Supplementary Information

Relaxation of the separation between ecological and evolutionary timescales has unexpected effects on community assembly of species with complex life cycles.

Izabel Eriksson

R-Code

The code is divided into five parts for easier handling. The first two parts are the functions used during simulations. The third part is how I define my resource distributions and parameters and the fourth part and example simulation. The last part is how I would plot the simulation example. I run simulations in many different iterations during my thesis, with different parameters and replicates and different plotting techniques. If you are interested in recreating something that is not in the simulation example please consult my thesis for the different change needed in resource distribution or parameter values.

I ran these simulations using R studio in R Statistical Software (R Core Team, 2023, version 4.3.2). The packages used where:

```
library(job)
library(tidyverse)
library(viridisLite)
library(viridis)
library(ggplot2)
library(extrafont)
library(patchwork)
library(dplyr)
library(FamilyRank)
```

(Lindeløv, 2021; Garnier et al., 2023; Wickham, 2016; Wickham et al., 2019; Wickham, 2016; Chang, 2023; Wickham et al., 2023; Saul, 2021; Pedersen, 2024).

Resource competition functions

This is the the actual simulation algorithm where competition and evolution occurs.

```
# Resource competition Models:

# Simple life cycle -------------


resourceCompetitionSLC <- function(popSize, resProp, resFreq, resGen=matrix(
   c(0.2,0.2),ncol=1, nrow=2), im = 0,
                                   fmax = 2, kA = 0.5, kJ = 0.5, mutProb
                                      =0.0005, mutVar=0.05, time.steps
                                      =50000, iniP=0,
                                   threshold = 0.0005, nmorphs = 1, maxTr =
                                      3, minTr = -3){
```

```
pop <- matrix(data = NA, ncol = 3, nrow = nmorphs)
                              # Each column in this matrix is one species.

pop[,1] <- popSize
                                                                  # Number of
   individuals per species is first row.
pop[,2] <- iniP
                                                                    #
   Phenotype for species is second row.


colnames(pop) <- c("Number␣of␣indivduals", "Trait", "Proxy")
                     # Third row is where f and m are stored to calculate
   fecundity and maturation.

stats          <- matrix(data = c(0, sum(pop[,1]), 0, nrow(pop), mean(pop
   [,2]), var(pop[,2]))
                         , nrow = 1, ncol = 6)
                                                             # Where we will
                            eventually save our data on number of species
                           etc
phenotypes <- matrix(data = c(0, popSize, iniP), nrow = 1, ncol = 3)
            # Where information on each different species is stored
colnames(phenotypes) <- c("Year", "Number␣of␣indivduals", "Trait")

epsilon <- .Machine$double.eps^10
                                                  # Added when there is risk
   of r rounding a number down to 0, very small number

for (t in 1:time.steps){

  # Deterministic fecundity proxy alpha
     ------------------------------------

  adults    <- pop
                                                                  # The
     adults matrix is only created so it easier to mentally seperate adult
      and juvenile step, also if the adult population wants to be
     extracted at end of timestep
  alphaA    <- NULL
                                                                # Will
     create a matrix with all alpha values, is nullified so previous time
     step is overwritten
  alphaSumA <- NULL


  resPropAduMatrix <- matrix(data = resProp, ncol = length(resProp),
              # Resource property is made into a matrix here so that
     matrix calculatsion can be used.
                             nrow = nrow(adults), byrow = T)
  aduTrait <- adults[, 2]
```

```
aduTraitMatrix <- matrix(data = rep(aduTrait, each = length(resProp)),
        # Made into matrix so matrix calculcations can be used
                        ncol = length(resProp), nrow = nrow(adults),
                        byrow = T)

alphaA           <- (1/(sqrt(2*pi*resGen[1,1]^2)))*exp(-(((
   aduTraitMatrix-resPropAduMatrix)^2)/(2*resGen[1,1])^2)) + epsilon
                  # Calculation of individual alpha values, equation 2
adultAbund       <- adults[,1]
adultAbundMatrix <- matrix(data = rep(adultAbund, each = length(resProp)
   ), ncol = length(resProp), nrow = nrow(adults), byrow = T)  #
   Creation of a matrix with population size of each type in the rows
alphaSumA        <- colSums((alphaA*adultAbundMatrix))
                      # Creates the denominator for equation 3. Which
   is each INDIVIDUALS alphaA added together for each resource type, so
   each column is one resource, each row is one species.


RdivAlphaSumA       <- resFreq/alphaSumA
RdivAlphaSumATrans  <- matrix(data = RdivAlphaSumA)
                         # Is transformed so that matrix caluclations
   can be done

Fec <- alphaA%*%RdivAlphaSumATrans
                                            # The result is a vector
   with each species total energy consumption, equation 4.
adults[,3] <- fmax*(Fec/(kA+Fec))
                                             # Gives f (fecundity)
   equation 5a



# Spawning of offspring
   ------------------------------------------------

juveniles <- adults
                                                          # Create a
   matrix were we will add juveniles into

juveniles[,1] <- rpois(n = nrow(juveniles),
                                  # The number of juveniles is drawn
   from a poisson distribution with mean f x species abundance
                       lambda = juveniles[,1]*juveniles[,3])
                                     # Since each individual of a
                          species has offspring

juvenile.pop <- c()
juvenile.pop <- sum(juveniles[,1])
                                            # To extract number of
   juveniles if that wants to be analysed
```

```
# Mutation of offspring
   ------------------------------------------------

probs <- juveniles[,1]/sum(juveniles[, 1])    # Generates probability of
    morph being mutated based upon number of individuals.
N.mut <- as.numeric(rbinom(n = 1, size = sum(juveniles[, 1]), prob =
   mutProb)) # Draws the number of mutations this generation


if(N.mut > 0){
  random.choice <- c()
  mutation.pos <- c()

  for (m in 1:length(N.mut)){
    random.choice <- rmultinom(n = 1 , size = 1, prob = probs)
                    # Randomly chooses which morphs to mutate based on
       probs
    mutation.pos[m] <- as.numeric(which(random.choice == 1))
  }

  for (i in  1:length(mutation.pos)){

    mutChange <- rnorm(n=1, mean=0, sd=mutVar)
    juveniles[mutation.pos[i], 1] <- juveniles[mutation.pos[i], 1] - 1
             # Removes the mutated individual from the morph

    new.morph <- matrix(data = c(1, juveniles[mutation.pos[i], 2] +
       mutChange, 0),  #Changes trait to a new trait and adds it to the
       juveniles
                        ncol = ncol(juveniles), nrow = 1)
    juveniles <- rbind(juveniles, new.morph)

  }
}



# Maturation off offspring
   ------------------------------------------------------

# The calculation of alphaJ is the same as the caluclation of alphaA see
    above comments for clarifications

alphaSumJ <- NULL
alphaJ    <- NULL

resPropJuvMatrix <- matrix(data = resProp, ncol = length(resProp), nrow
   = nrow(juveniles), byrow = T)
juvTrait <- juveniles[,2]
juvTraitMatrix <- matrix(data = rep(juvTrait, each = length(resProp)),
   ncol = length(resProp), nrow = nrow(juveniles), byrow = T)
```

```
alphaJ            <- (1/(sqrt(2*pi*resGen[2,1]^2)))*exp(-(((
   juvTraitMatrix-resPropJuvMatrix)^2)/(2*resGen[2,1]^2))) + epsilon
juvenAbund        <- juveniles[,1]
juvenAbundMatrix  <- matrix(data = rep(juvenAbund, each = length(resProp
   )), ncol = length(resProp), nrow = nrow(juveniles), byrow = T)
alphaSumJ         <- colSums(alphaJ*juvenAbundMatrix)


RdivAlphaSumJ     <- resFreq/alphaSumJ
RdivAlphaSumJTrans   <- matrix(data = RdivAlphaSumJ)


Sur <- alphaJ%*%RdivAlphaSumJTrans
juveniles[,3] <- (Sur/(kJ+Sur))
                                                  # Gives m, equation 5b




juveniles[,1] <- rbinom(n = nrow(juveniles) , size = juveniles[,1], prob
    = juveniles[,3])


pop <- juveniles[juveniles[, 1] != 0, , drop = FALSE]
                           # all adults die after reproducing, so the new
    generation is only juveniles, and all rows with zero individuals are
    removed.

# Adding immigrants
   ------------------------------------------------------------------------

# Only done when im = 1
if(im == 1) {

    trait <- runif(1, min = minTr, max = maxTr)
                                    # Creates phenotype of immigrant

    if(sum(pop[,2] == trait) == 0) {
                                                  # Checks whether a exact
       match of immigrant already exists
      pop <- rbind(pop, c(1, trait, NA))
                                                # If no duplicate is found,
        a new species is added to the pop matrix
    } else{
      same <- which(pop[,2] == trait)
      pop[same,1] <- pop[same,1]+1
                                                     # If so it is just
        added to that species
    }
  }
```

```
  # extract stats and phenotype
     ------------------------------------------

  if(nrow(pop) == 0){
                                                                  # Checks
     whether population has reached zero, then it breaks the for loop.
    print("Population␣extinction")
    break
  }

  if(sum(which(is.na(pop[,1])) != 0)){
                                                 # Checks whether population
     has reached zero, then it breaks the for loop.
    print("Population␣extinction")
    break
  }

  # Extracts the results from this generation into the stats and
     phenotypes matrices.

    stats <- rbind(stats, c(t, sum(adults[,1]), juvenile.pop, nrow(pop),
       mean(pop[,2]), var(pop[,2])))

    pStats <- cbind(rep(t, nrow(pop)), pop[,1], pop[,2])
    phenotypes <- rbind(phenotypes, pStats)


}

# Removing last time step

stats <- stats[stats[, 1] != time.steps, , drop = FALSE]
phenotypes <- phenotypes[phenotypes[,1] != time.steps, , drop = FALSE]

# Removing any morphs of very low abundance for last time step
pop <- pop[pop[, 1] > threshold*stats[nrow(stats), 2], , drop = FALSE]


# Re-adding modified last time step

stats <- rbind(stats, c(t, sum(adults[,1]), juvenile.pop, nrow(pop), mean(
   pop[,2]), var(pop[,2])))

pStats <- cbind(rep(t, nrow(pop)), pop[,1], pop[,2])
phenotypes <- rbind(phenotypes, pStats)

#return output
   ------------------------------------------------------------
colnames(stats) <- c("year", "Adult␣Population␣size", "Juvenile␣Population
   ␣Size", "Number␣of␣morphs", "mean␣trait", "var␣trait")
rownames(phenotypes) <- NULL
```

```
  return(list(stats=stats, phenotypes=phenotypes))
                                    #returns both the stats and the
     phenotype


}



# Complex life cycle -------------------------

# The simple life cycle function is more thoroughly commented, the same
    methods are used here, please consult above for explanations.
# Only code that differs is explained here.

resourceCompetitionCLC <- function(popSize, resProp, resFreq, resGen=matrix(
   c(0.2,0.2),ncol=1, nrow=2), fmax = 2,
                                   kA = 0.5, kJ = 0.5,mutProb=0.0005, mutVar
                                      =0.05, time.steps=50000, iniPA=0,
                                      iniPJ=0,
                                   threshold = 0.0005, nmorphs = 1, im = 0,
                                      maxTr = 3, minTr = -3){


  pop <- matrix(data = NA, ncol = 4, nrow = nmorphs)
                                  # Each column in this matrix is one
     phenotype combination.

  pop[,1] <- popSize
  pop[,2] <- iniPA
                                                                      # Adult
     trait is stored in second row and
  pop[,3] <- iniPJ
                                                                      # juvenile
      trait in third row.

  colnames(pop) <- c("Number␣of␣indivduals", "Adult␣trait", "Juvenile␣trait"
     , "Proxy")

  stats          <- matrix(data = c(0, sum(pop[,1]), 0, nrow(pop), mean(pop
     [,2]), var(pop[,2]),
                                   mean(pop[,3]), var(pop[,3])), nrow = 1,
                                      ncol = 8)

                                      #Where we will eventually save our
                                      stats and phenotypes
  phenotypes <- matrix(data = c(0, popSize, iniPA, iniPJ), nrow = 1, ncol =
     4)
  colnames(phenotypes) <- c("Year", "Number␣of␣indivduals", "Adult␣trait", "
     Juvenile␣trait")
```

```
epsilon <- .Machine$double.eps^10
                                                    # Added when some number
   become zero, very small number



for (t in 1:time.steps){
  # Deterministic fecundity proxy alpha
     ------------------------------------

  adults     <- pop
  alphaA     <- NULL
                                                                  # Will
     create a matrix with all alpha values
  alphaSumA  <- NULL


  resPropAduMatrix <- matrix(data = resProp[1,], ncol = ncol(resProp),
     nrow = nrow(adults), byrow = T)
  aduTrait <- adults[, 2]
  aduTraitMatrix <- matrix(data = rep(aduTrait, each = ncol(resProp)),
     ncol = ncol(resProp), nrow = nrow(adults), byrow = T)

  alphaA           <- (1/(sqrt(2*pi*resGen[1,1]^2)))*exp(-(((
     aduTraitMatrix-resPropAduMatrix)^2)/(2*resGen[1,1])^2)) + epsilon
                     # Calculation of individual alpha
  adultAbund       <- adults[,1]
  adultAbundMatrix <- matrix(data = rep(adultAbund, each = ncol(resProp)),
      ncol = ncol(resProp), nrow = nrow(adults), byrow = T)  # Creation of
      a matrix with population size of each type in the rows
  alphaSumA        <- colSums((alphaA*adultAbundMatrix))

     # Creation of matrix that reflects both the trait but also number of
     individuals in type


  RdivAlphaSumA       <- resFreq[1,]/alphaSumA
  RdivAlphaSumATrans  <- matrix(data = RdivAlphaSumA)

  Fec <- alphaA%*%RdivAlphaSumATrans
  adults[,4] <- fmax*(Fec/(kA+Fec))

  # Spawning of offspring
     ------------------------------------------------

  juveniles <- adults
                                                                  # Create a
     matrix were we will add juveniles into

  juveniles[,1] <- rpois(n = nrow(juveniles), lambda = juveniles[,1]*
     juveniles[,4])
```

```
juvenile.pop <- c()
juvenile.pop <- sum(juveniles[,1])   # To extract number of juveniles


# Mutation of offspring
   ------------------------------------------------

probs <- juveniles[,1]/sum(juveniles[, 1])    # Generates probability of
    morph being mutated based upon number of individuals.
N.mut <- as.numeric(rbinom(n = 1, size = sum(juveniles[, 1]), prob =
   mutProb))


if(N.mut > 0){
  random.choice <- c()
  mutation.pos <- c()

  for (m in 1:length(N.mut)){
    random.choice <- rmultinom(n = 1 , size = 1, prob = probs)
                    # Randomly chooses which morphs to mutate based on
       probs
    mutation.pos[m] <- as.numeric(which(random.choice == 1))
  }

  for (i in  1:length(mutation.pos)){

    mutChange <- rnorm(n=1, mean=0, sd=mutVar)
    juveniles[mutation.pos[i], 1] <- juveniles[mutation.pos[i], 1] - 1
            # Removes the mutated individual from the morph

    if(rbinom(n = 1, size = 1, prob = 0.5) == 0){
                                  # Randomly choose whether adult or
       juvenile trait gets morphed.


      new.morph <- matrix(data = c(1, juveniles[mutation.pos[i], 2] +
         mutChange, #Changes adult trait to a new trait and adds it to
         the juveniles
                                     juveniles[mutation.pos[i], 3], 0),
                          ncol = ncol(juveniles), nrow = 1)
      juveniles <- rbind(juveniles, new.morph)
    }
    else {

      new.morph <- matrix(data = c(1, juveniles[mutation.pos[i], 2] ,
                #Changes juvenile trait to a new trait and adds it to
         the juveniles
                                     juveniles[mutation.pos[i], 3]+
                                        mutChange, 0),
                          ncol = ncol(juveniles), nrow = 1)
      juveniles <- rbind(juveniles, new.morph)
```

```
    }

  }

}



# Maturation off offspring
   ----------------------------------------------------

alphaSumJ <- NULL
alphaJ    <- NULL
# Survival of juveniles also depends on resource availability
resPropJuvMatrix <- matrix(data = resProp[2,], ncol = ncol(resProp),
   nrow = nrow(juveniles), byrow = T)
juvTrait <- juveniles[,3]
juvTraitMatrix <- matrix(data = rep(juvTrait, each = ncol(resProp)),
   ncol = ncol(resProp), nrow = nrow(juveniles), byrow = T)

alphaJ            <- (1/(sqrt(2*pi*resGen[2,1]^2)))*exp(-(((
   juvTraitMatrix-resPropJuvMatrix)^2)/(2*resGen[2,1]^2))) + epsilon
juvenAbund        <- juveniles[,1]
juvenAbundMatrix  <- matrix(data = rep(juvenAbund, each = ncol(resProp))
   , ncol = ncol(resProp), nrow = nrow(juveniles), byrow = T)
alphaSumJ         <- colSums(alphaJ*juvenAbundMatrix)


RdivAlphaSumJ     <- resFreq[2,]/alphaSumJ
RdivAlphaSumJTrans   <- matrix(data = RdivAlphaSumJ)


Sur <- alphaJ%*%RdivAlphaSumJTrans
juveniles[,4] <- (Sur/(kJ+Sur))




juveniles[,1] <- rbinom(n = nrow(juveniles) , size = juveniles[,1], prob
    = juveniles[,4])

pop <- juveniles[juveniles[, 1] != 0, , drop = FALSE]
                          # all adults die after reproducing, so the new
    generation is only juveniles, and all rows with zero individuals are
    removed.

# Adding immigrants
   ----------------------------------------------------------------------



if(im == 1) {
```

```
    Atrait  <- runif(1, min = minTr, max = maxTr)
    Jtrait  <- runif(1, min = minTr, max = maxTr)

    if(sum(pop[,2] == Atrait & pop[,3] == Jtrait) == 0) {
                            # Checks whether a exact match of immigrant
       already exists, both traits are checked in complex
         pop <- rbind(pop, c(1, Atrait, Jtrait, NA))
        } else{
          same <- which(pop[,2] == Atrait & pop[,3] == Jtrait)
          pop[same,1] <- pop[same,1]+1
          }
    }




  # extract stats and phenotype
     ---------------------------------------------

  if(nrow(pop) == 0){
                                                                  # Checks
     whether population has reached zero, then it breaks the for loop.
    print("Population extinction")
    break
  }

  if(sum(which(is.na(pop[,1])) != 0)){
                                                # Checks whether population
     has reached zero, then it breaks the for loop.
    print("Population extinction")
    break
  }


  stats <- rbind(stats, c(t, sum(adults[,1]), juvenile.pop, nrow(pop),
     mean(pop[,2]), var(pop[,2]),
                            mean(pop[,3]), var(pop[,3])))

  pStats <- cbind(rep(t, nrow(pop)), pop[,1], pop[,2], pop[,3])
  phenotypes <- rbind(phenotypes, pStats)




}

#Removing last time step

stats <- stats[stats[, 1] != time.steps, , drop = FALSE]
phenotypes <- phenotypes[phenotypes[, 1] != time.steps, , drop = FALSE]
```

```
  # Removing any morphs of very low abundance for last time step
  pop <- pop[pop[, 1] > threshold*stats[nrow(stats), 2], , drop = FALSE]

  # Readding modified last time step
  stats <- rbind(stats, c(t, sum(adults[,1]), juvenile.pop, nrow(pop), mean(
     pop[,2]), var(pop[,2]),
                         mean(pop[,3]), var(pop[,3])))

  pStats <- cbind(rep(t, nrow(pop)), pop[,1], pop[,2], pop[,3])
  phenotypes <- rbind(phenotypes, pStats)

  #return output
     ------------------------------------------------------------
  colnames(stats) <- c("year", "Adult population size","Juvenile Population
     Size", "Number of morphs", "mean A trait", "var A", "mean J trait", "
     var J")
  rownames(phenotypes) <- NULL

  return(list(stats=stats, phenotypes=phenotypes))   #returns both the stats
      and the phenotype


}
```

Grouping functions

```
# Grouping functions
# These functions are used at the end of a simulation to ensure
# very closely clustered species
# are grouped into one.

# SLC -------------------------


slc.groups <- function(output = outputSLC, threshold = 0.15){
  outputSLC <- output

  # Prepare the data

  phenodataSLC <- data.frame(
  Year = outputSLC$phenotypes[, 1],
  Trait = outputSLC$phenotypes[, 3],
  Num_Individuals = outputSLC$phenotypes[, 2]
      )

  last_year_dataSLC <- phenodataSLC[phenodataSLC$Year == max(phenodataSLC$
     Year), ]


  last_year_dataS <- subset(last_year_dataSLC, select = -Year)
  last_year_dataS <- subset(last_year_dataS, select = -Num_Individuals)
```

```
rownames(last_year_dataS) <- NULL
rownames(last_year_dataSLC) <- NULL

# Creates a matrix where each entry indicates distances between two
   species.

distance_matrix <- as.matrix(dist(last_year_dataS[, 1, drop = FALSE],
   method = "euclidean"))

# Removes duplicates in lower diagnoal

distance_matrix[lower.tri(distance_matrix)] <- NA


if(sum(which(distance_matrix < threshold)) == 0){

  return(last_year_dataSLC)
} # Checks if there are zero individuals who are alike, then the function
   is stopped.


# Find indices of individuals to keep
same <- which(distance_matrix < threshold, arr.ind = T) # Creates groups
   of species that are clustered togerther


same <- same[same[, 1]-same[,2] != 0, , drop = FALSE]   # Removes the rows
    indicating a species itself is to similar to itself.
rownames(same) <- NULL


if(sum(same) == 0){
  return(last_year_dataSLC)
}  # Checks if there are zero individuals who are alike, then the function
    is stopped.



# Initialize an empty list to store groups
groups <- list()

# Function to find group index for a species
find_group <- function(species_id) {
  for (g in seq_along(groups)) {
    if (species_id %in% unlist(groups[[g]])) {
      return(g)
    }
  }
  return(0)
}

# Iterate over rows in the matrix
```

```
  for (s in 1:nrow(same)) {
    species1 <- same[s, 1]
    species2 <- same[s, 2]

    # Find groups for each species
    group1 <- find_group(species1)
    group2 <- find_group(species2)

    if (group1 == 0 & group2 == 0) {
      # Create a new group
      groups <- c(groups, list(c(species1, species2)))
    } else if (group1 == 0) {
      # Add species1 to the group containing species2
      groups[[group2]] <- c(groups[[group2]], species1)
    } else if (group2 == 0) {
      # Add species2 to the group containing species1
      groups[[group1]] <- c(groups[[group1]], species2)
    } else if (group1 != group2) {
      # Merge two groups
      groups[[group1]] <- c(groups[[group1]], groups[[group2]])
      groups <- groups[-group2]
    }
  }




  # Filter out duplicate species in each group
  groups <- lapply(groups, function(group) unique(group))

  rownames(last_year_dataSLC) <- NULL
  final_data <- last_year_dataSLC          # Place to store filtered data
  total.sub <- c()                       # Place to store subspecies

  #Add population count of "subspecies" to main species

  for(q in seq_along(groups)){
    combo <- NULL
    combo <- groups[[q]]
    main <- combo[which.max(final_data[combo,3])]
    sub <- combo[-which.max(final_data[combo,3])]
    final_data[main,3] <- final_data[main,3] + sum(final_data[sub,3])
    total.sub <- rbind(c(total.sub, sub))

  }
  # Remove subspecies
  final_data <- final_data[-total.sub, ,drop = FALSE]
  return(final_data)


}
```

```
# CLC ----------------------------------------------------------


clc.groups <- function(output = outputCLC, threshold = 0.15){

  outputCLC <- output

  # Prepare the data

  phenodataCLC <- data.frame(
    Year = outputCLC$phenotypes[, 1],
    Adult_Trait = outputCLC$phenotypes[, 3],
    Juvenile_Trait = outputCLC$phenotypes[, 4],
    Num_Individuals = outputCLC$phenotypes[, 2]
  )

  last_year_dataCLC <- phenodataCLC[phenodataCLC$Year == max(phenodataCLC$
     Year), ]
  last_year_dataC <- subset(last_year_dataCLC, select = -Year)
  last_year_dataC <- subset(last_year_dataC, select = -Num_Individuals)
  rownames(last_year_dataCLC) <- NULL
  rownames(last_year_dataC) <- NULL

  # Creates a matrix where each entry indicates distances between two
     species.

  distance_matrix_adult <- as.matrix(dist(last_year_dataC[, 1, drop = FALSE
     ], method = "euclidean"))
  distance_matrix_juvenile <- as.matrix(dist(last_year_dataC[, 2, drop =
     FALSE], method = "euclidean"))

  # The lower diagonal is removed as it is a duplicate

  distance_matrix_adult[lower.tri(distance_matrix_adult)] <- NA
  distance_matrix_juvenile[lower.tri(distance_matrix_juvenile)] <- NA




  if(sum(which(distance_matrix_adult < threshold & distance_matrix_juvenile
     < threshold)) == 0){
    return(last_year_dataCLC)
  }   # Checks if there are zero individuals who are alike.

  # Checks which individuals are closer than the threshold distance,
     indicating they are too similar.

  same <- which(distance_matrix_adult < threshold & distance_matrix_juvenile
      < threshold, arr.ind = T)
  same <- same[same[, 1]-same[,2] != 0, , drop = FALSE]  # Removes the rows
     indicating a species is to similar to itself.
  rownames(same) <- NULL
```

```
if(sum(same) == 0){
  return(last_year_dataCLC)
}
# Check is there are zero species that are the same and stops function if
   that is the case.



# Initialize an empty list to store groups
groups <- list()

# Function to find group index for a species
find_group <- function(species_id) {
  for (g in seq_along(groups)) {
    if (species_id %in% unlist(groups[[g]])) {
      return(g)
    }
  }
  return(0)
}

# Iterate over rows in the matrix
for (s in 1:nrow(same)) {
  species1 <- same[s, 1]
  species2 <- same[s, 2]

  # Find groups for each species
  group1 <- find_group(species1)
  group2 <- find_group(species2)

  if (group1 == 0 & group2 == 0) {
    # Create a new group
    groups <- c(groups, list(c(species1, species2)))
  } else if (group1 == 0) {
    # Add species1 to the group containing species2
    groups[[group2]] <- c(groups[[group2]], species1)
  } else if (group2 == 0) {
    # Add species2 to the group containing species1
    groups[[group1]] <- c(groups[[group1]], species2)
  } else if (group1 != group2) {
    # Merge two groups
    groups[[group1]] <- c(groups[[group1]], groups[[group2]])
    groups <- groups[-group2]
  }
}


# Filter out duplicate species in each group
groups <- lapply(groups, function(group) unique(group))

final_data <- last_year_dataCLC          # Place to store filtered data
```

```
  total.sub <- c()                    # Place to store subspecies

  #Add population count of "subspecies" to main species

  for(q in seq_along(groups)){
    combo <- NULL
    combo <- groups[[q]]
    main <- combo[which.max(final_data[combo,4])]
    sub <- combo[-which.max(final_data[combo,4])]
    final_data[main,4] <- final_data[main,4] + sum(final_data[sub,4])
    total.sub <- rbind(c(total.sub, sub))

  }
  # Remove subspecies
  final_data <- final_data[-total.sub, , drop = FALSE]

  return(final_data)
}
```

Resource and Parameter initialization

```
# Resource initializations -------------------


Num.Res <- 16                   # Number of resources
res.Abund <-  50000             # Abundance of resources

# Evenly distributed Resources

# SLC:
resource.freq.even.slc <- rep(1/Num.Res, times = Num.Res)
                          # res. freq.
resource.prop.even.slc <- c(seq(from = -2.5, to = 2.5, length.out = Num.Res)
   )   # res. property
resource.freq.even.slc <- res.Abund*resource.freq.even.slc


# CLC:

resource.property.even.clc <- c(seq(from = -2.5, to = 2.5, length.out = Num.
   Res))

resource.frequency.even.clc <- rep(1/Num.Res, times = Num.Res)

resource.abundance.adults.even.clc     <- res.Abund
                                # res. abundance of adults and juveniles
resource.abundance.juveniles.even.clc  <- res.Abund

resFreqMatrix.even.clc <- matrix(resource.frequency.even.clc, nrow=2, ncol=
   length(resource.frequency.even.clc ), byrow = TRUE)
resFreqMatrix.even.clc [1, ] <- resFreqMatrix.even.clc [1, ]*resource.
   abundance.adults.even.clc
```

```
resFreqMatrix.even.clc [2, ] <- resFreqMatrix.even.clc [2, ]*resource.
   abundance.juveniles.even.clc

resPropMatrix.even.clc <- matrix(resource.property.even.clc, nrow=2, ncol=
   length(resource.property.even.clc ), byrow = TRUE)

rownames(resFreqMatrix.even.clc) <- c("Adult", "Juvenile")
colnames(resFreqMatrix.even.clc)  <- paste0("Resource␣", 1:ncol(
   resFreqMatrix.even.clc))

rownames(resPropMatrix.even.clc)<-c("Adult", "Juvenile")
colnames(resPropMatrix.even.clc)  <- paste0("Resource␣", 1:ncol(
   resPropMatrix.even.clc))

# Normal resources:

m <- 0     # mean
s <- 1     # standard deviation
N.resource.frequency <- c()
N.resource.property<- c(seq(from = -2.5, to = 2.5, length.out = Num.Res))

mid.add <- c()
midpoint <- c()

# Loop to create an approximation of normal distribution for discrete
   resources.
for(i in 1:(length(N.resource.property))){
  mid.add <- (N.resource.property[i+1]-N.resource.property[i])/2
  high.midpoint <- N.resource.property[(i)]+mid.add
  low.midpoint <- N.resource.property[(i)]-mid.add
  if(i == 1){
    N.resource.frequency[i] <- pnorm(high.midpoint, mean = m, sd = s)
  }else if(i == length(N.resource.property)){
    low.midpoint <- N.resource.property[(i-1)] + (N.resource.property[i]-N.
       resource.property[i-1])/2
    N.resource.frequency[i] <- pnorm(low.midpoint, mean = m, sd = s, lower.
       tail = FALSE)
  }else{
    N.resource.frequency[i] <- pnorm(high.midpoint, mean = m, sd = s) -
       pnorm(low.midpoint, mean = m, sd = s)
  }
}



resource.abundance.adults.norm.clc      <- res.Abund
resource.abundance.juveniles.norm.clc   <- res.Abund

# SLC:

resource.prop.norm.slc  <- N.resource.property
resource.freq.norm.slc  <- res.Abund*N.resource.frequency
```

```
# CLC:

resFreqMatrix.norm.clc  <- matrix(N.resource.frequency, nrow=2, ncol=length(
   N.resource.frequency), byrow = TRUE)

resFreqMatrix.norm.clc [1, ] <- resFreqMatrix.norm.clc [1, ]*resource.
   abundance.adults.norm.clc
resFreqMatrix.norm.clc [2, ] <- resFreqMatrix.norm.clc [2, ]*resource.
   abundance.juveniles.norm.clc

rownames(resFreqMatrix.norm.clc ) <- c("Adult", "Juvenile")
colnames(resFreqMatrix.norm.clc )  <- paste0("Resource␣", 1:ncol(
   resFreqMatrix.norm.clc ))


resPropMatrix.norm.clc  <- matrix(N.resource.property, nrow=2, ncol=length(N
   .resource.property), byrow = TRUE)


rownames(resPropMatrix.norm.clc )<-c("Adult", "Juvenile")
colnames(resPropMatrix.norm.clc )  <- paste0("Resource␣", 1:ncol(
   resPropMatrix.norm.clc))


# Skewed resource distribution

# SLC

tot <- (Num.Res*(Num.Res+1))/2

x <- 1/tot
resource.freq.skew.slc <- c()

for (i in 1:Num.Res){
  resource.freq.skew.slc[i] <- i*x
}

resource.prop.skew.slc <- c(seq(from = -2.5, to = 2.5, length.out = Num.Res)
   )
resource.freq.skew.slc <- res.Abund*resource.freq.skew.slc


# CLC:

resource.property.skew.clc <- c(seq(from = -2.5, to = 2.5, length.out = Num.
   Res))


resource.frequency.skew.clc <- c()
for (i in 1:Num.Res){
```

```
  resource.frequency.skew.clc[i] <- i*x
}


resource.abundance.adults     <- res.Abund
resource.abundance.juveniles  <- res.Abund

resFreqMatrix.skew.clc <- matrix(resource.frequency.skew.clc, nrow=2, ncol=
   length(resource.frequency.skew.clc), byrow = TRUE)
resFreqMatrix.skew.clc[1, ] <- resFreqMatrix.skew.clc[1, ]*resource.
   abundance.adults
resFreqMatrix.skew.clc[2, ] <- resFreqMatrix.skew.clc[2, ]*resource.
   abundance.juveniles

resPropMatrix.skew.clc <- matrix(resource.property.skew.clc, nrow=2, ncol=
   length(resource.property.skew.clc), byrow = TRUE)

rownames(resFreqMatrix.skew.clc) <- c("Adult", "Juvenile")
colnames(resFreqMatrix.skew.clc)  <- paste0("Resource␣", 1:ncol(
   resFreqMatrix.skew.clc))

rownames(resPropMatrix.skew.clc)<-c("Adult", "Juvenile")
colnames(resPropMatrix.skew.clc)  <- paste0("Resource␣", 1:ncol(
   resPropMatrix.skew.clc))



# Bimodal normal resources
# These are created using two normal resource distributions next to eacht
   oher

m1 <- -1.25        # Mean of the two distributions
m2 <-  1.25
s <- 0.5           # Standard deviation
Bi.resource.frequency <- c()
Bi.resource.property<- c(seq(from = -2.5, to = 2.5, length.out = Num.Res))



mid.add <- c()
midpoint <- c()

# Similar as the for loop that creates an approximation of normal resource
   except the
# two different normal curves are used. At first a normal curve with mean
   -1.25 is used
# in the middle of the resource distribution it switches to the the normal
   curve with mean
# 1.25. As the total sum under two normal curves is = 2, each frequency
   value is diveded by
# 2 to create a frequency where the total sum = 1.
# This functions value of s and m1 and m2 would need to be adjusted and
```

```
   checked if another
# set of trait values is explored. For example: sum( Bi.resource.frequency)
   should be approximatly
# equal to 1.

for(i in 1:(length(Bi.resource.property))){
  mid.add <- (Bi.resource.property[i+1]-Bi.resource.property[i])/2
  high.midpoint <- Bi.resource.property[(i)]+mid.add
  low.midpoint <- Bi.resource.property[(i)]-mid.add
  if(i == 1){
    Bi.resource.frequency[i] <- pnorm(high.midpoint, mean = m1, sd = s)/2
  }else if(i == length(Bi.resource.property)){
    low.midpoint <- Bi.resource.property[(i-1)] + (Bi.resource.property[i]-
       Bi.resource.property[i-1])/2
    Bi.resource.frequency[i] <- pnorm(low.midpoint, mean = m2, sd = s, lower
       .tail = FALSE)
  }else if (Bi.resource.property[i]<0) {
    Bi.resource.frequency[i] <- (pnorm(high.midpoint, mean = m1, sd = s) -
       pnorm(low.midpoint, mean = m1, sd = s))/2
  }else{
    Bi.resource.frequency[i] <- (pnorm(high.midpoint, mean = m2, sd = s) -
       pnorm(low.midpoint, mean = m2, sd = s))/2
  }
}


resource.abundance.adults.binorm.clc      <- res.Abund
                                  # res. abundance of adults and juveniles
resource.abundance.juveniles.binorm.clc   <- res.Abund

# SLC:

resource.prop.binorm.slc  <- Bi.resource.property             # res.
   property
resource.freq.binorm.slc  <- res.Abund*Bi.resource.frequency


# CLC:

resFreqMatrix.binorm.clc  <- matrix(Bi.resource.frequency, nrow=2, ncol=
   length(Bi.resource.frequency), byrow = TRUE)

resFreqMatrix.binorm.clc [1, ] <- resFreqMatrix.binorm.clc [1, ]*resource.
   abundance.adults.binorm.clc
resFreqMatrix.binorm.clc [2, ] <- resFreqMatrix.binorm.clc [2, ]*resource.
   abundance.juveniles.binorm.clc

rownames(resFreqMatrix.binorm.clc ) <- c("Adult", "Juvenile")
colnames(resFreqMatrix.binorm.clc )  <- paste0("Resource␣", 1:ncol(
   resFreqMatrix.binorm.clc ))
```

```
resPropMatrix.binorm.clc  <- matrix(Bi.resource.property, nrow=2, ncol=
   length(Bi.resource.property), byrow = TRUE)


rownames(resPropMatrix.binorm.clc )<-c("Adult", "Juvenile")
colnames(resPropMatrix.binorm.clc )  <- paste0("Resource␣", 1:ncol(
   resPropMatrix.binorm.clc))


# Two resources

resource.prop <- c(-1,1)
resource.frequency <- c(0.5, 0.5)
resource.frequency.as <- c(0.2, 0.8)

resFreqMatrix.2res <- matrix(resource.frequency, nrow=2, ncol=length(
   resource.frequency), byrow = TRUE)
resFreqMatrixAs.2res <- matrix(resource.frequency.as, nrow=2, ncol=length(
   resource.frequency.as), byrow = TRUE)


resFreqMatrix.2res[1, ] <- resFreqMatrix.2res[1, ]*res.Abund
resFreqMatrix.2res[2, ] <- resFreqMatrix.2res[2, ]*res.Abund

resFreqMatrixAs.2res[1, ] <- resFreqMatrixAs.2res[1, ]*res.Abund
resFreqMatrixAs.2res[2, ] <- resFreqMatrixAs.2res[2, ]*res.Abund

rownames(resFreqMatrix.2res) <- c("Adult", "Juvenile")
colnames(resFreqMatrix.2res)  <- paste0("Resource␣", 1:ncol(resFreqMatrixAs
   .2res))

rownames(resFreqMatrixAs.2res) <- c("Adult", "Juvenile")
colnames(resFreqMatrixAs.2res)  <- paste0("Resource␣", 1:ncol(
   resFreqMatrixAs.2res))

resPropMatrix.2res <- matrix(resource.prop, nrow=2, ncol=length(resource.
   prop), byrow = TRUE)


rownames(resPropMatrix.2res)<-c("Adult", "Juvenile")
colnames(resFreqMatrix.2res)  <- paste0("Resource␣", 1:ncol(resPropMatrix.2
   res))
# Clean up -------------
# To de clutter the environment a bit

rm(high.midpoint)
rm(i)
rm(low.midpoint)
rm(m)
rm(m1)
rm(m2)
rm(mid.add)
```

```
rm(midpoint)
rm(N.resource.frequency)
rm(N.resource.property)
rm(res.Abund)
rm(resource.abundance.adults)
rm(resource.abundance.adults.binorm.clc)
rm(resource.abundance.adults.even.clc)
rm(resource.abundance.adults.norm.clc)
rm(resource.abundance.juveniles)
rm(resource.abundance.juveniles.binorm.clc)
rm(resource.abundance.juveniles.even.clc)
rm(resource.abundance.juveniles.norm.clc)
rm(resource.frequency.even.clc)
rm(resource.frequency.skew.clc)
rm(s)
rm(tot)
rm(x)
rm(Bi.resource.frequency)
rm(Bi.resource.property)
rm(resource.property.even.clc)
rm(resource.property.skew.clc)
rm(resource.prop)
rm(resource.frequency)
rm(resource.frequency.as)

# -------------------------

# Parameter initialization ----------------------------


popSize <- 10                                          # Initial population
   size
sigma <- seq(from = 0.05, to = 0.8, length.out = 6)  # Niche width
im <-  0                                               # Determines how if
   there is immigration or not, can be 0 or 1
fmax <-  2                                             # Maximum fecundity
   value
kA <-  0.5                                             # Half saturation
   constants
kJ <-  0.5
mutProb <- 0.00001                                     # Mutational
   probability
mutVar <- 0.05                                         # Variation in the
   amount of trait mutation
time.steps <- 50000                                    # Number of generations
    simulation is run for
iniP <- 0                                              # Initial phenotype for
    simple life cycle
iniPJ <- 0                                             # Initial phenotype for
    juvenile (complex)
iniPA <- 0                                             # Initial phenotype for
    adult (complex)
```

```
nmorphs <-  1                                          # Number of species at
   start of simulation
threshold <-  0.0005                                   # Threshold which is
   used at end of simulation to remove very low abundance species
maxTr = 3                                              # Maximum possible
   trait value of immigrant
minTr = -3                                             # Minimum possible
   trait value of immigrant
```

Simulation example

I run simulations using jobs to free up the console for other things. If you do not have a computer with the cpu power to handle several simulations at once, do the jobs separately. This simulation usually takes my computer around 1.5 days to run. The time can be shortened significantly by reducing the number of repetitions and the number of σ_s values explored. To check on the progress of the simulations, look under background jobs, (same page as console). Please ensure you have run the above code before this for it to work.

```
 # Running simulations ------------------------------


# Even

# The job function runs in a separate environment so we need to import the
   desired parameters and functions.
# The result will be an environment titled the same thing as the job's name.

job::job(even = {

  rep <- 10                                # Number of times simulation will be
     run

  # Where the output of simulations will be stored:

  Total.species.SLC.single.even <- c()

  Total.species.CLC.even <- matrix(data = NA, nrow = length(sigma), ncol =
     length(sigma))
  rownames(Total.species.CLC.even) <- sigma  #ADULTS
  colnames(Total.species.CLC.even) <- sigma #JUVENILES



  # SLC

  Total.species.SLC.even <- list()
  Total.endpoint.SLC.even <- list()

  # Simulations:

  for(r in 1:rep) {

    id <- 1
                                                                          #
```

```
      Used to differentiate different runs in the output

  print(paste0("loop␣", r, "␣started"))
                                          # To see progress of simulation
      in "background jobs"

  Number.species.SLC.even <- c()
  endpoint.SLC.even <- list()

  for(i in 1:length(sigma)){


    outputSLC <- resourceCompetitionSLC(resProp=resource.prop.even.slc,
       iniP = iniP, resFreq=resource.freq.even.slc, resGen=matrix(c(sigma[
       i],sigma[i])),
                                        popSize = popSize, mutProb=mutProb
                                           , mutVar=mutVar, time.steps =
                                           time.steps, im = im, fmax =
                                           fmax, kA = kA, nmorphs =
                                           nmorphs,
                                        threshold = threshold)


    #Filter out similar "species" and collect number of species data

    final.data.SLC.even <- slc.groups(output = outputSLC)
    Number.species.SLC.even[i] <- nrow(final.data.SLC.even)

    #Collect endpoint data
    final.data.SLC.even$ID <- c(rep(id, times = nrow(final.data.SLC.even))
       )


    endpoint.SLC.even[[i]] <- final.data.SLC.even
    id <- id + 1
  }

  Total.species.SLC.even[[r]] <- Number.species.SLC.even
  Total.endpoint.SLC.even[[r]] <- endpoint.SLC.even
}


# Caluclating mean and SD of 10 runs



Total.mean.SLC.even <- sapply(1:length(sigma), function(i) mean(sapply(
   Total.species.SLC.even, function(x) x[i])))

array.data.SLC <- array(unlist(Total.species.SLC.even), dim = c(dim(Total.
   species.SLC.even[[1]]), length(Total.species.SLC.even)))
```

```
Total.sd.SLC.even <- sapply(1:length(sigma), function(i) sd(sapply(Total.
   species.SLC.even, function(x) x[i])))


# CLC

print("clc start")

Total.species.CLC.even <- list()

Total.endpoint.CLC.even <- list()



for(a in 1:rep){
  print(paste0("loop ", a, " started"))


  id <- 1

  species.CLC.even <- matrix(data = NA, nrow = length(sigma), ncol =
     length(sigma))
  rownames(species.CLC.even) <- sigma  #ADULTS
  colnames(species.CLC.even) <- sigma #JUVENILES

  endpoint.CLC.even <- c()

  for(b in 1:length(sigma)){

    for(k in 1:length(sigma)){


      outputCLC <- resourceCompetitionCLC(resProp=resPropMatrix.even.clc,
         resFreq=resFreqMatrix.even.clc, iniPA = iniPA, iniPJ = iniPJ,
         resGen=matrix(c(sigma[b],sigma[k])),
                                          popSize = popSize, mutProb=
                                             mutProb, mutVar=mutVar, time.
                                             steps = time.steps, im = im,
                                             fmax = fmax, kA = kA, nmorphs
                                              = nmorphs,
                                          threshold = threshold)



      #Filter out similar "species"
      final.data.CLC.even <- clc.groups(output = outputCLC)

      # Collect Data

      species.CLC.even[b, k] <- nrow(final.data.CLC.even)
      final.data.CLC.even$Adult.gen <- c(rep(sigma[b], times = nrow(final.
         data.CLC.even)))
```

```
        final.data.CLC.even$Juv.gen <- c(rep(sigma[k], times = nrow(final.
           data.CLC.even)))
        final.data.CLC.even$ID <- c(rep(id, times = nrow(final.data.CLC.even
           )))
        endpoint.CLC.even <- rbind(endpoint.CLC.even, final.data.CLC.even)
        id <- id + 1
      }

    }
    Total.species.CLC.even[[a]] <- species.CLC.even
    Total.endpoint.CLC.even[[a]] <- endpoint.CLC.even
  }

  # Calculating mean of 10 runs

  # Combine matrices in the list into a 3D array
  array.data.CLC <- array(unlist(Total.species.CLC.even), dim = c(dim(Total.
     species.CLC.even[[1]]), length(Total.species.CLC.even)))


  # Calculate mean and standard deviation along the third dimension (across
     the list)
  Total.mean.CLC.even <- apply(array.data.CLC, c(1, 2), mean)
  Total.sd.CLC.even <- apply(array.data.CLC, c(1, 2), sd)



  job::export(list(Total.mean.CLC.even, Total.sd.CLC.even, Total.mean.SLC.
     even, Total.sd.SLC.even, Total.endpoint.SLC.even, Total.endpoint.CLC.
     even))
}, import = c(resPropMatrix.even.clc, resFreqMatrix.even.clc,
   resourceCompetitionCLC, resource.prop.even.slc, resource.freq.even.slc,
   resourceCompetitionSLC,
              clc.groups, slc.groups, sigma, popSize, im, fmax, kA, kJ,
                 mutProb, mutVar, time.steps, iniP, iniPA, iniPJ, nmorphs,
                 threshold, maxTr, minTr))




# Normal

job::job(norm = {

  rep <- 10

  Total.species.SLC.single.norm <- c()

  Total.species.CLC.norm <- matrix(data = NA, nrow = length(sigma), ncol =
     length(sigma))
  rownames(Total.species.CLC.norm) <- sigma  #ADULTS
  colnames(Total.species.CLC.norm) <- sigma #JUVENILES
```

```
# SLC

Total.species.SLC.norm <- list()
Total.endpoint.SLC.norm <- list()


for(r in 1:rep) {


  id <- 1

  print(paste0("loop␣", r, "␣started"))

  Number.species.SLC.norm <- c()
  endpoint.SLC.norm <- list()

  for(i in 1:length(sigma)){


    outputSLC <- resourceCompetitionSLC(resProp=resource.prop.norm.slc,
       iniP = iniP, resFreq=resource.freq.norm.slc, resGen=matrix(c(sigma[
       i],sigma[i])),
                                        popSize = popSize, mutProb=mutProb
                                           , mutVar=mutVar, time.steps =
                                           time.steps, im = im, fmax =
                                           fmax, kA = kA, nmorphs =
                                           nmorphs,
                                        threshold = threshold)


    #Filter out similar "species" and collect number of species data

    final.data.SLC.norm <- slc.groups(output = outputSLC)
    Number.species.SLC.norm[i] <- nrow(final.data.SLC.norm)

    #Collect endpoint data
    final.data.SLC.norm$ID <- c(rep(id, times = nrow(final.data.SLC.norm))
       )


    endpoint.SLC.norm[[i]] <- final.data.SLC.norm
    id <- id + 1
  }

  Total.species.SLC.norm[[r]] <- Number.species.SLC.norm
  Total.endpoint.SLC.norm[[r]] <- endpoint.SLC.norm
}
```

```
# Caluclating mean and SD of 10 runs


Total.mean.SLC.norm <- sapply(1:length(sigma), function(i) mean(sapply(
   Total.species.SLC.norm, function(x) x[i])))

array.data.SLC <- array(unlist(Total.species.SLC.norm), dim = c(dim(Total.
   species.SLC.norm[[1]]), length(Total.species.SLC.norm)))

Total.sd.SLC.norm <- sapply(1:length(sigma), function(i) sd(sapply(Total.
   species.SLC.norm, function(x) x[i])))


# CLC

print("clc␣start")

Total.species.CLC.norm <- list()

Total.endpoint.CLC.norm <- list()



for(a in 1:rep){

  print(paste0("loop␣", a, "␣started"))


  id <- 1

  species.CLC.norm <- matrix(data = NA, nrow = length(sigma), ncol =
     length(sigma))
  rownames(species.CLC.norm) <- sigma  #ADULTS
  colnames(species.CLC.norm) <- sigma #JUVENILES

  endpoint.CLC.norm <- c()

  for(b in 1:length(sigma)){

    for(k in 1:length(sigma)){


      outputCLC <- resourceCompetitionCLC(resProp=resPropMatrix.norm.clc,
         resFreq=resFreqMatrix.norm.clc, iniPA = iniPA, iniPJ = iniPJ,
         resGen=matrix(c(sigma[b],sigma[k])),
                                             popSize = popSize, mutProb=
                                                mutProb, mutVar=mutVar, time.
                                                steps = time.steps, im = im,
                                                fmax = fmax, kA = kA, nmorphs
                                                 = nmorphs,
                                             threshold = threshold)
```

```
        #Filter out similar "species"
        final.data.CLC.norm <- clc.groups(output = outputCLC)

        # Collect Data

        species.CLC.norm[b, k] <- nrow(final.data.CLC.norm)
        final.data.CLC.norm$Adult.gen <- c(rep(sigma[b], times = nrow(final.
           data.CLC.norm)))
        final.data.CLC.norm$Juv.gen <- c(rep(sigma[k], times = nrow(final.
           data.CLC.norm)))
        final.data.CLC.norm$ID <- c(rep(id, times = nrow(final.data.CLC.norm
           )))
        endpoint.CLC.norm <- rbind(endpoint.CLC.norm, final.data.CLC.norm)
        id <- id + 1
      }

    }
    Total.species.CLC.norm[[a]] <- species.CLC.norm
    Total.endpoint.CLC.norm[[a]] <- endpoint.CLC.norm
  }

  # Calculating mean of 10 runs

  # Combine matrices in the list into a 3D array
  array.data.CLC <- array(unlist(Total.species.CLC.norm), dim = c(dim(Total.
     species.CLC.norm[[1]]), length(Total.species.CLC.norm)))


  # Calculate mean and standard deviation along the third dimension (across
     the list)
  Total.mean.CLC.norm <- apply(array.data.CLC, c(1, 2), mean)
  Total.sd.CLC.norm <- apply(array.data.CLC, c(1, 2), sd)



  job::export(list(Total.mean.CLC.norm, Total.sd.CLC.norm, Total.mean.SLC.
     norm, Total.sd.SLC.norm, Total.endpoint.SLC.norm, Total.endpoint.CLC.
     norm))
}, import = c(resPropMatrix.norm.clc, resFreqMatrix.norm.clc,
   resourceCompetitionCLC, resource.prop.norm.slc, resource.freq.norm.slc,
   resourceCompetitionSLC,
              clc.groups, slc.groups, sigma, popSize, im, fmax, kA, kJ,
                 mutProb, mutVar, time.steps, iniP, iniPA, iniPJ, nmorphs,
                 threshold, maxTr, minTr))
```

```
# Skewed


job::job(skew = {

  rep <- 10

  Total.species.SLC.single.skew <- c()

  Total.species.CLC.skew <- matrix(data = NA, nrow = length(sigma), ncol =
     length(sigma))
  rownames(Total.species.CLC.skew) <- sigma  #ADULTS
  colnames(Total.species.CLC.skew) <- sigma #JUVENILES



  # SLC

  Total.species.SLC.skew <- list()
  Total.endpoint.SLC.skew <- list()


  for(r in 1:rep) {

    id <- 1

    print(paste0("loop␣", r, "␣started"))

    Number.species.SLC.skew <- c()
    endpoint.SLC.skew <- list()

    for(i in 1:length(sigma)){


      outputSLC <- resourceCompetitionSLC(resProp=resource.prop.skew.slc,
         iniP = iniP, resFreq=resource.freq.skew.slc, resGen=matrix(c(sigma[
         i],sigma[i])),
                                          popSize = popSize, mutProb=mutProb
                                             , mutVar=mutVar, time.steps =
                                             time.steps, im = im, fmax =
                                             fmax, kA = kA, nmorphs =
                                             nmorphs,
                                          threshold = threshold)


      #Filter out similar "species" and collect number of species data

      final.data.SLC.skew <- slc.groups(output = outputSLC)
      Number.species.SLC.skew[i] <- nrow(final.data.SLC.skew)

      #Collect endpoint data
```

```
    final.data.SLC.skew$ID <- c(rep(id, times = nrow(final.data.SLC.skew))
       )


    endpoint.SLC.skew[[i]] <- final.data.SLC.skew
    id <- id + 1
  }

  Total.species.SLC.skew[[r]] <- Number.species.SLC.skew
  Total.endpoint.SLC.skew[[r]] <- endpoint.SLC.skew
}


# Caluclating mean and SD of 10 runs



Total.mean.SLC.skew <- sapply(1:length(sigma), function(i) mean(sapply(
   Total.species.SLC.skew, function(x) x[i])))

array.data.SLC <- array(unlist(Total.species.SLC.skew), dim = c(dim(Total.
   species.SLC.skew[[1]]), length(Total.species.SLC.skew)))

Total.sd.SLC.skew <- sapply(1:length(sigma), function(i) sd(sapply(Total.
   species.SLC.skew, function(x) x[i])))


# CLC

print("clc␣start")

Total.species.CLC.skew <- list()

Total.endpoint.CLC.skew <- list()



for(a in 1:rep){


  print(paste0("loop␣", a, "␣started"))

  id <- 1

  species.CLC.skew <- matrix(data = NA, nrow = length(sigma), ncol =
     length(sigma))
  rownames(species.CLC.skew) <- sigma  #ADULTS
  colnames(species.CLC.skew) <- sigma #JUVENILES

  endpoint.CLC.skew <- c()

  for(b in 1:length(sigma)){
```

```
    for(k in 1:length(sigma)){


      outputCLC <- resourceCompetitionCLC(resProp=resPropMatrix.skew.clc,
         resFreq=resFreqMatrix.skew.clc, iniPA = iniPA, iniPJ = iniPJ,
         resGen=matrix(c(sigma[b],sigma[k])),
                                          popSize = popSize, mutProb=
                                             mutProb, mutVar=mutVar, time.
                                             steps = time.steps, im = im,
                                             fmax = fmax, kA = kA, nmorphs
                                              = nmorphs,
                                          threshold = threshold)



      #Filter out similar "species"
      final.data.CLC.skew <- clc.groups(output = outputCLC)

      # Collect Data

      species.CLC.skew[b, k] <- nrow(final.data.CLC.skew)
      final.data.CLC.skew$Adult.gen <- c(rep(sigma[b], times = nrow(final.
         data.CLC.skew)))
      final.data.CLC.skew$Juv.gen <- c(rep(sigma[k], times = nrow(final.
         data.CLC.skew)))
      final.data.CLC.skew$ID <- c(rep(id, times = nrow(final.data.CLC.skew
         )))
      endpoint.CLC.skew <- rbind(endpoint.CLC.skew, final.data.CLC.skew)
      id <- id + 1
    }

  }
  Total.species.CLC.skew[[a]] <- species.CLC.skew
  Total.endpoint.CLC.skew[[a]] <- endpoint.CLC.skew
}

# Calculating mean of 10 runs

# Combine matrices in the list into a 3D array
array.data.CLC <- array(unlist(Total.species.CLC.skew), dim = c(dim(Total.
   species.CLC.skew[[1]]), length(Total.species.CLC.skew)))


# Calculate mean and standard deviation along the third dimension (across
   the list)
Total.mean.CLC.skew <- apply(array.data.CLC, c(1, 2), mean)
Total.sd.CLC.skew <- apply(array.data.CLC, c(1, 2), sd)
```

```
  job::export(list(Total.mean.CLC.skew, Total.sd.CLC.skew, Total.mean.SLC.
     skew, Total.sd.SLC.skew, Total.endpoint.SLC.skew, Total.endpoint.CLC.
     skew))
}, import = c(resPropMatrix.skew.clc, resFreqMatrix.skew.clc,
   resourceCompetitionCLC, resource.prop.skew.slc, resource.freq.skew.slc,
   resourceCompetitionSLC,
              clc.groups, slc.groups, sigma, popSize, im, fmax, kA, kJ,
                 mutProb, mutVar, time.steps, iniP, iniPA, iniPJ, nmorphs,
                 threshold, maxTr, minTr))


# Bimodal Normal

job::job(binorm = {

  rep <- 3

  Total.species.SLC.single.binorm <- c()

  Total.species.CLC.binorm <- matrix(data = NA, nrow = length(sigma), ncol =
      length(sigma))
  rownames(Total.species.CLC.binorm) <- sigma  #ADULTS
  colnames(Total.species.CLC.binorm) <- sigma #JUVENILES



  # SLC

  Total.species.SLC.binorm <- list()
  Total.endpoint.SLC.binorm <- list()

  for(r in 1:rep) {


    id <- 1

    print(paste0("loop␣", r, "␣started"))

    Number.species.SLC.binorm <- c()
    endpoint.SLC.binorm <- list()

    for(i in 1:length(sigma)){


      outputSLC <- resourceCompetitionSLC(resProp=resource.prop.binorm.slc,
         iniP = iniP, resFreq=resource.freq.binorm.slc, resGen=matrix(c(
         sigma[i],sigma[i])),
                                          popSize = popSize, mutProb=mutProb
                                             , mutVar=mutVar, time.steps =
                                             time.steps, im = im, fmax =
                                             fmax, kA = kA, nmorphs =
                                             nmorphs,
```

```
                                        threshold = threshold)

    #Filter out similar "species" and collect number of species data

    final.data.SLC.binorm <- slc.groups(output = outputSLC)
    Number.species.SLC.binorm[i] <- nrow(final.data.SLC.binorm)

    #Collect endpoint data
    final.data.SLC.binorm$ID <- c(rep(id, times = nrow(final.data.SLC.
       binorm)))


    endpoint.SLC.binorm[[i]] <- final.data.SLC.binorm
    id <- id + 1
  }

  Total.species.SLC.binorm[[r]] <- Number.species.SLC.binorm
  Total.endpoint.SLC.binorm[[r]] <- endpoint.SLC.binorm
}


# Caluclating mean and SD of 10 runs



Total.mean.SLC.binorm <- sapply(1:length(sigma), function(i) mean(sapply(
   Total.species.SLC.binorm, function(x) x[i])))

array.data.SLC <- array(unlist(Total.species.SLC.binorm), dim = c(dim(
   Total.species.SLC.binorm[[1]]), length(Total.species.SLC.binorm)))

Total.sd.SLC.binorm <- sapply(1:length(sigma), function(i) sd(sapply(Total
   .species.SLC.binorm, function(x) x[i])))


# CLC

print("clc␣start")

Total.species.CLC.binorm <- list()

Total.endpoint.CLC.binorm <- list()


for(a in 1:rep){
  print(paste0("loop␣", a, "␣started"))

  id <- 1

  species.CLC.binorm <- matrix(data = NA, nrow = length(sigma), ncol =
     length(sigma))
```

```
  rownames(species.CLC.binorm) <- sigma  #ADULTS
  colnames(species.CLC.binorm) <- sigma #JUVENILES

  endpoint.CLC.binorm <- c()

  for(b in 1:length(sigma)){

    for(k in 1:length(sigma)){


      outputCLC <- resourceCompetitionCLC(resProp=resPropMatrix.binorm.clc
         , resFreq=resFreqMatrix.binorm.clc, iniPA = iniPA, iniPJ = iniPJ,
          resGen=matrix(c(sigma[b],sigma[k])),
                                          popSize = popSize, mutProb=
                                             mutProb, mutVar=mutVar, time.
                                             steps = time.steps, im = im,
                                             fmax = fmax, kA = kA, nmorphs
                                              = nmorphs,
                                          threshold = threshold)



      #Filter out similar "species"
      final.data.CLC.binorm <- clc.groups(output = outputCLC)

      # Collect Data

      species.CLC.binorm[b, k] <- nrow(final.data.CLC.binorm)
      final.data.CLC.binorm$Adult.gen <- c(rep(sigma[b], times = nrow(
         final.data.CLC.binorm)))
      final.data.CLC.binorm$Juv.gen <- c(rep(sigma[k], times = nrow(final.
         data.CLC.binorm)))
      final.data.CLC.binorm$ID <- c(rep(id, times = nrow(final.data.CLC.
         binorm)))
      endpoint.CLC.binorm <- rbind(endpoint.CLC.binorm, final.data.CLC.
         binorm)
      id <- id + 1
    }

  }
  Total.species.CLC.binorm[[a]] <- species.CLC.binorm
  Total.endpoint.CLC.binorm[[a]] <- endpoint.CLC.binorm
}

# Calculating mean of 10 runs

# Combine matrices in the list into a 3D array
array.data.CLC <- array(unlist(Total.species.CLC.binorm), dim = c(dim(
   Total.species.CLC.binorm[[1]]), length(Total.species.CLC.binorm)))
```

```
  # Calculate mean and standard deviation along the third dimension (across
     the list)
  Total.mean.CLC.binorm <- apply(array.data.CLC, c(1, 2), mean)
  Total.sd.CLC.binorm <- apply(array.data.CLC, c(1, 2), sd)



  job::export(list(Total.mean.CLC.binorm, Total.sd.CLC.binorm, Total.mean.
     SLC.binorm, Total.sd.SLC.binorm, Total.endpoint.SLC.binorm, Total.
     endpoint.CLC.binorm))
}, import = c(resPropMatrix.binorm.clc, resFreqMatrix.binorm.clc,
   resourceCompetitionCLC, resource.prop.binorm.slc, resource.freq.binorm.
   slc,
              resourceCompetitionSLC, clc.groups, slc.groups, sigma, popSize
                 , im, fmax, kA, kJ, mutProb, mutVar, time.steps, iniP,
                 iniPA, iniPJ, nmorphs, threshold, maxTr, minTr))



# ------------------------------------------------
```

Plotting example

There are of course many ways to plot the data so feel free to use your own method. But this is how I produced the graphs in the thesis. The graphs produced by this is the mean number of species for all the different combinations of σ_s (figure 4 and 8) and graphs showing phenotypic endpoints for the complex communities for all $\sigma_a = \sigma_j$

```
# Plotting Mean number of Species
   ------------------------------------------------

# Even


Total.mean.CLC.even <- even$Total.mean.CLC.even
Total.mean.SLC.even <- even$Total.mean.SLC.even

Total.sd.CLC.even <-  even$Total.sd.CLC.even
Total.sd.SLC.even <-  even$Total.sd.SLC.even

x <- as.factor(sigma)



df.CLC <- data.frame(
  Juvenile.trait = rep(x, each = length(x)),
  Adult.trait = rep(x, times = length(x)),
  Richness = as.vector(Total.mean.CLC.even),
  sd = as.vector(Total.sd.CLC.even),
  Cycle = rep("Complex", times = length(x)*length(x))
)

df.SLC <- data.frame(
```

```
  Juvenile.trait = x,
  Adult.trait = x,
  Richness = as.vector(Total.mean.SLC.even),
  sd = as.vector(Total.sd.SLC.even),
  Cycle = rep("Simple", times = length(x))
)



df.combined <- rbind(df.CLC, df.SLC)

color_palette <- magma(length(sigma))

even.plot <- ggplot(df.combined, aes(x = Adult.trait, y = Richness, shape =
   Cycle, color = Juvenile.trait, stroke = 1.7)) +
  geom_point(data = ~filter(.x, Cycle == "Simple"),size = 7, position =
     position_dodge(0.2), color = "black") +
  geom_point(data = ~filter(.x, Cycle == "Complex"),size = 7, position =
     position_dodge(0.2)) +
  #geom_errorbar(aes(ymin=Richness-sd, ymax=Richness+sd), width=.05) +   #
     position=position_dodge(.9)  # If you wish to add error bars include
     this.
  scale_y_continuous(limits = c(0, 30)) +
  xlab("Adult␣Generalism") +
  ylab("Number␣of␣species") +
  labs(color = "Juvenile␣\nGeneralism", shape = "Life␣cycle") +
  ggtitle("Even␣Resource␣distribution") +
  theme_minimal(base_family = "LM␣Roman␣10", base_size = 15) +
  theme(plot.title = element_text(size = 18)) +
  scale_shape_manual(values = c(1,4)) +
  scale_color_manual(values = c(color_palette, "black"))




even.plot

# Normal

Total.mean.CLC.norm <- norm$Total.mean.CLC.norm
Total.mean.SLC.norm <- norm$Total.mean.SLC.norm

Total.sd.CLC.norm <-  norm$Total.sd.CLC.norm
Total.sd.SLC.norm <-  norm$Total.sd.SLC.norm

x <- as.factor(sigma)



df.CLC <- data.frame(
  Juvenile.trait = rep(x, each = length(x)),
  Adult.trait = rep(x, times = length(x)),
```

```
  Richness = as.vector(Total.mean.CLC.norm),
  sd = as.vector(Total.sd.CLC.norm),
  Cycle = rep("Complex", times = length(x)*length(x))
)

df.SLC <- data.frame(
  Juvenile.trait = x,
  Adult.trait = x,
  Richness = as.vector(Total.mean.SLC.norm),
  sd = as.vector(Total.sd.SLC.norm),
  Cycle = rep("Simple", times = length(x))
)



df.combined <- rbind(df.CLC, df.SLC)


norm.plot <- ggplot(df.combined, aes(x = Adult.trait, y = Richness, shape =
   Cycle, color = Juvenile.trait, stroke = 1.7)) +
  geom_point(data = ~filter(.x, Cycle == "Simple"),size = 7, position =
     position_dodge(0.2), color = "black") +
  geom_point(data = ~filter(.x, Cycle == "Complex"),size = 7, position =
     position_dodge(0.2)) +
  #geom_errorbar(aes(ymin=Richness-sd, ymax=Richness+sd), width=.1) +   #
     position=position_dodge(.9)
  scale_y_continuous(limits = c(0, 30)) +
  xlab("Adult␣Generalism") +
  ylab("Number␣of␣species") +
  labs(color = "Juvenile␣\nGeneralism", shape = "Life␣cycle") +
  ggtitle("Normal␣Resource␣distribution") +
  theme_minimal(base_family = "LM␣Roman␣10", base_size = 15) +
  theme(plot.title = element_text(size = 18)) +
  scale_shape_manual(values = c(1,4)) +
  scale_color_manual(values = c(color_palette, "black"))

norm.plot

# Skewed


Total.mean.CLC.skew <- skew$Total.mean.CLC.skew
Total.mean.SLC.skew <- skew$Total.mean.SLC.skew

Total.sd.CLC.skew <-  skew$Total.sd.CLC.skew
Total.sd.SLC.skew <-  skew$Total.sd.SLC.skew

x <- as.factor(sigma)



df.CLC <- data.frame(
```

```
  Juvenile.trait = rep(x, each = length(x)),
  Adult.trait = rep(x, times = length(x)),
  Richness = as.vector(Total.mean.CLC.skew),
  sd = as.vector(Total.sd.CLC.skew),
  Cycle = rep("Complex", times = length(x)*length(x))
)

df.SLC <- data.frame(
  Juvenile.trait = x,
  Adult.trait = x,
  Richness = as.vector(Total.mean.SLC.skew),
  sd = as.vector(Total.sd.SLC.skew),
  Cycle = rep("Simple", times = length(x))
)



df.combined <- rbind(df.CLC, df.SLC)


skew.plot <- ggplot(df.combined, aes(x = Adult.trait, y = Richness, shape =
   Cycle, color = Juvenile.trait, stroke = 1.7)) +
  geom_point(data = ~filter(.x, Cycle == "Simple"),size = 7, position =
     position_dodge(0.2), color = "black") +
  geom_point(data = ~filter(.x, Cycle == "Complex"),size = 7, position =
     position_dodge(0.2)) +
  #geom_errorbar(aes(ymin=Richness-sd, ymax=Richness+sd), width=.05) +   #
     position=position_dodge(.9)
  scale_y_continuous(limits = c(0, 30)) +
  xlab("Adult␣Generalism") +
  ylab("Number␣of␣species") +
  labs(color = "Juvenile␣\nGeneralism", shape = "Life␣cycle") +
  ggtitle("Skewed␣Resource␣distribution") +
  theme_minimal(base_family = "LM␣Roman␣10", base_size = 15) +
  theme(plot.title = element_text(size = 18)) +
  scale_shape_manual(values = c(1,4)) +
  scale_color_manual(values = c(color_palette, "black"))

skew.plot

# Binormal

Total.mean.CLC.binorm <- binorm$Total.mean.CLC.binorm
Total.mean.SLC.binorm <- binorm$Total.mean.SLC.binorm

Total.sd.CLC.binorm <-  binorm$Total.sd.CLC.binorm
Total.sd.SLC.binorm <-  binorm$Total.sd.SLC.binorm

x <- as.factor(sigma)
```

```
df.CLC <- data.frame(
  Juvenile.trait = rep(x, each = length(x)),
  Adult.trait = rep(x, times = length(x)),
  Richness = as.vector(Total.mean.CLC.binorm),
  sd = as.vector(Total.sd.CLC.binorm),
  Cycle = rep("Complex", times = length(x)*length(x))
)

df.SLC <- data.frame(
  Juvenile.trait = x,
  Adult.trait = x,
  Richness = as.vector(Total.mean.SLC.binorm),
  sd = as.vector(Total.sd.SLC.binorm),
  Cycle = rep("Simple", times = length(x))
)



df.combined <- rbind(df.CLC, df.SLC)


binorm.plot <- ggplot(df.combined, aes(x = Adult.trait, y = Richness, shape
   = Cycle, color = Juvenile.trait, stroke = 1.7)) +
  geom_point(data = ~filter(.x, Cycle == "Simple"),size = 7, position =
     position_dodge(0.2), color = "black") +
  geom_point(data = ~filter(.x, Cycle == "Complex"),size = 7, position =
     position_dodge(0.2)) +
  #geom_errorbar(aes(ymin=Richness-sd, ymax=Richness+sd), width=.05) +   #
     position=position_dodge(.9)
  scale_y_continuous(limits = c(0, 30)) +
  xlab("Adult␣Generalism") +
  ylab("Number␣of␣species") +
  labs(color = "Juvenile␣\nGeneralism", shape = "Life␣cycle") +
  ggtitle("Bimodal␣Normal␣Resource␣distribution") +
  theme_minimal(base_family = "LM␣Roman␣10", base_size = 15) +
  theme(plot.title = element_text(size = 18)) +
  scale_shape_manual(values = c(1,4)) +
  scale_color_manual(values = c(color_palette, "black"))

binorm.plot

# Together:

all.plots <- (even.plot + norm.plot) / (skew.plot + binorm.plot)
all.plots + plot_layout(guides = "collect") + plot_annotation(tag_levels = "
   A",
  tag_prefix = "(",
  tag_suffix = ")")

# Plotting Phenotype Endpoint --------------------

# Even ----------------
```

```
# Plotting several runs ---------------------------------------

# Adult = Juvenile sigma

Res <- list()

pdf("plots.even.combined.side.pdf")   # This will create a pdf in your
   working directory that is filled with the plots
                                          # created after the command dev.
   off() is used.

for(s in 1:length(sigma)){
  adu.sigma <- sigma[s]


  juv.sigma <- adu.sigma


  last.year.list.even <- data.frame()

  for(i in 1:length(even$Total.endpoint.CLC.even)){
    this.run <- even$Total.endpoint.CLC.even[[i]]
    this.run$run <- rep(i, time = nrow(this.run))
    this.run <- this.run[this.run$Adult.gen == adu.sigma, ]
    this.run <- this.run[this.run$Juv.gen == juv.sigma, ]
    last.year.list.even <- rbind(last.year.list.even, this.run)
  }

  plot.list.even <- list()

  for (i in 1:9){

    data <- last.year.list.even[last.year.list.even$run == i, ]

    color.palette <- plasma(length(data$Adult_Trait))

    plot.list.even[[i]] <- ggplot(data, aes(x = Juvenile_Trait, y = Adult_
       Trait)) +
      geom_point(aes(size=Num_Individuals), color = color.palette, show.
         legend = FALSE) +
      labs(title = substitute(sigma == value, list(value = adu.sigma)), x =
         "Juvenile␣Trait", y = "Adult␣Trait", size = "Number␣of␣individuals"
         ) +                  # Labels for the axes
      scale_x_continuous(limits = c(-3, 3))+
      scale_y_continuous(limits = c(-3, 3))+
      scale_size_continuous(limits=c(1,40000),breaks=c(seq(from = 0, to =
         40000, by = 5000))) +
      theme(aspect.ratio=1) +
      theme_minimal(base_family = "LM␣Roman␣10", base_size = 10)


  }
```

```
  plots <- wrap_plots(plot.list.even)



  Res[[s]] <- plots + plot_annotation(
    title = 'Even Distribution',
    theme = theme(plot.title = element_text(hjust = 0.5, size = 15, family =
        "LM Roman 10"), plot.subtitle = element_text(hjust = 0.5, size = 15,
        family = "LM Roman 10"))
  )+ coord_fixed()

  print(Res[[s]])


}

dev.off()

# Normal ------------------
# Plotting several runs ----------------------------------------

# Adult = Juvenile sigma

Res <- list()

pdf("plots.norm.combined.side.pdf")

for(s in 1:length(sigma)){
  adu.sigma <- sigma[s]

  juv.sigma <- adu.sigma


  last.year.list.norm <- data.frame()

  for(i in 1:length(norm$Total.endpoint.CLC.norm)){
    this.run <- norm$Total.endpoint.CLC.norm[[i]]
    this.run$run <- rep(i, time = nrow(this.run))
    this.run <- this.run[this.run$Adult.gen == adu.sigma, ]
    this.run <- this.run[this.run$Juv.gen == juv.sigma, ]
    last.year.list.norm <- rbind(last.year.list.norm, this.run)
  }

  plot.list.norm <- list()

  for (i in 1:9){

    data <- last.year.list.norm[last.year.list.norm$run == i, ]

    color.palette <- plasma(length(data$Adult_Trait))
```

```
    plot.list.norm[[i]] <- ggplot(data, aes(x = Juvenile_Trait, y = Adult_
       Trait)) +
      geom_point(aes(size=Num_Individuals), color = color.palette, show.
         legend = FALSE) +
      labs(title = substitute(sigma == value, list(value = adu.sigma)), x =
         "Juvenile Trait", y = "Adult Trait", size = "Number of individuals"
         ) +                  # Labels for the axes
      scale_x_continuous(limits = c(-3, 3))+
      scale_y_continuous(limits = c(-3, 3))+
      scale_size_continuous(limits=c(1,40000),breaks=c(seq(from = 0, to =
         40000, by = 5000))) +
      theme_minimal(base_family = "LM Roman 10", base_size = 10)


  }


  plots <- wrap_plots(plot.list.norm)

  Res[[s]] <- plots + plot_annotation(
    title = 'Normal Distribution',
    theme = theme(plot.title = element_text(hjust = 0.5, size = 15, family =
        "LM Roman 10"), plot.subtitle = element_text(hjust = 0.5, size = 15,
        family = "LM Roman 10"))
  )+ coord_fixed()
  print(Res[[s]])


}

dev.off()

# Skewed --------------------
# Plotting several runs ---------------------------------------

# Adult = Juvenile sigma
Res <- list()

pdf("plots.skew.combined.side.pdf")

for(s in 1:length(sigma)){
  adu.sigma <- sigma[s]

  juv.sigma <- adu.sigma


  last.year.list.skew <- data.frame()

  for(i in 1:length(skew$Total.endpoint.CLC.skew)){
    this.run <- skew$Total.endpoint.CLC.skew[[i]]
    this.run$run <- rep(i, time = nrow(this.run))
```

```
    this.run <- this.run[this.run$Adult.gen == adu.sigma, ]
    this.run <- this.run[this.run$Juv.gen == juv.sigma, ]
    last.year.list.skew <- rbind(last.year.list.skew, this.run)
  }

  plot.list.skew <- list()

  for (i in 1:9){

    data <- last.year.list.skew[last.year.list.skew$run == i, ]

    color.palette <- plasma(length(data$Adult_Trait))

    plot.list.skew[[i]] <- ggplot(data, aes(x = Juvenile_Trait, y = Adult_
       Trait)) +
      geom_point(aes(size=Num_Individuals), color = color.palette, show.
         legend = FALSE) +
      labs(title = substitute(sigma == value, list(value = adu.sigma)), x =
         "Juvenile␣Trait", y = "Adult␣Trait", size = "Number␣of␣individuals"
         ) +                  # Labels for the axes
      scale_x_continuous(limits = c(-3, 3))+
      scale_y_continuous(limits = c(-3, 3))+
      scale_size_continuous(limits=c(1,40000),breaks=c(seq(from = 0, to =
         40000, by = 5000))) +
      theme_minimal(base_family = "LM␣Roman␣10", base_size = 10)


  }


  plots <- wrap_plots(plot.list.skew)

  Res[[s]] <- plots + plot_annotation(
    title = 'Skewed␣Distribution',
    theme = theme(plot.title = element_text(hjust = 0.5, size = 15, family =
        "LM␣Roman␣10"), plot.subtitle = element_text(hjust = 0.5, size = 15,
        family = "LM␣Roman␣10"))
  )+ coord_fixed()
  print(Res[[s]])


}

dev.off()


# Bi modal Normal -------------------
# Plotting several runs ---------------------------------------

# Adult = Juvenile sigma
Res <- list()
```

```
pdf("plots.binorm.combined.pdf")

for(s in 1:length(sigma)){
  adu.sigma <- sigma[s]

  juv.sigma <- adu.sigma


  last.year.list.binorm <- data.frame()

  for(i in 1:length(binorm$Total.endpoint.CLC.binorm)){
    this.run <- binorm$Total.endpoint.CLC.binorm[[i]]
    this.run$run <- rep(i, time = nrow(this.run))
    this.run <- this.run[this.run$Adult.gen == adu.sigma, ]
    this.run <- this.run[this.run$Juv.gen == juv.sigma, ]
    last.year.list.binorm <- rbind(last.year.list.binorm, this.run)
  }

  plot.list.binorm <- list()

  for (i in 1:9){

    data <- last.year.list.binorm[last.year.list.binorm$run == i, ]

    color.palette <- plasma(length(data$Adult_Trait))

    plot.list.binorm[[i]] <- ggplot(data, aes(x = Juvenile_Trait, y = Adult_
       Trait)) +
      geom_point(aes(size=Num_Individuals), color = color.palette, show.
         legend = FALSE) +
      labs(title = substitute(sigma == value, list(value = adu.sigma)), x =
         "Juvenile␣Trait", y = "Adult␣Trait", size = "Number␣of␣individuals"
         ) +                  # Labels for the axes
      scale_x_continuous(limits = c(-3, 3))+
      scale_y_continuous(limits = c(-3, 3))+
      scale_size_continuous(limits=c(1,40000),breaks=c(seq(from = 0, to =
         40000, by = 5000))) +
      theme_minimal(base_family = "LM␣Roman␣10", base_size = 10)


  }


  plots <- wrap_plots(plot.list.binorm)

  Res[[s]] <- plots + plot_annotation(
    title = 'Bimodal␣Normal␣Distribution',
    theme = theme(plot.title = element_text(hjust = 0.5, size = 15, family =
        "LM␣Roman␣10"), plot.subtitle = element_text(hjust = 0.5, size = 15,
        family = "LM␣Roman␣10"))
  )+ coord_fixed()
  print(Res[[s]])
```

```
}

dev.off()
```

References

Chang, W. 2023. extrafont: Tools for Using Fonts.

Garnier, Simon, Ross, Noam, Rudis, Robert, Camargo, A. Pedro, Sciaini, Marco, Scherer, and Cédric. 2023. viridis(Lite) - Colorblind-Friendly Color Maps for R.

Lindeløv, J. K. 2021. job: Run Code as an RStudio Job - Free Your Console.

Pedersen, T. L. 2024. patchwork: The Composer of Plots.

R Core Team. 2023. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Saul, M. 2021. FamilyRank: Algorithm for Ranking Predictors Using Graphical Domain Knowledge.

Wickham, H. 2016. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York.

Wickham, H., M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. 2019. Welcome to the tidyverse. Journal of Open Source Software 4:1686.

Wickham, H., R. François, L. Henry, K. Müller, and D. Vaughan. 2023. dplyr: A Grammar of Data Manipulation.