summary	56
4.6	Plot the estimated of rAME for all species with a forest plot	57
4.7	Model diagnostics	59
4.8	Model predictions of how species abundance is related to strength of CDD . . .	61

References	63
-------------------	-----------

1 Preface

This document serves as an appendix to the manuscript *Conspecific density dependence in plant communities: a theory-based toolkit for empirical studies*, with the goal of providing code and guidance on conducting analyses of density dependence following the best practices outlined in the main manuscript.

2 Calculating neighborhood densities

2.1 Overview

This tutorial follows from the Section 2. “Modeling considerations when estimating CDD” of the main text:

i Section 2. “Modeling considerations when estimating CDD”

For survival and growth models, defining the density metric and the size of local neighborhoods also require consideration. The density of conspecific and heterospecific neighbors surrounding a focal individual (or plot) can be measured within different distances (i.e., radii). Within a radius, neighbors should be weighted in the density summation with regard to their size and distance to the focal individuals, with the rationale that neighbors at a greater distance and of smaller size have smaller effects. Typically, weighting involves dividing the basal area or diameter by a linear function or an exponential function of distance to allow competitive effects of an individual of a given size to decline with distance (Uriarte et al. 2010; Canham et al. 2006). Biologically meaningful radii should be tested based on the size/life stage, vital rate of interest, and the likely agents of density dependence in the system. For a system of interest, we suggest comparing models with multiple distances (e.g., using maximum likelihood or information criterion) because forests may differ in how neighbors influence performance.

Here, we demonstrate how to calculate the density of conspecific, heterospecific, and all neighbors surrounding a focal individual or plot for a given distance when XY coordinates are known. As a result, this section requires that the data set contains the location of mapped stems. If that is not available (as is common in seedling studies or smaller plots), continue to part 2 of this appendix.

We then demonstrate how to weight the calculation of neighborhood density by individual neighbor size (*i.e.*, basal area) and distance using an exponential decay function (one of many possible decay functions), allowing the competitive effects of density effects to saturate (Uriarte et al. 2010; Canham et al. 2006).

To assess which shape parameters of the exponential decay function are most appropriate for the data set, we fit models with multiple combinations of decay function values and compare models using log likelihood.

From a computational perspective, this approach can be relatively resource intensive both in terms of time and object size. It's possible to make this approach more efficient by subdividing the data (*e.g.*, by plot) or using more efficient data structures such as [data.table](#).

We also note that alternative approaches allow the estimation of the effective scale of neighborhood interactions directly from data (see Barber et al. 2022). An excellent case study using Stan is [available here](#).

i Note

The following code is adapted from the [latitudinalCNDD repository](#) by [Lisa Hülsmann](#).

2.2 Load libraries and data

```
# Load libraries
library(tidyr) # For data manipulation
library(dplyr) # For data manipulation
library(ggplot2) # For data plotting
library(spatstat.geom) # For spatial analysis
library(here) # For managing working directories
library(mgcv) # For fitting gams
library(lubridate) # For calculating census intervals
library(broom) # For processing fit models
library(purrr) # For data manipulation
library(kableExtra) # For table styling
```

2.3 Data format explanation

For this tutorial, we will be using an example data set from Barro Colorado Island (BCI), Panama ([available here](#)) that includes 7,028 observations, of 3,771 individuals of 16 tree species across two censuses from the larger 50 ha BCI 50 ha Forest Dynamics Plot data set ([more information here](#)). Each stem is individually mapped and its diameter at 1.3m above ground is measured (DBH), which allows us to calculate neighborhood density across different distance thresholds and as a function of neighbor size (*e.g.*, basal area), respectively.

The code below assumes the data is in a format where each row is an observation for an individual from a census. For this data set, the column descriptions are as follows:

- **treeID**: unique identifier for each tree
- **sp**: species code

- **gx**: spatial coordinate on x axis
- **gy**: spatial coordinate on y axis
- **dbh**: diameter at breast height (mm)
- **ba**: basal area (m^2)
- **status**: status at census, A = alive, D = dead
- **date**: date of observation
- **census**: census number
- **mort**: mortality status at census, 1 = dead, 0 = alive
- **mort_next**: mortality status at next census, 1 = dead, 0 = alive
- **interval**: time interval between censuses in years

Let's take a quick look at the data set we'll be working with:

```
head(dat, n = 5)
```

```
# A tibble: 5 x 14
  treeID sp      gx      gy    dbh      ba status date      surv surv_next mort
  <chr> <chr> <dbl> <dbl> <dbl> <dbl> <chr> <date>    <dbl>    <dbl> <dbl>
1 3884 ceibpe 296.  24.8  1500 1.77  A    1981-11-10      1          1      0
2 3998 cordbi 280.  45.4   281 0.0620 A    1981-12-11      1          1      0
3 4065 ceibpe 289. 266.   2000 3.14  A    1982-01-08      1          1      0
4 4070 cordbi 283. 287.    356 0.0995 A    1982-01-08      1          1      0
5 4119 ceibpe 258. 224.   1600 2.01  A    1981-12-13      1          1      0
# i 3 more variables: mort_next <dbl>, interval <dbl>, census <dbl>
```

We can produce a plot Figure 2.1 of the tree locations, where the size of the point is scaled to basal area and colored by species, with each census displayed as a panel:

```
ggplot(dat, aes(x = gx, y = gy, size = ba, col = sp)) +
  geom_point() +
  facet_wrap(~census) +
  theme_bw(10) +
  theme(legend.position = "right") +
  labs(size = "Basal area", col = "Species")
```

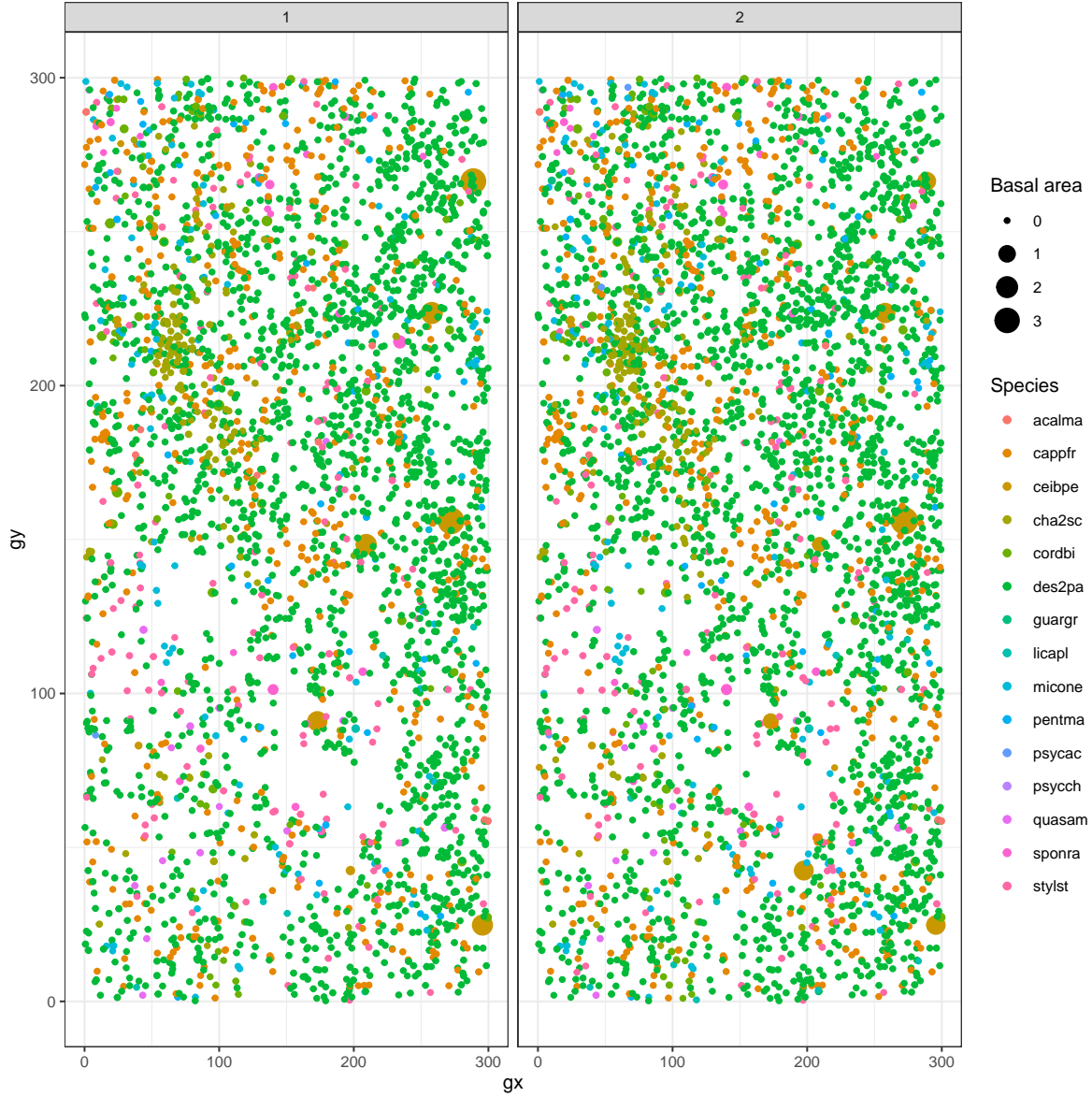


Figure 2.1: Map of tree locations by census

2.4 Define exponential decay function

We will demonstrate how to calculate neighborhood densities using an exponential decay function. In principle, it's possible to use any number of different decay functions and to explore various ranges of values that determine the rate of decay. Here, we present the general

framework for how one might explore different scenarios, though the ultimate choice of decay function and parameter values will depend on the study. **Note that this method of calculating neighborhood density is only possible when neighbors' x-y coordinates are known.**

First, we write a function in R for exponential decay with distance, where mu controls the shape of the curve:

```
exponential_decay <- function(mu, distance){  
  return(exp(-(1/mu * distance)))  
}
```

Let's see what the exponential decay function looks like across a range of mu values (Figure 2.2):

```
# Set range of mu values and distances  
decay_values <- seq(from = 1, to = 25, by = 2)  
  
# Use sprintf to add leading 0s, will help with sorting later on  
decay_names = paste("exp", sprintf("%02s", decay_values), sep = "")  
  
distances <- seq(1, 100, 1)  
  
# Generate a dataframe with each combination of mu and distance  
example_decay_values <- expand_grid(decay_values, distances) %>%  
  # Rename columns  
  rename(decay_value = decay_values,  
         distance = distances)  
  
# Evaluate distance decay function for each combination of  
# mu and distance  
example_decay_values$decay <- exponential_decay(  
  mu = example_decay_values$decay_value,  
  distance = example_decay_values$distance)  
  
# Plot results  
ggplot(example_decay_values,  
  aes(x = distance, y = decay,  
      color = decay_value, group = decay_value)) +  
  ylab("Density") +  
  xlab("Distance (m)") +  
  scale_color_continuous(name = "Value of \ndecay constant") +
```

```
geom_line() +  
theme_bw(12)
```

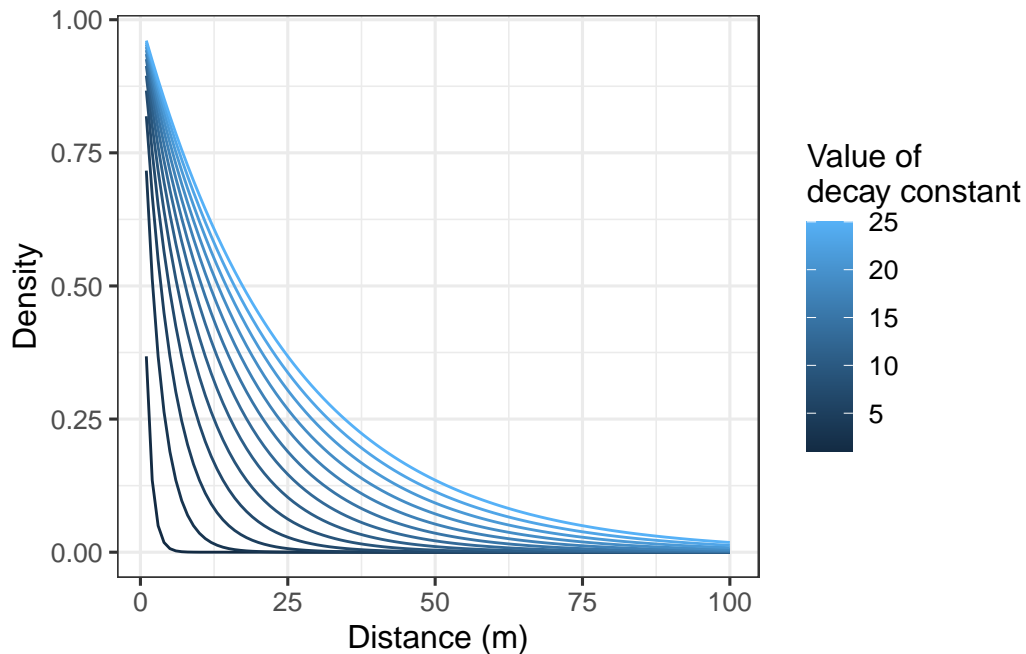


Figure 2.2: Plot of decay function

2.4.1 Determine which trees are at edge of plot

Trees near the edge of plot boundaries have incomplete information about their neighborhood, because neighboring trees outside of the plot boundaries are not mapped. Typically trees within certain distance threshold from the plot edge are excluded from analysis, but still included in calculations of neighborhood densities.

We are going to add a column to our data set called 'edge' that is TRUE if within a set distance to the edge of the plot (*e.g.*, 30 m in the example below).

For this example data set, the dimensions of the overall plot are 300 x 300 m, ranging from 0-300 on both the x and y axis, and representing a subset of the overall 50 ha forest dynamics plot at BCI.

```
# Set threshold for distance to edge  
distance_threshold_edge = 30
```

```
# Add in min and max values for corners of plot
min_x <- 0
max_x <- 300
min_y <- 0
max_y <- 300

dat$edge = dat$gx < min_x + distance_threshold_edge |
           dat$gx > max_x - distance_threshold_edge |
           dat$gy < min_y + distance_threshold_edge |
           dat$gy > max_y - distance_threshold_edge

# How many trees fall within the edge threshold?
table(dat$edge)
```

```
FALSE  TRUE
4526   2502
```

Below is a plot Figure 2.3 of tree locations colored by whether they fall within the edge threshold or not, separated out for each census.

```
ggplot(dat, aes(x = gx, y = gy, col = edge)) +
  geom_point() +
  facet_wrap(~census) +
  coord_fixed(ratio = 1) +
  theme_bw(12)
```

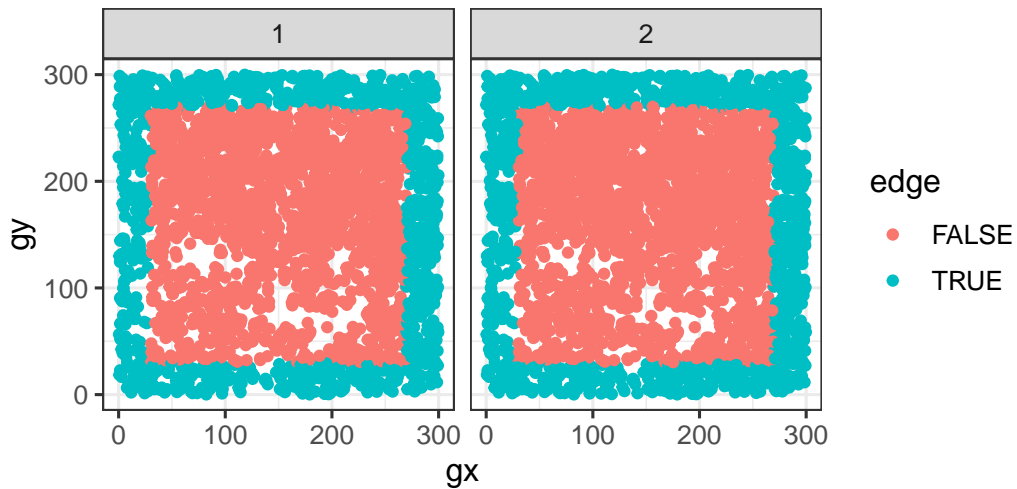


Figure 2.3: Map of tree locations colored by whether they fall within the edge threshold

2.5 Calculate distances among focal individual and neighbors

Next, we will calculate distances among individuals in our plot, using an upper threshold distance of what to consider a neighbor.

We use the `'spatstat.geom'` package to efficiently calculate distances among individuals.

Because this example only includes one census interval (two censuses total), we will subset down to just the first census to calculate neighborhood density.

If the data set were to contain multiple census intervals, it would be necessary to calculate neighborhood density separately for each individual in each census interval, using only the individuals that were alive at the beginning of that census interval.

```
# Set distance threshold for considering neighbors
distance_threshold_neighborhood <- 30

# Subset to first census - this will be different for different
# datasets
dat_first_census <- dat %>%
  filter(census == 1)
```

```

# Format into 'ppp' object
dat_ppp = spatstat.geom::ppp(dat_first_census$gx, dat_first_census$gy,
                             window = owin(range(dat$gx),
                                             range(dat$gy)),
                             checkdup = F)

# Determine close pairs based on distance threshold
# Returns a list that we convert to tibble later
neighbors = spatstat.geom::closepairs(dat_ppp,
                                       rmax = distance_threshold_neighborhood, # Max radius
                                       what = "ij", # return indices i, j, and distance
                                       twice = TRUE)

# Convert to dataframe
neighbors <- as_tibble(neighbors)

# Take a peek at the data
# i = index of focal individual
# j = index of neighbor
# d = distance to neighbor
head(neighbors)

```

```

# A tibble: 6 x 3
      i     j     d
  <int> <int> <dbl>
1  3227  3252  27.0
2  3227  3238  17.1
3  3227  3240   5.75
4  3227  3229  18.6
5  3227  3253  24.4
6  3227  3222  20.8

```

Next we add in additional columns for individual characteristic, (*e.g.*, species, size) and whether they are located near the edge of the plot (blue area in Figure 2.3).

```

# add additional columns

# Add species for individual i
neighbors$sp_i = dat_first_census$sp[neighbors$i]

# Add whether individual i is near edge

```

```

neighbors$edge_i = dat_first_census$edge[neighbors$i]

# Add species for individual j
neighbors$sp_j = dat_first_census$sp[neighbors$j]

# Add basal area of individual j
neighbors$ba_j = dat_first_census$ba[neighbors$j]

```

We then want to add a column that indicates whether the comparison between the focal individual and the neighbor is conspecific or heterospecific because we are interested separately estimating the densities of conspecifics and heterospecifics.

```

neighbors$comparison_type <- ifelse(neighbors$sp_i == neighbors$sp_j,
                                   yes = "con", # conspecific
                                   no = "het") # heterospecific

```

We then remove focal trees that are too close to the edge of the plot.

```

# remove focal trees i that are at the edge
neighbors = neighbors[!neighbors$edge_i, ]

```

Next, we add columns to our neighbors data set that indicates the distance decay multiplier and the distance decay multiplier weighted by basal area.

```

# Loop through distance decay values
for(x in 1:length(decay_values)){

  # Add in column for distance decay multiplier for each decay value
  # add _ba suffix to column name - will eventually be summed based on
  # number of individual neighbors
  neighbors[, paste0(decay_names[x], "_N")] <- exponential_decay(mu = decay_values[x],
                                                                distance = neighbors$d)

  # Weight distance decay multiplier by basal area of neighbor
  # add _ba suffix to column name
  neighbors[, paste0(decay_names[x], "_BA")] <- exponential_decay(
                                                                mu = decay_values[x],
                                                                distance = neighbors$d) * neighbors$ba_j
}

```

Depending on how many distance decay values are being investigated, there may be many columns in the data frame.

```
head(neighbors)
```

```
# A tibble: 6 x 34
      i     j     d sp_i edge_i sp_j     ba_j comparison_type exp01_N exp01_BA
  <int> <int> <dbl> <chr> <lgl> <chr>   <dbl> <chr>           <dbl>   <dbl>
1  2973  2993 14.3 cord~ FALSE capp~ 1.57e-4 het          6.36e- 7 9.98e-11
2  2973  3122 20.4 cord~ FALSE des2~ 4.91e-4 het          1.40e- 9 6.86e-13
3  2973  2974  5.86 cord~ FALSE des2~ 7.85e-5 het          2.85e- 3 2.24e- 7
4  2973  3123 27.2 cord~ FALSE des2~ 7.85e-5 het          1.47e-12 1.16e-16
5  2973  3121 14.5 cord~ FALSE mico~ 1.77e-4 het          5.27e- 7 9.32e-11
6  2973  2936 13.4 cord~ FALSE des2~ 7.85e-5 het          1.45e- 6 1.14e-10
# i 24 more variables: exp03_N <dbl>, exp03_BA <dbl>, exp05_N <dbl>,
#   exp05_BA <dbl>, exp07_N <dbl>, exp07_BA <dbl>, exp09_N <dbl>,
#   exp09_BA <dbl>, exp11_N <dbl>, exp11_BA <dbl>, exp13_N <dbl>,
#   exp13_BA <dbl>, exp15_N <dbl>, exp15_BA <dbl>, exp17_N <dbl>,
#   exp17_BA <dbl>, exp19_N <dbl>, exp19_BA <dbl>, exp21_N <dbl>,
#   exp21_BA <dbl>, exp23_N <dbl>, exp23_BA <dbl>, exp25_N <dbl>,
#   exp25_BA <dbl>
```

2.6 Calculate neighborhood density

Then we summarize neighborhood density for each focal tree separately for conspecifics and heterospecifics.

```
# Simple calculations of number of neighbors and total basal area,
# ignoring distance decay
neighbors_summary <- neighbors %>%
  group_by(i, comparison_type) %>%
  summarise(nodecay_N = n(), # count of neighbors
            nodecay_BA = sum(ba_j)) # sum of basal area)

# Add in decay columns
neighbors_summary_decay <- neighbors %>%
  group_by(i, comparison_type) %>%
  # Select only columns related to distance decay
  select(starts_with("exp")) %>%
  # Summarize them all by summing columns
  summarise_all(sum)

# Join both together
```

```

neighbors_summary <- left_join(neighbors_summary,
                              neighbors_summary_decay,
                              by = c("i", "comparison_type"))

# Add treeID column
neighbors_summary$treeID<-dat_first_census$treeID[neighbors_summary$i]

# If there are any focal individuals with no neighbors, add values
# of 0 for neighborhood densities
noNeighbors = dat_first_census$treeID[!dat_first_census$treeID
                                       %in% neighbors_summary$treeID &
                                       !dat_first_census$edge]

# If there are individuals with no neighbors
if (length(noNeighbors) > 0) {
  neighbors_summary = bind_rows(neighbors_summary,
                                expand_grid(i = NA,
                                             treeID = noNeighbors,
                                             comparison_type = c("het", "cons"))) %>%
  # Add 0s where NA
  mutate_all(replace_na, replace = 0)
}

# Take a peak at the data
head(neighbors_summary)

```

```

# A tibble: 6 x 31
# Groups:   i [6]
      i comparison_type nodecay_N nodecay_BA exp01_N exp01_BA exp03_N exp03_BA
<int> <chr>           <int>     <dbl> <dbl>     <dbl> <dbl> <dbl>
1     5 het           115     0.487 0.0991    4.05e-5 2.03 0.00172
2     8 het            94     0.0539 0.181    3.63e-5 1.21 0.000703
3     9 het           151     2.17 0.0464    1.67e-4 1.65 0.00287
4    10 het            76     0.0509 0.00131 4.72e-7 0.464 0.000175
5    11 het            59     0.0631 0.00859 1.99e-6 0.493 0.000277
6    12 het            93     0.0506 0.0441    9.51e-6 1.57 0.000745
# i 23 more variables: exp05_N <dbl>, exp05_BA <dbl>, exp07_N <dbl>,
# exp07_BA <dbl>, exp09_N <dbl>, exp09_BA <dbl>, exp11_N <dbl>,
# exp11_BA <dbl>, exp13_N <dbl>, exp13_BA <dbl>, exp15_N <dbl>,
# exp15_BA <dbl>, exp17_N <dbl>, exp17_BA <dbl>, exp19_N <dbl>,

```



```
# exp19_BA <dbl>, exp21_N <dbl>, exp21_BA <dbl>, exp23_N <dbl>,
# exp23_BA <dbl>, exp25_N <dbl>, exp25_BA <dbl>, treeID <chr>
```

As described in the main text, it can be advantageous to use total density that includes both conspecific and heterospecific density as a covariate, rather than only heterospecific density.

Here, we calculate overall density by summing heterospecific and conspecific densities.

```
# First convert to long format which will make it easy to sum across
# heterospecific and conspecific values
neighbors_summary_long_format <- neighbors_summary %>%
  pivot_longer(cols = -c("i", "comparison_type", "treeID"))

# Sum across heterospecific and conspecific values and rename to 'all'
neighbors_total_long_format <- neighbors_summary_long_format %>%
  group_by(i, treeID, name) %>%
  summarize(value = sum(value)) %>%
  mutate(comparison_type = "all")

# Bind together conspecific and 'all' densities
# remove heterospecific columns
# fill in 0s where there are no neighbors
neighbors_summary_total = bind_rows(neighbors_summary_long_format,
  neighbors_total_long_format) %>%
  # Can filter out heterospecific neighborhood
  # values by uncommenting this line of the
  # objects become too large
  # filter(comparison_type != "het") %>%
  mutate(name = paste0(comparison_type, "_", name)) %>%
  select(-comparison_type) %>%
  pivot_wider(names_from = name, values_from = value,
    values_fill = 0)
```

2.7 Model mortality as a function of neighborhood density

To determine the ‘best’ decay parameter to use, we fit species-specific mortality models using Generalized Additive Models [GAMs](#) and compare models based on log likelihoods. GAMs are flexible statistical models that combine linear and non-linear components to capture complex relationships in data.

We first create our data set that we will use in the GAMs, subsetting down to just one census interval (because our example dataset only has 2 censuses) and removing trees close to the

edge. In other datasets, you may have multiple census intervals, where it would be common practice to include ‘year’ or ‘census’ as a random effect in the model, but otherwise the overall approach is similar.

```
# Join census data with neighborhood data
dat_gam <- left_join(dat_first_census,
                    neighbors_summary_total,
                    by = "treeID")

# Remove edge trees
dat_gam <- dat_gam %>%
  filter(edge == FALSE)
```

For each species, we summarize data availability to help determine parameters for the GAM smooth terms.

```
# Summarize data availability at species level to set the degree of
# smoothness for GAM smooth terms
sp_summary <- dat_gam %>%
  group_by(sp) %>%
  summarise(ndeath = sum(mort_next),
            nsurv = sum(surv_next),
            range_con_BA = max(con_noddecay_BA) - min(con_noddecay_BA),
            max_con_BA = max(con_noddecay_BA),
            unique_con_BA = length(unique(con_noddecay_BA)),
            unique_all_BA = length(unique(all_noddecay_BA)),
            range_con_N = max(con_noddecay_N) - min(con_noddecay_N),
            max_con_N = max(con_noddecay_N),
            unique_con_N = length(unique(con_noddecay_N)),
            unique_all_N = length(unique(all_noddecay_N)),
            unique_dbh = length(unique(dbh))
  )

# Filter out species that have 0 or 100% mortality
sp_summary <- sp_summary %>%
  filter(ndeath > 0) %>%
  filter(nsurv > 0)
```

In this long block of code, we loop over all possible combinations of decay values for neighborhood densities weighted by abundance (N) and size (BA) for each species and fit a separate GAM for each model. For each GAM, we assess whether the model was able to be fit, and if it was able to be fit, whether it converged and for potential warnings. We save the results of successful model fits into a list that we will process later.

For large datasets where individual GAMs take a long time to run, the code could be modified to run in parallel, either locally on a personal computer or across a computing cluster.

```
# Initialize list that will save model outputs
res_mod <- list()

# Model run settings
run_settings <- expand_grid(species = unique(sp_summary$sp),
                           decay_con = c("nodecay", decay_names),
                           decay_total = c("nodecay", decay_names),
                           nhood_data_type = c("N", "BA"))

# Loop through model run settings
for(run_settings_row in 1:nrow(run_settings)){

  # Extract values from run settings dataframe
  species <- run_settings$species[run_settings_row]
  decay_con <- run_settings$decay_con[run_settings_row]
  decay_total <- run_settings$decay_total[run_settings_row]
  nhood_data_type <- run_settings$nhood_data_type[run_settings_row]

  # Subset down to just focal species
  dat_subset <- dat_gam %>%
    filter(sp == species)

  # Set run name
  run_name <- paste0(species, "_total", decay_total, "_con",
                    decay_con, "_", nhood_data_type)

  # Print status if desired
  # cat("Working on run: ", run_name, " ...\n")

  # Create model formula
  # Initial DBH included as predictor variable
  form = paste0("mort_next ~ s(dbh, k = k1) + s(all_", decay_total,
                    "_", nhood_data_type,
                    ", k = k2) + s(con_", decay_con,
                    "_", nhood_data_type,
                    ", k = k3)")
```

```

# Convert into formula
form <- as.formula(form)

# Choose penalties for model fitting
# set to default 10 (the same as -1)
# The higher the value of k, the more flexible the smooth term becomes, allowing for m
k1 = k2 = k3 = 10
if (k1 > sp_summary$unique_dbh[sp_summary$sp == species]) {
  k1 = sp_summary$unique_dbh[sp_summary$sp == species] - 2
}
if (k2 > sp_summary$unique_all_N[sp_summary$sp == species]) {
  k2 = sp_summary$unique_all_N [sp_summary$sp == species] - 2
}
if (k3 > sp_summary$unique_con_N[sp_summary$sp == species]) {
  k3 = sp_summary$unique_con_N[sp_summary$sp == species] - 2
}

# Fit model
# wrap in a try function to catch any errors
mod = try(gam(form,
  family = binomial(link=cloglog),
  offset = log(interval),
  data = dat_subset,
  method = "REML"),
  silent = T)

# Check if model was able to fit
if (!any(class(mod) == "gam")) {
  # print(paste("gam failed for:", run_name))
} else {

# Check if gam converged
if (!mod$converged) {
  # print(paste("no convergence for:", run_name))
} else {

# check for complete separation
# https://stats.stackexchange.com/questions/336424/issue-with-complete-separation-in
# Explore warning "glm.fit: fitted probabilities numerically 0
# or 1 occurred"

```

```

eps <- 10 * .Machine$double.eps
glm0.resids <- augment(x = mod) %>%
  mutate(p = 1 / (1 + exp(-.fitted)),
         warning = p > 1-eps,
         influence = order(.hat, decreasing = T))
infl_limit = round(nrow(glm0.resids)/10, 0)
# check if none of the warnings is among the 10% most
# influential observations, than it is okay..
num = any(glm0.resids$warning & glm0.resids$influence < infl_limit)

# complete separation
if (num) {
  # print(paste("complete separation is likely for:", run_name))
} else {

  # missing Vc
  if (is.null(mod$Vc)) {
    # print(paste("Vc not available for:", run_name))
  } else {

    # Add resulting model to list if it passes all checks
    res_mod[[run_name]] <- mod

  } # Vc ifelse
} # complete separation ifelse
} # convergence ifelse
} # model available ifelse
} # end run settings loop

```

2.8 Summarize model fits

Next, we will extract regression coefficients into a dataframe using `broom::tidy()`

```

# Extract coefficients for each model into a list
coefs = lapply(res_mod, broom::tidy)

# Add a column for model run to each object in the list
coefs = Map(cbind, coefs, run_name = names(coefs))
coefs = do.call(rbind, coefs) # Bind elements of list together by rows
rownames(coefs) <- NULL # Remove row names

```

run_name	term	edf	ref.df	statistic	p.value
cappfr_totalnodecay_connodecay_N	s(dbh)	1.00	1.00	2.46	0.12
cappfr_totalnodecay_connodecay_N	s(all_nodecay_N)	1.23	1.43	1.25	0.31
cappfr_totalnodecay_connodecay_N	s(con_nodecay_N)	1.00	1.00	1.84	0.17
cappfr_totalnodecay_connodecay_BA	s(dbh)	4.36	5.31	9.83	0.09
cappfr_totalnodecay_connodecay_BA	s(all_nodecay_BA)	1.00	1.00	0.36	0.55
cappfr_totalnodecay_connodecay_BA	s(con_nodecay_BA)	1.00	1.00	3.04	0.08

```

coefs <- coefs %>%
  select(run_name, everything()) # Rearrange columns

# Take a look at the data
knitr::kable(head(coefs), digits = 2, booktabs = T) %>%
  kable_styling(latex_options = c("striped", "scale_down"))

```

Next we will extract model summaries for each model with `broom::glance()` that provides key information like degrees of freedom, log likelihood, AIC, etc.

```

# Extract summaries for each model into a list
sums = lapply(res_mod, broom::glance)

# Add a column for model run to each object in the list
sums = Map(cbind, sums, run_name = names(sums))
sums = do.call(rbind, sums) # Bind elements of list together by rows
rownames(sums) <- NULL # Remove row names

# Separate run name into columns for species, decay, and density type
sums <- sums %>%
  separate(run_name, into = c("sp", "decay_total",
                              "decay_con", "density_type"),
           remove = FALSE)

# Remove 'total' and 'con' from decay columns
sums$decay_total <- gsub("total", "", sums$decay_total)
sums$decay_con <- gsub("con", "", sums$decay_con)

# Rearrange columns
sums <- sums %>%
  select(run_name, sp, decay_total, decay_con, density_type,
         everything())

```

run_name	sp	decay_total	decay_con	density_type	df	logLik	AIC	BIC	deviance	df.residual	nobs
cappfr_totalnodecay_connodecay_N	cappfr	nodecay	nodecay	N	4.23	-22.62	54.10	70.37	45.23	285.77	290
cappfr_totalnodecay_connodecay_BA	cappfr	nodecay	nodecay	BA	7.36	-17.21	51.03	81.52	34.41	282.64	290
cappfr_totalexp01_connodecay_N	cappfr	exp01	nodecay	N	5.06	-22.64	56.52	77.14	45.29	284.94	290
cappfr_totalexp01_connodecay_BA	cappfr	exp01	nodecay	BA	8.08	-17.54	53.90	88.44	35.07	281.92	290
cappfr_totalexp03_connodecay_N	cappfr	exp03	nodecay	N	8.07	-17.87	54.98	90.30	35.73	281.93	290
cappfr_totalexp03_connodecay_BA	cappfr	exp03	nodecay	BA	7.93	-17.62	53.85	87.98	35.24	282.07	290

```
# Take a look at the model summaries
knitr::kable(head(sums), digits = 2) %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

Due to limited sample sizes, it is likely that GAMs will not fit for each species. For example, in the table below of summary data of species where models did not successfully fit/converge, there are several instances of where all individuals of the species survived during the census, leading to no mortality events to use in the model.

We need to exclude species without complete model runs from our overall calculations when looking for optimal decay parameters across the entire data set.

```
# Tally up number of model runs by species and total decay values
table(sums$sp, sums$decay_total)
```

	exp01	exp03	exp05	exp07	exp09	exp11	exp13	exp15	exp17	exp19	exp21
cappfr	28	28	28	28	28	28	28	28	28	28	28
cordbi	27	28	28	28	28	28	28	28	28	27	27
des2pa	28	28	28	28	28	28	28	28	28	28	28
micone	28	28	28	28	28	28	28	28	28	28	28
pentma	28	28	28	28	28	23	27	28	28	28	28
stylst	28	28	28	28	28	28	28	28	28	28	28

	exp23	exp25	nodecay
cappfr	28	28	28
cordbi	28	28	28
des2pa	28	28	28
micone	28	28	28
pentma	28	28	28
stylst	28	28	28

```
# get incomplete run-site-species combinations
run_counts_by_sp <- sums %>%
```

sp	ndead	nsurv	range_con_BA	max_con_BA	unique_con_BA	unique_all_BA	range_con_N	max_con_N	unique_con_N	unique_all_N	unique_dbh
cappfr	5	285	0.02	0.02	273	290	30	32	31	111	31
des2pa	174	1172	0.14	0.15	1335	1343	123	140	123	157	77
micone	38	28	0.00	0.00	48	66	12	12	13	55	11
stylst	3	101	0.02	0.02	91	104	13	13	14	68	30

sp	ndead	nsurv	range_con_BA	max_con_BA	unique_con_BA	unique_all_BA	range_con_N	max_con_N	unique_con_N	unique_all_N	unique_dbh
acalma	2	5	0.00	0.00	4	7	2	2	2	7	6
cordbi	7	36	0.25	0.25	36	43	10	10	11	38	28
pentma	12	21	0.00	0.00	20	33	6	6	6	31	11
sponra	3	15	0.08	0.08	12	18	5	5	4	17	16

```

group_by(sp) %>%
tally() %>%
# Join with overall species list
left_join(sp_summary %>% select(sp), ., by = "sp")

# Get expected number of runs if all models work
expected_total_runs <- run_settings %>%
  group_by(species) %>%
  tally() %>%
  pull(n) %>%
  max()

# Save species names where they didn't have all expected combinations
# of model runs
incomplete = run_counts_by_sp$sp[run_counts_by_sp$n <
  expected_total_runs |
  is.na(run_counts_by_sp$n)]

# Species with successful runs
knitr::kable(sp_summary[!sp_summary$sp %in% incomplete, ],
  digits = 2) %>%
  kable_styling(latex_options = c("striped", "scale_down"))

# Species without successful runs
knitr::kable(sp_summary[sp_summary$sp %in% incomplete, ],
  digits = 2) %>%
  kable_styling(latex_options = c("striped", "scale_down"))

```

2.9 Selecting optimum decay parameter values across all species

We then summarize different model criteria across all species runs. To look for the optimal value for decay parameters, we sum log likelihoods across all species for a given decay parameter

combination and choose the resulting parameter combination with the highest summed log likelihood.

```
sums_total <- sums %>%
  filter(!sp %in% incomplete) %>%
  group_by(decay_total, decay_con, density_type) %>%
  summarise(nvalues = n(),
            sumlogLik = sum(logLik),
            meanlogLik = mean(logLik)) %>%
  arrange(decay_total, decay_con, density_type)

sums_total
```

```
# A tibble: 392 x 6
# Groups:   decay_total, decay_con [196]
  decay_total decay_con density_type nvalues sumlogLik meanlogLik
  <chr>        <chr>    <chr>          <int>    <dbl>      <dbl>
1 exp01      exp01      BA              4    -556.     -139.
2 exp01      exp01      N               4    -554.     -138.
3 exp01      exp03      BA              4    -553.     -138.
4 exp01      exp03      N               4    -558.     -140.
5 exp01      exp05      BA              4    -554.     -138.
6 exp01      exp05      N               4    -561.     -140.
7 exp01      exp07      BA              4    -557.     -139.
8 exp01      exp07      N               4    -563.     -141.
9 exp01      exp09      BA              4    -558.     -140.
10 exp01     exp09      N               4    -564.     -141.
# i 382 more rows
```

We create a heatmap plot Figure 2.4 of summed log likelihoods for all parameter combinations across all species, with the optimal parameter combination marked with an X

```
# Find optimum value separately for N and BA
optimum <- sums_total %>%
  group_by(density_type) %>%
  slice_max(sumlogLik)

# Plot heatmap of log likelihood values
ggplot(sums_total, aes(x = decay_total, y = decay_con,
                      fill = sumlogLik)) +
  geom_tile(width = 0.9, height = 0.9, col = "black") +
```

decay_total	decay_con	density_type	nvalues	sumlogLik	meanlogLik
exp19	exp25	BA	4	-541.69	-135.42
exp03	exp01	N	4	-541.59	-135.40

```

scale_fill_viridis_c() + # viridis color palette
geom_label(data = optimum, label = "X") +
labs(x = "Decay total density", y = "Decay conspecific density",
     fill = "sumlogLik") +
facet_wrap(~density_type, ncol = 1) +
theme_bw(12) +
theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
                                   hjust=1))

```

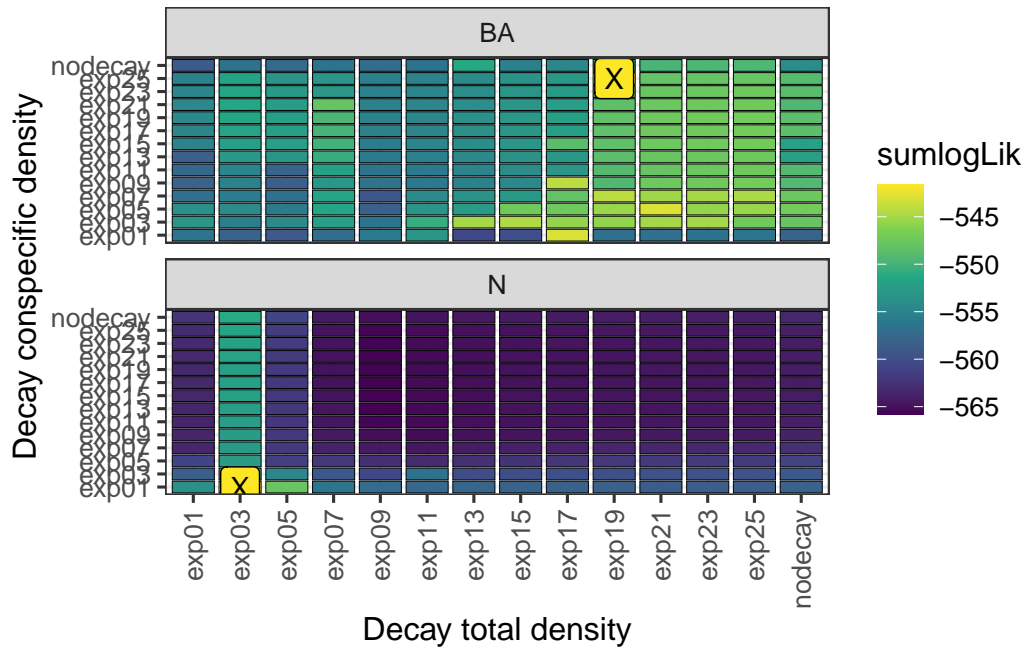


Figure 2.4: Heatmap of optimal values for decay constants

For this data set, the following are the optimal decay parameter values across all species separately for neighborhood density calculated by abundance (N) and by basal area (BA)

```

knitr::kable(optimum, digits = 2) %>%
  kable_styling(latex_options = c("striped", "scale_down"))

```

2.10 Selecting optimum decay parameter values separately for each species

Alternatively, it may be useful to determine optimum decay parameters on a species by species basis. To do this, we find the maximum log likelihood for each species separately across decay parameter combination and choose the resulting parameter combination with the highest log likelihood.

```
sums_total_by_spp <- sums %>%  
  filter(!sp %in% incomplete) %>%  
  group_by(decay_total, decay_con, density_type, sp) %>%  
  summarise(nvalues = n(),  
            sumlogLik = sum(logLik),  
            meanlogLik = mean(logLik)) %>%  
  arrange(decay_total, decay_con, density_type)  
  
sums_total_by_spp
```

```
# A tibble: 1,568 x 7  
# Groups:   decay_total, decay_con, density_type [392]  
  decay_total decay_con density_type sp      nvalues sumlogLik meanlogLik  
  <chr>      <chr>    <chr>    <chr>    <int>    <dbl>    <dbl>  
1 exp01      exp01      BA      cappfr      1     -16.6     -16.6  
2 exp01      exp01      BA      des2pa      1    -489.    -489.  
3 exp01      exp01      BA      micone      1    -43.7    -43.7  
4 exp01      exp01      BA      stylst      1     -6.84    -6.84  
5 exp01      exp01      N       cappfr      1    -14.3    -14.3  
6 exp01      exp01      N       des2pa      1    -489.    -489.  
7 exp01      exp01      N       micone      1    -41.3    -41.3  
8 exp01      exp01      N       stylst      1     -8.64    -8.64  
9 exp01      exp03      BA      cappfr      1    -20.7    -20.7  
10 exp01      exp03      BA      des2pa      1   -485.    -485.  
# i 1,558 more rows
```

We create a heatmap plot Figure 2.5 of log likelihoods for all parameter combinations for each species separately, with the optimal parameter combination marked with an X. We only display the first three species here, because with data sets containing many species, it will be hard to visualize all the species on one graph and you may need to subdivide the plot further for visualization.

```

sums <- sums %>%
  filter(sp %in% c("cappfr", "cordbi", "des2pa")) %>%
  group_by(sp) %>%
  # Scale loglikelihood by species to help with visualization
  mutate(logLik_scaled = scale(logLik)) %>%
  ungroup()

# Find optimum value separately for N and BA
optimum_by_sp <- sums %>%
  group_by(sp, density_type) %>%
  slice_max(logLik_scaled, with_ties = FALSE)

# Plot heatmap of log likelihood values
ggplot(sums, aes(x = decay_total, y = decay_con,
  fill = logLik_scaled)) +
  geom_tile(width = 0.9, height = 0.9, col = "black") +
  geom_label(data = optimum_by_sp, label = "X") +
  labs(x = "Decay total density", y = "Decay conspecific density",
  fill = "logLik") +
  facet_wrap(~sp + density_type, ncol = 2, scales = "free") +
  scale_fill_viridis_c() + # viridis color palette
  theme_bw(12) +
  theme(legend.position = "right") +
  labs(fill = "Log likelihood\n(scaled)") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
    hjust=1))

```

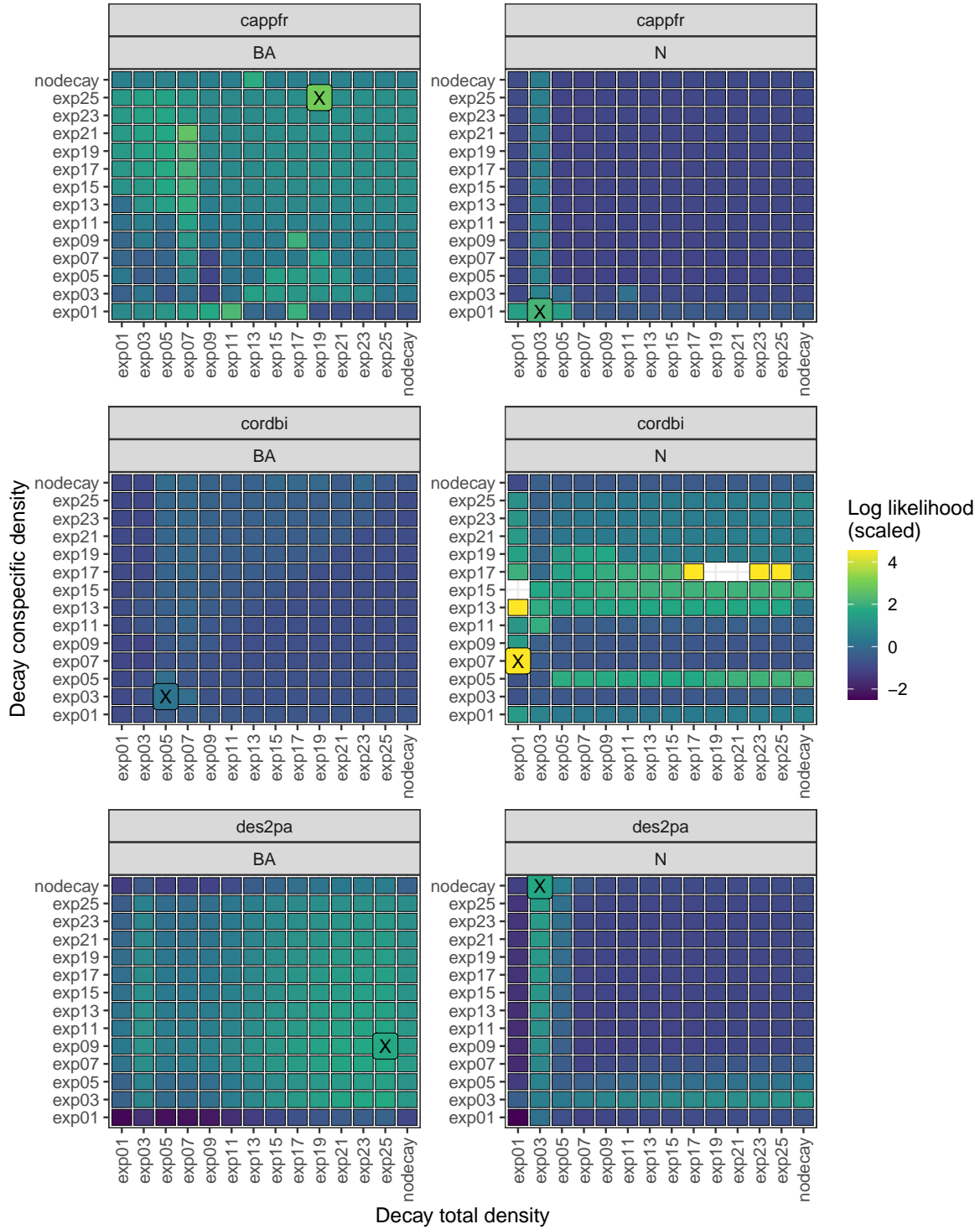


Figure 2.5: Heatmap of optimal values for decay constants separately for each species

run_name	sp	decay_total	decay_con	density_type	df	logLik	AIC	BIC	deviance	df.residual	nobs	logLik_scaled
cappfr_totalexp19_conexp25_BA	cappfr	exp19	exp25	BA	11.09	-8.74	43.29	90.63	17.48	278.91	290	2.9564926
cappfr_totalexp03_conexp01_N	cappfr	exp03	exp01	N	11.03	-11.80	49.64	97.43	23.59	278.97	290	2.1291275
cordbi_totalexp05_conexp03_BA	cordbi	exp05	exp03	BA	7.61	-12.46	43.10	58.90	24.93	34.39	42	0.2471958
cordbi_totalexp01_conexp07_N	cordbi	exp01	exp07	N	14.84	0.00	31.83	59.48	0.00	27.16	42	4.5489731
des2pa_totalexp25_conexp09_BA	des2pa	exp25	exp09	BA	8.36	-481.80	984.07	1037.31	963.61	1337.64	1346	1.7645474
des2pa_totalexp03_connodecay_N	des2pa	exp03	nodecay	N	7.33	-481.92	981.23	1026.49	963.84	1338.67	1346	1.6992914

For this data set, the following are the optimal decay parameter values for each species separately for neighborhood density calculated by abundance (N) and by basal area (BA):

```
knitr::kable(optimum_by_sp, digits = 2) %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

3 Calculating Marginal Effects

3.1 Overview

In this section, we examine a *subset* of the Barro Colorado Island (BCI) seedling data to illustrate the impact of conspecific density on mortality probability. We calculate the Average Marginal Effect ([more about marginal effects in general here](#)) as our metric of the strength conspecific density dependence. We then estimate both the absolute Average Marginal Effect (AME) and the relative Average Marginal Effect (rAME). The **AME** represents the average absolute change in mortality probability due to a specified increase in conspecific density. In contrast, the **rAME** is the relative change in mortality probability compared to a baseline value.

The standard approach for modeling plant CDD patterns presumes a linear relationship between performance (e.g., mortality/survival) and conspecific neighborhood density metrics. However, in this section, we use ‘[Generalized Additive Models \(GAMs\)](#)’ as it offer flexibility for non-linear relationships between performance and predictors.

Note

Note: The code is adapted from the [latitudinalCNDD repository](#) by [Lisa Hüelsmann](#).

3.2 Load libraries

```
# Load libraries
library(boot)
library(broom)
library(dplyr)
library(ggplot2)
library(gratia)
library(here)
library(kableExtra)
library(knitr)
library(lubridate)
```

```
library(mgcViz)
library(mgcv)
library(MASS)
library(parallel)
library(pbapply)
library(readr)
library(skimr)
library(spatstat.geom)
library(tidyr)
```

3.3 Load data

Our demonstration utilizes a subset of the Barro Colorado Island (BCI) seedling data, encompassing only 30 species. Please note that this analysis is *solely for demonstration purposes*; hence, no biological conclusions should be drawn from the results due to the limited data subset used.

The code below assumes the data is in a format where each row is an observation for an individual from a census. For this particular data set, the column descriptions are as follows:

- **Id:** unique identifier for each seedling
- **plot:** location
- **spp:** species
- **date:** date of observation
- **year:** year of observation
- **census:** census number
- **status:** status at census, 0 = alive, 1 = dead
- **height.last.census:** height.last.census
- **con_dens:** number of conspecific neighbors
- **total_dens:** number of total neighbors
- **het_dens:** number of heterospecifics neighbors
- **time.since.last.census:** time interval between censuses

```
data_BCI <- read_csv("../data/BCI_seedling_data_30_spp_2023.csv")
colnames(data_BCI)
```



```

[1] "id"                "q20"
[3] "plot"              "tag"
[5] "spp"               "date"
[7] "year"              "census"
[9] "status"            "height.last.census"
[11] "height.last.census.log.scaled" "con_dens"
[13] "total_dens"        "het_dens"
[15] "time.since.last.census"

```

```

# make sure variables are taken as factor
data_BCI$census <- factor(data_BCI$census)
data_BCI$spp <- factor(data_BCI$spp)
data_BCI$plot <- factor(data_BCI$plot)
data_BCI$height=data_BCI$height.last.census
data_BCI$interval=as.numeric(data_BCI$time.since.last.census)

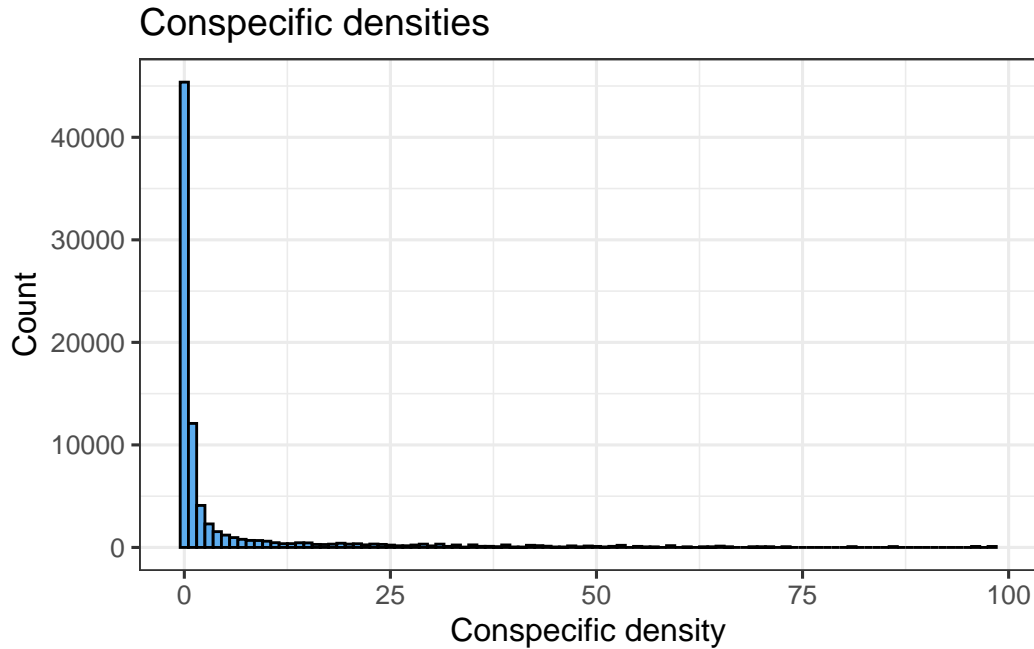
```

Let's take a quick look at the data set we'll be working with:

```

# Exploring data
# visualize
ggplot(data_BCI, aes(x = con_dens)) +
  geom_histogram(binwidth = 1, color = "black", fill = "steelblue2") +
  labs(x = "Conspecific density", y = "Count",
       title = "Conspecific densities") +
  theme_bw(12)

```



3.4 Handling “data deficient” species

We categorize a species as *data deficient* if it has fewer than four unique conspecific density values. At the end of this section of the code, a data frame named ‘**nsp**’ will be generated. This data frame will classify each species as either “data deficient” or “not data deficient”.

From our data set 9 species out of 30 were assigned as *data deficient*. Here, we used the arbitrary thresholds of 4 unique values of conspecific densities and a minimum range of 1 for conspecific as our thresholds for what constitutes a data deficient species. These exact values will depend on the data set and the questions of the study, so we suggest thinking carefully about what the appropriate values are for the purposes of a given study.

```
nval = 4 ## this is the number of 'unique' conspecific values
minrange = 1 # minimum range for conspecific density

data_BCI %>%
  group_by(spp) %>%
  summarise(
    range_con_dens = max(con_dens) - min(con_dens),
    max_con_dens = max(con_dens),
    unique_con_dens = length(unique(con_dens)),
```

spp	range_con_dens	max_con_dens	unique_con_dens	unique_total_dens	unique_height	issue_nval	issue_range	trymodel	data_deficient
ACALDI	7	7	8	35	379	FALSE	FALSE	TRUE	FALSE
AEGICE	4	4	5	28	499	FALSE	FALSE	TRUE	FALSE
BEILPE	98	98	69	72	894	FALSE	FALSE	TRUE	FALSE
CAPPFR	5	5	6	52	767	FALSE	FALSE	TRUE	FALSE
CECRIN	7	7	7	20	177	FALSE	FALSE	TRUE	FALSE
CORDLA	4	4	5	32	457	FALSE	FALSE	TRUE	FALSE

```

        unique_total_dens = length(unique(total_dens)),
        unique_height = length(unique(height.last.census))
    ) %>%

# check if conspecific less than defined nval
mutate(issue_nval = unique_con_dens < nval,
# range should at least be equal to minrange
issue_range = range_con_dens < minrange,
trymodel = !(issue_nval|issue_range),
# Assignment of "data deficient" species
data_deficient = !trymodel
) -> nsp # Store the resulting dataframe in the object 'nsp'

# Visualize the top rows of the table 'nsp' in a formatted manner
nsp %>%
  head() %>%
  kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))

```

3.5 Function for fitting models

In the context of the Janzen-Connell Hypothesis, we focus on the differences between CDD and HDD (‘Stabilizing effect’), where conspecific neighbors’ negative effects surpass those from heterospecifics, leading to population stabilization (Broekman et al. 2019). This effect is vital for estimating species’ self-limitation.

In this section we illustrate the conspecific density impact, considering total tree density. The estimated effect (slope) in our analysis corresponds to the result of subtracting HDD from CDD.

We use here a Generalized Additive Model (GAM) with a complementary log-log (cloglog) link function to model the seedling status (‘alive’ or ‘dead’) as a function of conspecific density **con_dens**, total density **total_dens** and tree height or size, the latter serving as a potential confounder or precision covariate.

The cloglog link accounts for differences in observation time Δ through an offset term, applicable to datasets where (0=alive and 1=alive)(Currie 2016).

After defining covariates for the GAM models, we specify the smooth term in the model formula. If a k value exceeds the number of unique values in the variable, it is adjusted to be two less than that number. Model convergence and warnings are also checked. which this set up here we define k=2 as minimum as K=1 is a linear relation.

After defining the covariates for the GAMs, we specify the smooth term in the model formula. The k value determines the complexity of the smooth. If k exceeds the number of unique values in a variable, it's adjusted to be two less than that number. We also monitor model convergence and any warnings that may arise.

```
# Define a function to fit a model to the given data
model_fit = function(data, spinfo, reduced = F) {

  # create new factor with correct factor levels per species
  data$census = factor(data$census)

  # # Determine if there's more than one unique value in 'census'
  # and construct the relevant term for the model formula

  term_c = ifelse(length(unique(data$census)) > 1,
    "+ s(census, bs = 're')", "")
  #term_p = "+ s(plot, bs = 're')"

  if (reduced) {
    form = as.formula(paste0("status ~ s(height, k = k1) +
      s(total_dens, k = k2)"
      , term_c)) # reduced model #,term_p
  } else {
    form = as.formula(paste0("status ~ s(height, k = k1) +
      s(total_dens, k = k2) +
      s(con_dens, k = k3)"
      , term_c)) # full model #, term_p
  }

  # Choose penalty
  # set to default k=10
  k1 = k2 = k3 = 10
  if (k1 > spinfo$unique_height) k1 = spinfo$unique_height - 2
  if (k2 > spinfo$unique_total_dens) k2 = spinfo$unique_total_dens - 2
```

```

if (k3 > spinfo$unique_con_dens) k3 = spinfo$unique_con_dens - 2

# k = 1 would force linearity for that term, and we aim to consider
# also potential non-linear relationships, so conspecific k is k=2
# minimum.

# Fit the Generalized Additive Model (GAM)
mod = try(gam(form
  , family = binomial(link=cloglog)
  , offset = log(interval)
  , data = data
  , method = "REML"
) , silent = T
)
# Return the fitted model
return(mod)
}

# check model run

# Define a function to check the convergence of the model
model_convergence = function(model) {

  # gam not available
  if (!any(class(model)=="gam")) {
    print(paste(spp, "gam failed"))
  } else {

    # gam not converged
    if (!model$converged) {
      print(paste(spp, "no convergence"))
    } else {

# Explore warning "glm.fit: fitted probabilities numerically 0 or 1
# occurred (complete separation)"
eps <- 10 * .Machine$double.eps
glm0.resids <- augment(x = model) %>%
  mutate(p = 1 / (1 + exp(-.fitted)),
    warning = p > 1-eps,

```

```

        influence = order(.hat, decreasing = T))
infl_limit = round(nrow(glm0$resids)/10, 0)

# Check if any of the warnings correspond to the 10% most
# influential observations. If not, then it is okay.

num = any(glm0$resids$warning & glm0$resids$influence <
          infl_limit)

# If there's complete separation
if (num) {
  print(paste(spp, "complete separation is likely"))
} else {

  # Check if the Vc component of the model is missing
  if (is.null(model$Vc)) {
    print(paste(spp, "Vc not available"))
  } else {

    # If everything is fine, return the model
    return(model)
  }
}
}
}
}

```

3.6 Fit models

Here we fit the model for all species. *data deficient* species are treated as one group since there is not sufficient data to be treated independently. In this dataset, we have 9 species that are flagged as 'data deficient',

```

# Check the distribution of species marked as 'data deficient'
# (T or F)
# and determine the number of species for which we will try the model

# We have 9 species that are flagged as 'data deficient'
table(data_deficient = nsp$data_deficient, trymodel = nsp$trymodel)

```

```

      trymodel
data_deficient FALSE TRUE
      FALSE      0    21
      TRUE       9     0

# Convert spp to character
data_BCI$spp <- as.character(data_BCI$spp)

# Extract species that are flagged as 'data deficient' from the nsp
# dataframe
data_deficient_species <- nsp$spp[nsp$data_deficient == TRUE]

# Replace species names with "data_deficient_seedling" for those
# flagged as 'data_deficient'
data_BCI2 <- data_BCI %>%
  mutate(spp = ifelse(spp %in% data_deficient_species,
                      "data_deficient_seedling", spp))

# # Summarize attributes for each species in the modified dataframe
# (including the new "#data_deficient_seedling" group)

data_BCI2 %>%
  group_by(spp) %>%
  summarise(
    range_con_dens = max(con_dens, na.rm = TRUE) - min(con_dens,
                                                         na.rm = TRUE),

    max_con_dens = max(con_dens, na.rm = TRUE),
    unique_con_dens = length(unique(con_dens)),
    unique_total_dens = length(unique(total_dens)),
    unique_height = length(unique(height.last.census))
  ) %>%
  mutate(
    # less than nval unique values in consp densities
    issue_nval = unique_con_dens < nval,
    # range should at least be equal to minrange
    issue_range = range_con_dens < minrange,
    trymodel = !(issue_nval | issue_range),
    # preliminary assignment of data_deficient species
    data_deficient = !trymodel
  ) -> nsp_data_deficient

```

```

####
## Fit model for each species
###

# Create lists to store results of the main and reduced model fits

# List for main model fits
res_mod = list()
# List for reduced model fits (for calculating Pseudo R2)
res_red_mod = list()

# Loop through species in the nsp_data_deficient dataframe for which
# we will try modeling
# (Here, the group of data_deficient species is treated as a single
# species)

for(spp in nsp_data_deficient$spp[nsp_data_deficient$trymodel]) {

  # select data for individual species
  dat_sp = data_BCI2[data_BCI2$spp == spp, ]

  # Fit the main and reduced models for the current species
  mod = model_fit(data = dat_sp,
                  spinfo = nsp_data_deficient[nsp_data_deficient$spp
                                              == spp, ])
  mod_red = model_fit(data = dat_sp,
                      spinfo = nsp_data_deficient[nsp_data_deficient$spp
                                                  == spp, ], reduced = T)

  # Check the convergence of both models
  res = model_convergence(model = mod)
  res_red = model_convergence(model = mod_red)

  # save result
  if (is.character(res)) {
    nsp$data_deficient[nsp$spp == spp] = T
  } else {
    res_mod[[spp]] = res
    res_red_mod[[spp]] = res_red
  }
}

```



```
}
```

3.7 Summarize model fits

Regression table via `broom::tidy()`

```
coefs = lapply(res_mod, broom::tidy)
coefs = Map(cbind, coefs, sp = names(coefs))
coefs = do.call(rbind, coefs)
```

Model summary via `broom::glance()`

```
# For each model result in 'res_mod', extract summary statistics
# (like log-likelihood, AIC, BIC #, etc.) using the 'glance' function
# from the 'broom' package

# df logLik AIC BIC deviance df.residual nobs
sums = lapply(res_mod, broom::glance)
sums = Map(cbind, sums, sp = names(sums))
sums = do.call(rbind, sums)
head(sums)
```

	df	logLik	AIC	BIC	deviance	df.residual	nobs
ACALDI	7.738186	-428.3495	876.0873	924.9256	856.6990	1131.2618	1139
AEGICE	8.763874	-524.7678	1070.5298	1123.3674	1049.5355	1125.2361	1134
BEILPE	17.776666	-11142.3710	22323.8580	22478.1537	22284.7420	19697.2233	19715
CAPPFR	10.232458	-2024.4980	4075.8754	4174.3285	4048.9960	11210.7675	11221
CECRIN	6.221103	-143.4318	301.0402	326.2521	286.8635	252.7789	259
CORDLA	6.840230	-531.0591	1079.4468	1125.3893	1062.1182	1477.1598	1484

```
sp
ACALDI ACALDI
AEGICE AEGICE
BEILPE BEILPE
CAPPFR CAPPFR
CECRIN CECRIN
CORDLA CORDLA
```

```
# AUC
aucs = lapply(res_mod, function(x) {
```

```

    roc <- performance::performance_roc(x, new_data = x$model)
    bayestestR::area_under_curve(roc$Spec, roc$Sens)
  })
sums$AUC = unlist(aucs)

# Pseudo R2
sums$pseudoR2 = 1 - (unlist(lapply(res_mod, function(x) x$deviance)) /
                     unlist(lapply(res_red_mod, function(x)
                                   x$deviance)))

```

3.8 Plotting results

```

# plot splines in pdf -----

# # Specify the name of the PDF file where the plots will be saved
pdf_file <- "mortality.pdf"

# Open the PDF file for writing
pdf(pdf_file)

# Loop through all the model results stored in 'res_mod'
for (i in 1:length(res_mod)) {

  # Get the vizmod for the current species
  vizmod <- getViz(res_mod[[i]], post = T, unconditional = T)
  pl <- plot(vizmod, nsim = 20, allTerms = T) +
    # Add confidence interval line/ fit line/ simulation line
    l_ciLine() + l_fitLine() + l_simLine() +
    #Add confidence interval bar/# Add fitted points
    l_ciBar() + l_fitPoints(size = 1) +
    l_rug() + # Add rug plot
    # Add title with the name of the current species
    labs(title = names(res_mod)[i])

  # Print the plot to the R console only for the first 3 species
  if (i <= 3) {
    print(pl, pages = 1)
  }
}

```

```

    # Save the plot to the PDF
    print(pl, pages = 1)
}

# Close the PDF file
dev.off()

```

pdf
2

3.9 AMEs Absolute and Relative

In this section we illustrate the calculation of the average marginal effect and both the absolute Average Marginal Effect (AME) and the relative Average Marginal Effect (rAME).

Here, **AMEs** are computed by determining the marginal effect (essentially the partial derivative or slope) of a predictor for a unit change in conspecific density at each observed value, and then averaging these effects. This method yields a single, interpretable measure that offers an averaged estimate of the predictor’s impact. It’s worth noting that the AME, compared to traditional effect sizes, provides a clearer measure of a predictor’s influence by quantifying the average change in the outcome for each unit change in the predictor rather than the raw coefficient (effect size) that may be influenced by the scale of the variable or confounded by interactions with other predictors.

Furthermore, **rAMEs** enhance the level of interpretability by delivering a normalized measure of these averaged effects. They represent the percentage change in the response variable due to a unit change in the predictor relative to the base mortality, providing an intuitive, relative grasp of the predictor’s influence. This normalization process allows for the comparison of effects across different species, each with its own base level of mortality.

For **AMEs** we need the file “res_model” that includes all the models results.

There is also alternative approaches for calculating average marginal effects using ‘[marginal-effects](#)’ package that we encourage the user to explore.

3.9.1 Settings for AMEs

Here we provide three scenarios for calculating **AMEs** or **rAMEs** through the **change** argument.

Equilibrium: This scenario computes **AMEs** or **rAMEs** for a unit increase in conspecific density, providing insight into the marginal effect of density increase in an existing ecosystem.

Invasion: This scenario models the effects of introducing species into a new habitat, transitioning the conspecific density from zero to a specified unit, helping understand the impact of sudden species introductions. **IQR:** This scenario evaluates **AMEs** or **rAMEs** within the middle 50% range of conspecific density, offering a perspective less influenced by extremes, hence providing a more robust understanding of effects within typical density ranges.

```
#### chose predictors for local density -setting AMEs

# Define a vector of predictors with their corresponding names
predictors <- c(con_dens = "con_dens",
               total_dens = "total_dens")

# change in conspecific density for AME calculations----

# One more neighbor seedling
# or for adult trees: pi*((dbh_neighbor/1000)/2)^2 *
                        #dec_fun(decay_con,dist_neighbor, decay_type)
additive=1

# different change settings for con-specific densities

interval = 1

# Specify how the conspecific density should be changed for
# different scenarios:
# 'equilibrium', 'invasion', and 'iqr' (interquartile range)

change = list(equilibrium = data.frame(con_dens =
                                       paste('+', additive))
              , invasion = data.frame(con_dens = "c(0, additive)")
              , iqr = data.frame(con_dens = "c(q1, q3)")
              )

## Set the number of iterations for any subsequent calculations or
# simulations
iter = 500
```

3.9.2 Functions to calculate AMEs and rAME

This section defines two functions. `setstep` and `get_AME`. `Get_AME` is the function calculates the Average Marginal Effects for a given term in a model. The function first creates two copies

of the data, **d0** and **d1**, and adjusts the term of interest based on the **change** argument (one of the three scenarios). It then calculates the predictions for the two data sets and computes the marginal effects.

```
# Define a function to set the step size for numerical derivatives
# This function determines an appropriate step size using machine's
# precision/machine epsilon to strike a balance between accuracy and
# rounding errors.
```

```
setstep = function(x) {
  eps = .Machine$double.eps
  return(x + (max(abs(x), 1, na.rm = TRUE) * sqrt(eps)) - x)
}
```

```
# Function to compute Average and relative Marginal Effects
# (AME and rAME) for a given model-----
```

```
get_AME = function(mod, data, term
                    , change = NULL
                    , at = NULL
                    , offset = 1
                    , relative = F
                    , iterations = 1000
                    , seed = 10
                    , samples = F) {
```

```
  # Prepare two dataframes for different scenarios in marginal
  # effect computation
  d0 = d1 = data
```

```
  # Adjust the 'term' in the data based on the 'change' parameter
  if (is.null(change)) {
```

```
    # If change is NULL, adjust the term for numerical derivative
    # computation
    d0[[term]] = d0[[term]] - setstep(d0[[term]])
    d1[[term]] = d1[[term]] + setstep(d1[[term]])
```

```
  }
```

```
  # If change has an additive component, adjust the term accordingly
```

```

if (grepl("\\\\+", paste(change, collapse = "_"))) {

  d1[[term]] = d1[[term]] + as.numeric(gsub("\\\\+", "", change))

}

# If change is explicit with two values, set the term values
# directly
if (length(change) == 2) {

  d0[[term]] = as.numeric(change[1])
  d1[[term]] = as.numeric(change[2])

}

# If 'at' is specified, set predictor values in the data to these
# fixed values
# (allows the function to calculate the marginal effects at the
# specified values)
if (!is.null(at)) {
  for (i in names(at))
    d0[[i]] = at[[i]]
    d1[[i]] = at[[i]]
}

# Create matrices for prediction based on the model
Xp0 <- predict(mod, newdata = d0, type="lpmatrix")
Xp1 <- predict(mod, newdata = d1, type="lpmatrix")

# Extract model parameters
ilink <- family(mod)$linkinv
beta <- coef(mod)
vc <- mod$Vc # covariance matrix

# Compute marginal effects based on the adjusted data
pred0 <- 1 - (1-ilink(Xp0 %*% beta))^offset
pred1 <- 1 - (1-ilink(Xp1 %*% beta))^offset
ME <- (pred1-pred0)

# Adjust for numerical derivative if change is NULL

```

```

if (is.null(change)) {
  ME <- ME/(d1[[term]] - d0[[term]])
}

# convert to relative if requested
if (relative == T) ME = ME/pred0

# average marginal effect
AME = mean(ME)

# Simulate AMEs to compute uncertainty in the estimates

# Compute the variance of the average marginal effect through a
# "posterior" simulation.
# This involves simulating from a multivariate normal distribution
# using the model's
# coefficient means and covariance matrix

if (!is.null(seed)) set.seed(seed)
coefmat = mvrnorm(n = iterations
                  , mu = beta
                  , Sigma = vc)

# For each simulated coefficient vector, estimate the Average
# Marginal Effect (AME).
AMEs = apply(coefmat, 1, function(coefrow) {

  # Calculate marginal effects based on the simulated coefficient
  pred0 <- 1 - (1-ilink(Xp0 %*% coefrow))^offset
  pred1 <- 1 - (1-ilink(Xp1 %*% coefrow))^offset
  ME <- (pred1-pred0)

  # if change is NULL, use numerical derivative
  if (is.null(change)) {
    ME <- ME/(d1[[term]] - d0[[term]])
  }

  # convert to relative if requested
  if (relative == T) ME = ME/pred0

```

```

    # average marginal effect
    AME = mean(ME)
    return(AME)
  })

# Combine results
# If the 'samples' flag is FALSE, return the summary results.
# Otherwise, return both the summary and the sample results.

if (!samples) {
  res = data.frame(term
    , estimate = AME
    , std.error = sqrt(var(AMEs))
    , estimate.sim = mean(AMEs)
    , offset
    , change.value = paste(change, collapse = "_"))

  return(res)
} else {

  res_sums = data.frame(term
    , estimate = AME
    , std.error = sqrt(var(AMEs))
    , offset
    , change.value = paste(change, collapse = "_"))

  res_samples = data.frame(term
    , estimate = AMEs
    , MLE = AME
    , offset
    , change.value = paste(change,
                           collapse = "_"))

  res = list(res_sums, res_samples)
  return(res)
}
}

```


3.9.3 Calculating Absolute Average Marginal Effect (AMEs)

At the end of this code segment, the **AME** data frame contains average marginal estimates for each predictor, each corresponding to different change scenarios. In essence, AME provides the average effect that a predictor has on the outcome across these scenarios. On the other hand, the **AMEsamples** data frame contains multiple samples of AME for each predictor, each aligned with a specific type of change scenario, which allows for an evaluation of the uncertainty inherent in the AME estimates.

```
# Absolute AMEs -----

# Initialize empty data frames to store the results
AME = data.frame()
AMEsamples = data.frame()

# Loop through predictor names that match "con_"
for (i in names(predictors)[grepl("con_", names(predictors))]) {

  # Loop through different change settings (e.g., equilibrium, invasion,
  # iqr)
  for (j in names(change)) {

    # Calculate the AME for each model in res_mod
    temp = lapply(res_mod, function(x){

      # If the change is based on IQR (interquartile range), calculate the
      # 1st and 3rd quartiles
      if (j == "iqr") {
        q1 = quantile(x$model$con_dens, probs = 0.25)
        q3 = quantile(x$model$con_dens, probs = 0.75)
      }

      # Use the get_AME function to calculate the AME for the current model
      get_AME(x
        , data = x$model
        , offset = interval
        , term = i
        , change = eval(parse(text = change[[j]][,i]))
        , iterations = iter
        , samples = T
      )
    })
  }
}
```

```

)

# AME
tempAME = lapply(temp, function(x) x[[1]])
tempAME = Map(cbind, tempAME, change = j, sp = names(tempAME))
tempAME = do.call(rbind, tempAME)
AME = rbind(AME, tempAME)

# AME samples
tempSamples = lapply(temp, function(x) x[[2]])
tempSamples = Map(cbind, tempSamples, change = j,
                  sp = names(tempSamples), iter = iter)
tempSamples = do.call(rbind, tempSamples)
AMESamples = rbind(AMESamples, tempSamples)
}
}
head(AME)

```

	term	estimate	std.error	offset	change.value	change	sp
ACALDI	con_dens	0.017532523	0.016546708	1	+ 1	equilibrium	ACALDI
AEGICE	con_dens	0.089367793	0.025930769	1	+ 1	equilibrium	AEGICE
BEILPE	con_dens	0.006043419	0.001062937	1	+ 1	equilibrium	BEILPE
CAPPFR	con_dens	0.020944544	0.004310472	1	+ 1	equilibrium	CAPPFR
CECRIN	con_dens	0.028621858	0.027053981	1	+ 1	equilibrium	CECRIN
CORDLA	con_dens	0.013464775	0.030623821	1	+ 1	equilibrium	CORDLA

3.9.4 Calculating Relative Average Marginal Effect (rAMEs)

At the end of this code segment, the **rAME** data frame contains the **rAME** estimates for the predictor and each type of change, and the **rAMESamples** data frame contains the **rAME** samples for each predictor and type of change.

```

# Relative rAMEs -----

# Initialize empty data frames to store the results
rAME = data.frame()
rAMESamples = data.frame()

# Loop through predictor names that match "con_"

```

```

for (i in names(predictors)[grepl("con_", names(predictors))]) {

# Loop through different change settings
# (e.g., equilibrium, invasion, iqr)
for (j in names(change)) {

# Calculate the relative AME (rAME) for each model in res_mod
temp = lapply(res_mod, function(x){

# If the change is based on IQR (interquartile range),
# calculate the 1st and 3rd quartiles
if (j == "iqr") {
  q1 = quantile(x$model$con_dens, probs = 0.25)
  q3 = quantile(x$model$con_dens, probs = 0.75)
}

# Use the get_AME function to calculate the rAME for the
# current model, setting the 'relative' argument to TRUE
get_AME(x
  , data = x$model
  , offset = interval
  , term = i
  , change = eval(parse(text = change[[j]][, i]))
  , iterations = iter
  , relative = T
  , samples = T
)
}
)

# rAME
tempAME = lapply(temp, function(x) x[[1]])
tempAME = Map(cbind, tempAME, change = j, sp = names(tempAME))
tempAME = do.call(rbind, tempAME)
rAME = rbind(rAME, tempAME)

# rAME samples
tempSamples = lapply(temp, function(x) x[[2]])
tempSamples = Map(cbind, tempSamples, change = j,
  sp = names(tempSamples), iter = iter)
tempSamples = do.call(rbind, tempSamples)
rAMESamples = rbind(rAMESamples, tempSamples)

```

```

    }
  }
  head(rAME)

```

	term	estimate	std.error	offset	change.value	change	sp
ACALDI	con_dens	0.12190548	0.122096554	1	+ 1	equilibrium	ACALDI
AEGICE	con_dens	0.46332386	0.140120748	1	+ 1	equilibrium	AEGICE
BEILPE	con_dens	0.02026319	0.003624488	1	+ 1	equilibrium	BEILPE
CAPPFR	con_dens	0.42931069	0.087277501	1	+ 1	equilibrium	CAPPFR
CECRIN	con_dens	0.04121409	0.042571758	1	+ 1	equilibrium	CECRIN
CORDLA	con_dens	0.10440629	0.241547627	1	+ 1	equilibrium	CORDLA

3.10 Saving results

```

# Save results -----
save(list = c("AME", "AMEsamples", "rAME", "rAMEsamples", "nsp",
              "coefs", "sums") # "nsp_data_deficient"
      , file = paste0( "./data/mortality.Rdata"))
write.csv(AME, "./data/AME.csv")
write.csv(rAME, "./data/rAME.csv")

```

4 Using Meta-regressions to Compare CDD across Species or Sites

4.1 Overview

Following from *Section 4. How does CDD vary across species, abiotic gradients or in time?* in the main text, here, we demonstrate one approach to comparing the strength of CDD across species using a meta-analysis framework. The same approach can be used to compare CDD across sites, plots, or any other unit of interest, as long as it is possible to generate reliable estimates of the strength of CDD (suitable sample sizes, etc.). As noted in the main text, *“Correct propagation of uncertainty in CDD estimates requires meta-regressions in frequentist or Bayesian frameworks. Weighted regressions (e.g., lm, lmer, gam) can also estimate how CDD varies e.g., across latitude or between species with different life-history strategies, but incorrectly estimate the associated uncertainty.”*

We use a subset of the BCI seedling data (Comita et al. 2023) of only 30 species to compare how species abundance is related to the strength of CDD at the species level. This is the same dataset used in section 2. This analysis is for demonstration purposes only, and biological conclusions should not be made about the results, given this is only a small subset of the data.

An advantage of using meta-regressions over simple weighted regressions is that the models are able to simultaneously account for uncertainty in species-specific CDD estimates as well as systematic differences in species' CDD when regressed against a predictor via the inclusion of random effects. A simple weighted regression, on the other hand, assumes that there is only one error for which the relative strength is known when regressing the estimates against a predictor. The latter can lead to incorrect weighting of species in the metaregression.

In this tutorial, we use relative average marginal effect () calculated in the previous chapter as our response variable, calculated separately for each species. in this case estimates the relative increase in the probability of annual mortality with the addition of one new conspecific neighbor, while keeping total densities constant. Positive numbers indicate a relative increase in mortality with an increase in conspecific density, a signature of NCDD. In principle, any metric of the strength of CDD can be used, though care must be taken to ensure that the metrics are comparable across species and sites (see main text for more information).

We use the popular [metafor package](#) to fit the meta-regression models.

i Note

The following code is adapted from the [latitudinalCNDD repository](#) by [Lisa Hülsmann](#).

4.2 Load libraries and data

```
# Load libraries
library(tidyr)
library(dplyr)
library(readr)
library(ggplot2)
library(here)
library(metafor)

# Load in species abundances for BCI data subset
abundances <- read_csv(
  here("./data/BCI seedling data - 30 species - abundance 2023_05_18.csv")
)

# Load marginal effects calculations from previous section
load(here("./data/mortality.Rdata"))

# Subset down to just equilibrium change
rAME <- rAME %>%
  filter(change == "equilibrium")

# Join marginal effects and abundance data
rAME <- left_join(rAME, abundances, by = c("sp" = "spp"))

# Add in average abundance for 'rare' species
rAME$abundance[rAME$sp == "data_deficient_seedling"] <-
  mean(abundances$abundance[abundances$spp %in%
    nsp$spp[nsp$data_deficient]])

# Log transform abundance to use in models
rAME$log_abundance <- log(rAME$abundance)
```

Let's take a quick look at the data set we'll be working with:

```
head(rAME, n = 10)
```

	term	estimate	std.error	offset	change.value	change	sp
1	con_dens	0.12190548	0.122096554	1	+ 1	equilibrium	ACALDI
2	con_dens	0.46332386	0.140120748	1	+ 1	equilibrium	AEGICE
3	con_dens	0.02026319	0.003624488	1	+ 1	equilibrium	BEILPE
4	con_dens	0.42931069	0.087277501	1	+ 1	equilibrium	CAPPFR
5	con_dens	0.04121409	0.042571758	1	+ 1	equilibrium	CECRIN
6	con_dens	0.10440629	0.241547627	1	+ 1	equilibrium	CORDLA
7	con_dens	0.11658356	0.200294935	1	+ 1	equilibrium	DAVINI
8	con_dens	0.01113154	0.123539647	1	+ 1	equilibrium	DESMAX
9	con_dens	0.23375216	0.227252222	1	+ 1	equilibrium	HEISCO
10	con_dens	-0.06498043	0.061086207	1	+ 1	equilibrium	JUSTGR
	abundance	log_abundance					
1	169	5.129899					
2	153	5.030438					
3	5574	8.625868					
4	1604	7.380256					
5	51	3.931826					
6	248	5.513429					
7	27	3.295837					
8	55	4.007333					
9	56	4.025352					
10	136	4.912655					

4.3 Reformat data for model fitting

First, we use the ‘escalc’ function in the metafor package to essentially repackaging our data frame into a format used in the meta-regression model fitting. Since we already calculated our effect size (), we just pass through the estimate and corresponding standard error using the ‘GEN’ option for the ‘measure’ argument, rather than calculating an effect size within the ‘escalc’ function.

```
# Reformat data for model fitting
# Set measure to generic, which passes the observed effect sizes or
# outcomes via the yi argument and the corresponding sampling
# variances via the vi argument (or the standard errors via the sei
# argument) to the function.
```

```
dat_meta = metafor::escalc(measure = "GEN",
                           yi = estimate, # observed outcomes
                           sei = std.error, # standard errors
                           slab = sp, # label for species
                           data = rAME)
```

4.4 Fit meta-regression model

Next, we use the ‘rma’ function to fit a meta-regression model, where `yi` is our response variable (renamed as `yi` in the previous step) and log species abundance is our predictor. While not shown here, it is possible to fit mixed effects meta-regression models with the ‘rma.mv’ function. We suggest consulting the [extensive documentation for the metafor package](#) for further details.

```
# Fit model
metamod = metafor::rma(yi = yi,
                      vi = vi,
                      mods = ~ log_abundance,
                      method = "REML",
                      data = dat_meta)
```

4.5 Print model summary

```
summary(metamod)
```

Mixed-Effects Model (k = 22; tau² estimator: REML)

logLik	deviance	AIC	BIC	AICc
2.2364	-4.4727	1.5273	4.5145	3.0273

```
tau^2 (estimated amount of residual heterogeneity):    0.0166 (SE = 0.0090)
tau (square root of estimated tau^2 value):           0.1287
I^2 (residual heterogeneity / unaccounted variability): 91.39%
H^2 (unaccounted variability / sampling variability):  11.62
```


R² (amount of heterogeneity accounted for): 0.00%

Test for Residual Heterogeneity:

QE(df = 20) = 74.0420, p-val < .0001

Test of Moderators (coefficient 2):

QM(df = 1) = 0.3751, p-val = 0.5402

Model Results:

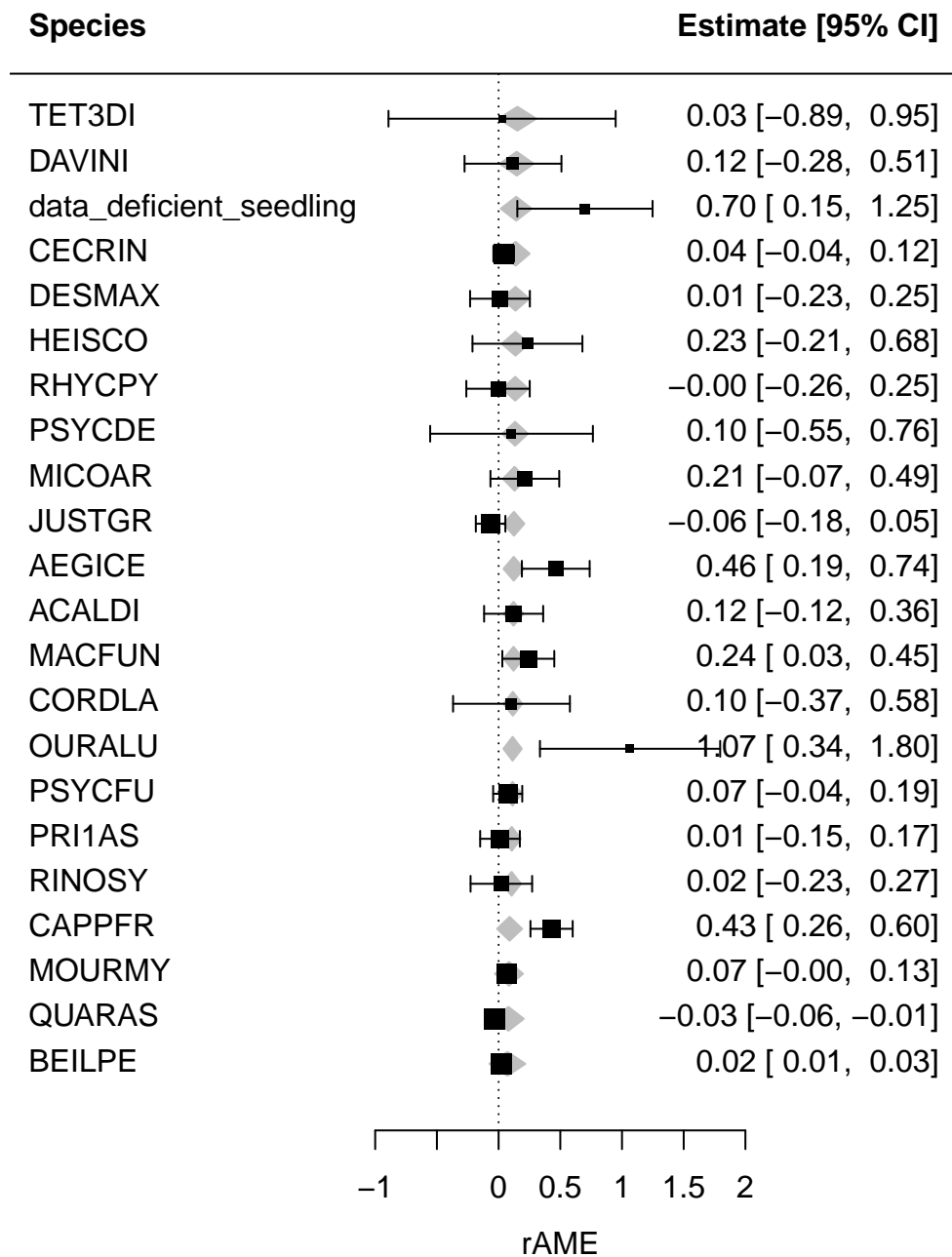
	estimate	se	zval	pval	ci.lb	ci.ub
intrcpt	0.1935	0.1401	1.3815	0.1671	-0.0810	0.4681
log_abundance	-0.0141	0.0231	-0.6125	0.5402	-0.0594	0.0311

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

4.6 Plot the estimated of rAME for all species with a forest plot

In this case, a forest plot shows the estimates of the strength of CDD for individual species, here ordered by least to most abundant going from top to bottom.

```
forest(metamod,
      header = "Species",
      xlab = "rAME",
      order = log_abundance)
```

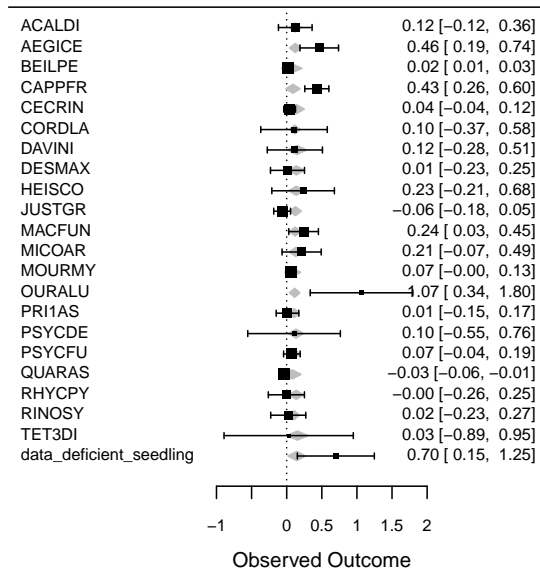


4.7 Model diagnostics

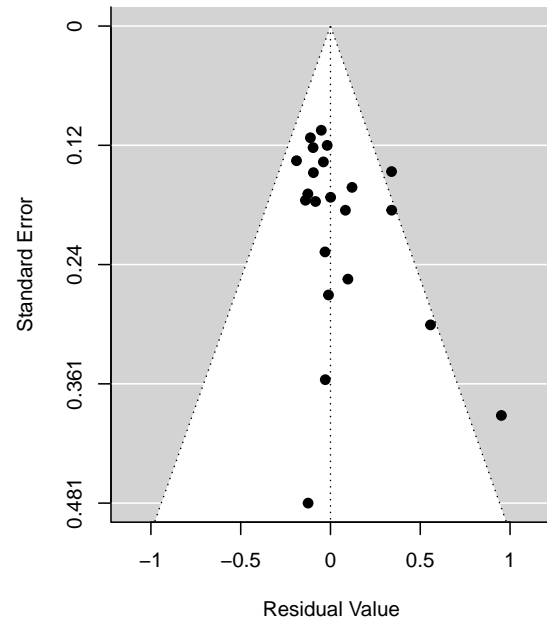
The `plot` method displays model diagnostics of the meta-regression model ([more info here](#)) in addition to a forest plot.

```
plot(metamod)
```

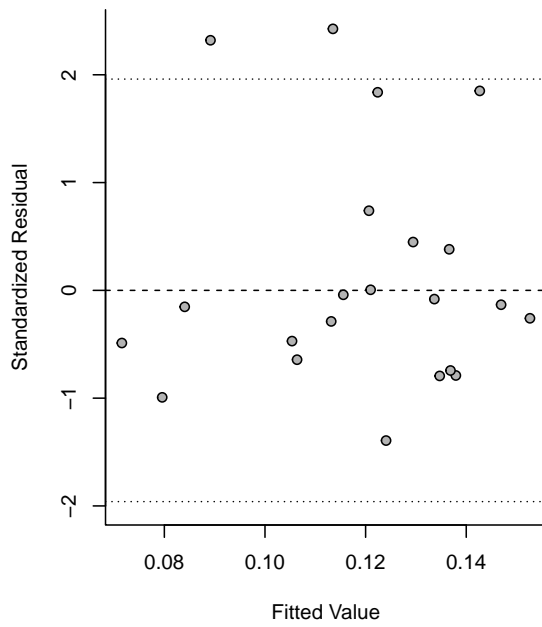
Forest Plot



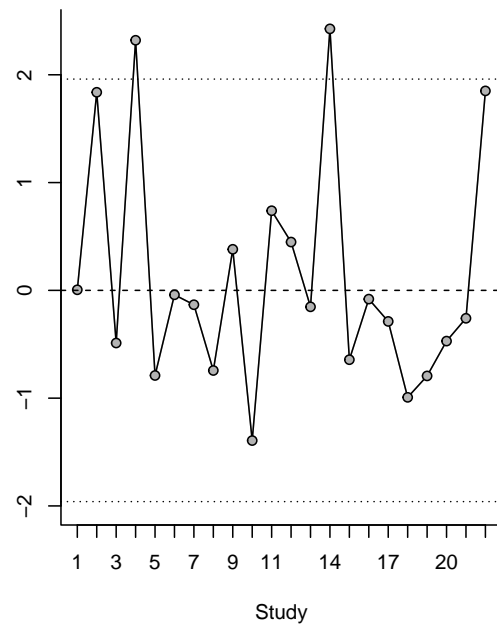
Residual Funnel Plot



Fitted vs. Standardized Residuals



Standardized Residuals



4.8 Model predictions of how species abundance is related to strength of CDD

Here, we generate predictions and corresponding confidence intervals for how our predictor of interest is related to the strength of CDD using the ‘predict’ function. The y-axis here indicates the relative increase in annual mortality probability with the addition of one conspecific neighbor. Higher values indicate stronger negative conspecific density dependence. In this example, our predictor of interest is species abundance. We also scale the size of the points based on their weight in the meta-analysis, with larger points indicating higher weights. Note that this analysis is for demonstration purposes only, and biological conclusions should not be made about the results, given this is only a small subset of the data.

```
# Generate a prediction dataframe
pred <- expand_grid(log_abundance = seq(min(dat_meta$log_abundance,
                                         na.rm = TRUE),
                                         max(dat_meta$log_abundance,
                                         na.rm = TRUE),
                                         length.out = 50))

pred$abundance <- exp(pred$log_abundance) # Back transform abundance

# Bind predictions to dataframe
pred <- cbind(pred, predict(object = metamod,
                           newmods = pred$log_abundance))

# Extract observed values
observed_values <- broom::augment(metamod)

# Add in variance estimates to be able to scale size of points by
# amount of variance in estimate
observed_values <- left_join(observed_values,
                             dat_meta %>%
                               dplyr::select(sp, vi, log_abundance),
                             by = c(".rownames" = "sp"))

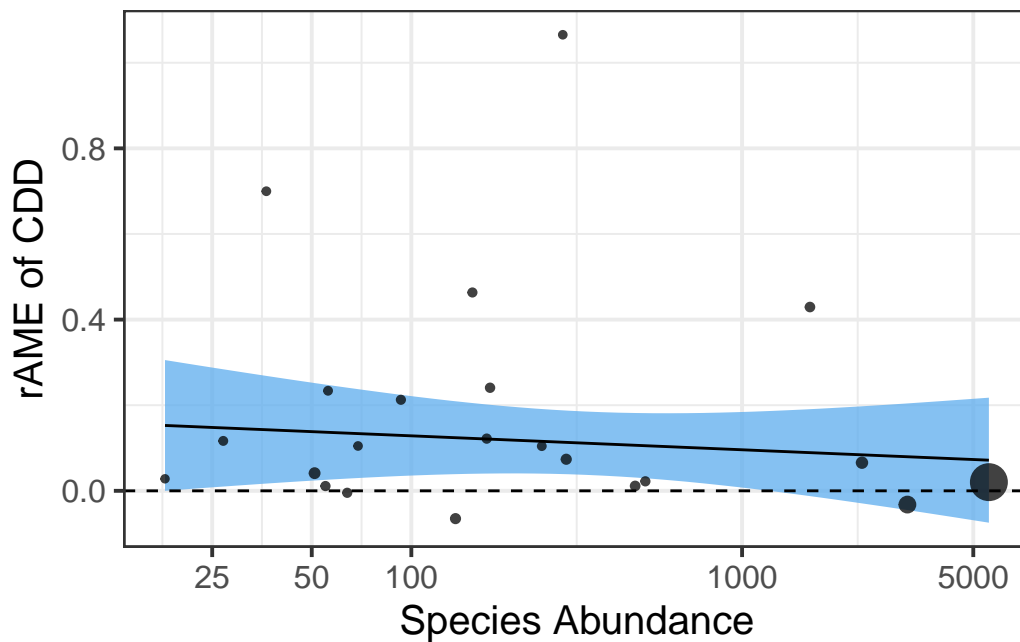
# Set abundance values for x axis
abundances_x_axis <- c(25, 50, 100, 1000, 5000)

# Plot prediction
```

```

ggplot(pred, aes(x = log_abundance, y = pred)) +
  geom_ribbon(aes(ymin = ci.lb, ymax = ci.ub),
            fill = "steelblue2", alpha = 0.75) +
  geom_line() +
  geom_hline(yintercept = 0, lty = 2) +
  labs(x = "Species Abundance", y = "rAME of CDD") +
  scale_x_continuous(breaks = log(abundances_x_axis),
                    labels = abundances_x_axis) +
  # Add observed points
  geom_point(data = observed_values,
            aes(x = log_abundance, y = .observed, size = 1/vi),
            alpha = 0.75) +
  theme_bw(15) +
  theme(legend.position = "none")

```



In principle, the approach outlined in this appendix of estimating marginal effects followed by a meta-regression analysis can be used to test for a correlation between strength of CDD and other species-level traits (*e.g.*, wood density, leaf area, etc) or to determine how site-level CDD varies with site-level variables, such as latitude (Hülsmann et al. 202x), precipitation, soil fertility, etc.

References

- Barber, Cristina, Andrii Zaiats, Cara Applestein, Lisa Rosenthal, and T Trevor Caughlin. 2022. “Bayesian Models for Spatially Explicit Interactions Between Neighbouring Plants.” *Methods in Ecology and Evolution*.
- Broekman, Maarten JE, Helene C Muller-Landau, Marco D Visser, Eelke Jongejans, SJ Wright, and Hans de Kroon. 2019. “Signs of Stabilisation and Stable Coexistence.” *Ecology Letters* 22 (11): 1957–75.
- Canham, Charles D, Michael J Papaik, María Uriarte, William H McWilliams, Jennifer C Jenkins, and Mark J Twery. 2006. “Neighborhood Analyses of Canopy Tree Competition Along Environmental Gradients in New England Forests.” *Ecological Applications* 16 (2): 540–54.
- Comita, Liza S, Salomón Aguilar, Stephen P Hubbell, and Rolando Pérez. 2023. “Long-Term Seedling and Small Sapling Census Data from the b Arro c Olorado i Sland 50 Ha f Orest d Ynamics p Lot, p Anama.” *Ecology* 104 (9): e4140.
- Currie, Iain D. 2016. “On Fitting Generalized Linear and Non-Linear Models of Mortality.” *Scandinavian Actuarial Journal* 2016 (4): 356–83.
- Uriarte, María, Nathan G Swenson, Robin L Chazdon, Liza S Comita, W John Kress, David Erickson, Jimena Forero-Montaña, Jess K Zimmerman, and Jill Thompson. 2010. “Trait Similarity, Shared Ancestry and the Structure of Neighbourhood Interactions in a Sub-tropical Wet Forest: Implications for Community Assembly.” *Ecology Letters* 13 (12): 1503–14.