PhD Chapter 4: Simulation Framework

Erik Kusch; Anna Vinton

2022-06-24

Contents

Introduction

Scientific Goals

Throughout this document, Anna Vinton and I, Erik Kusch, build a data simulation framework for assessment of performance of ecological network inference frameworks. This simulation framework will additionally be used to attempt to improve on pre-existing ecological network inference methodology.

The final simulation function will be characterised by:

1. **Input** - a network of species and their interactions
2. **Process** - the *input* is used to:
    a. generate virtual species corresponding to nodes in the *input*

    b. generate individuals for each virtual species with a unique trait value, location

    c. generate a spatial product of environmental parameters for the individuals of all species to inhabit

    d. place individuals of all virtual species into the environment

    e. simulate population growth/decline using a dynamic death rate driven by mismatches in trait- and environment-values as well as spatially explicit effects of species interactions

3. **Output** - time series of population sizes of each species at each time-step of the simulation

R Leprations

For the following, we require some packages - mostly for visualisations. These are:

```
library(ggplot2) # for visualisations
library(cowplot) # for combining plots
```

Building the Simulation

Here, I document how the simulation is built up piece-by-piece.

One-Species Through Time

The base code for this part of the simulation, we have adapted from course material provided on-line by Dan McGlinn. This simulation represents a basic **simple model of logistic growth**:

```r
dNt <- function(r, # population growth rate
                Nt, # population size
                k # carrying capacity
                ){ r * Nt * (k - Nt)}
```

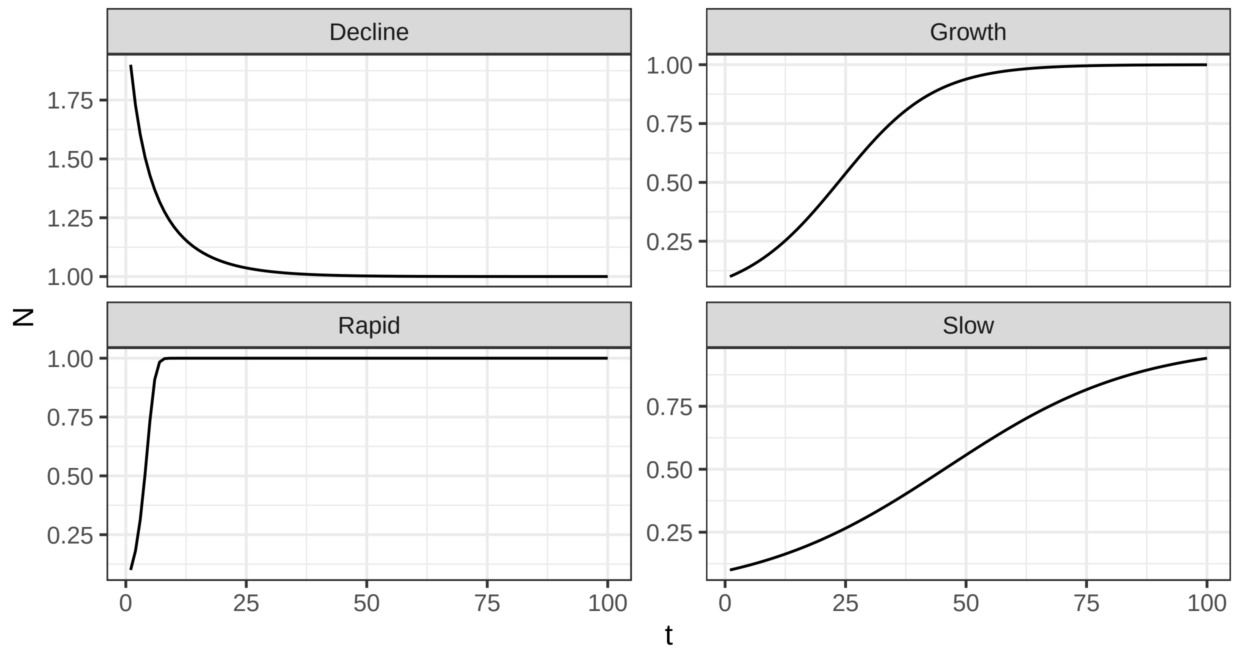
With this approach, a population will increase in size towards the carrying capacity if its starting population size (N_0) $N_0 < k$. If $N_0 > k$ the population size will decrease towards the carrying capacity.

This function now needs to be iterated over time:

```r
SIM.Nt <- function(r, N_0, t, k) {
  N <- N_0 # registering starting population size as a new object
  for(i in 1:(t - 1)){ # loop over time-steps specified
    # identify new population size given logistic growth
    N[i + 1] <- N[i] + dNt(r = r, Nt = N[i], k = k)
  } # end of time-step loop
  N # return population size vector
}
```

Now, we execute this function with a few different settings to see the effects they are having:

```r
SIM_Growth <- SIM.Nt(N_0 = 0.1, t = 100, r = 0.1, k = 1) # N_0 < k --> growth
SIM_Decline <- SIM.Nt(N_0 =  1.9, t = 100, r = 0.1, k = 1) # N_0 > k --> decline
SIM_Rapid <- SIM.Nt(N_0 =  0.1, t = 100, r = 0.9, k = 1) # high population growth rate
SIM_Slow <- SIM.Nt(N_0 =  0.1, t = 100, r = 0.05, k = 1) # low population growth rate
```



The approach behaves as expected by us!

Indidividuals in Simulation

Next, we make the simulation more complex by running it for multiple individuals of one species. To do so, we require a new function that determines what happens at each time step:

```
dt <- function(b0, # background birth rate
               d0, # background death rate
               k, # carrying capacity
               N
               ){d0 + N * (b0 - d0)/k}
```

This function will build the backbone for our more complex simulations. It represents a **dynamic death rate**. Given a static birth rate and carrying capacity, it takes into account current population size and a background death rate to calculate a probability of death for each individual. You will see that this function cannot differentiate between individuals as it is not supplied with any individual-level characteristics yet. We will add this functionality in our next development step. For now, `dt()` calculates the same death probability for each individual alive at each time step.

Note that $b_0 + d_0 = 1$ **!**

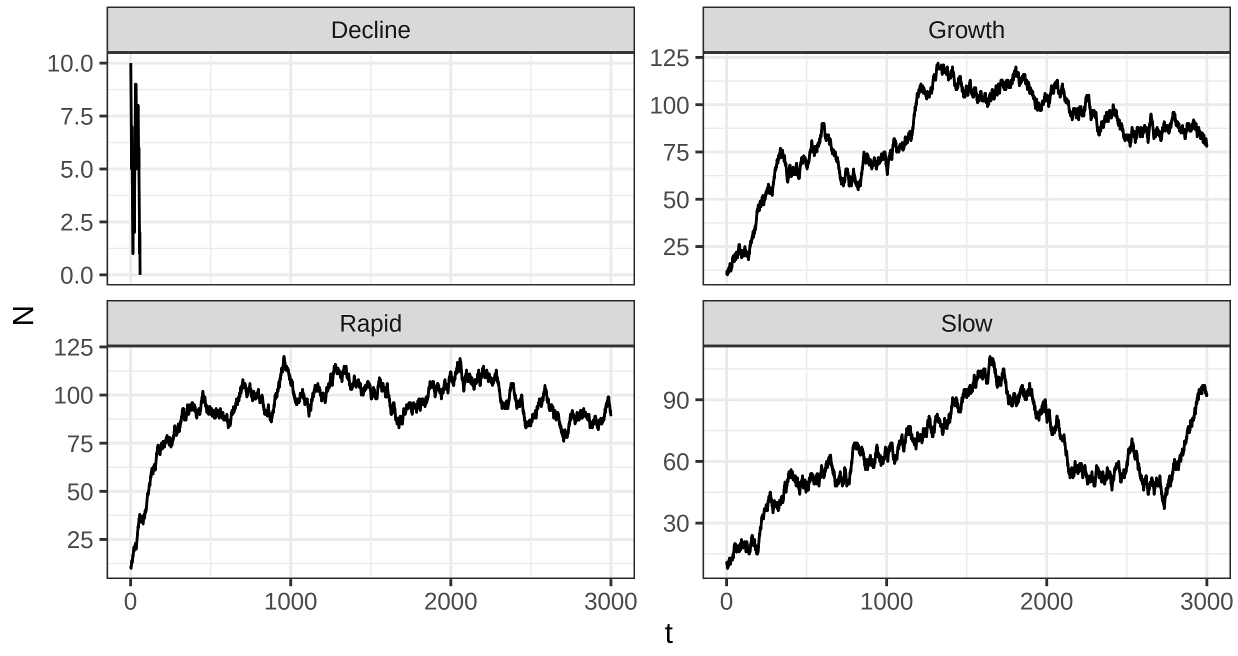
Let's iterate all of this over time:

```
SIM.Ind <- function(tsteps, # how many time steps to simulate for
                    d0, # background death rate
                    b0, # background birth rate
                    k, # carrying capacity
                    ID_df # data frame containing individuals
){
  ## object creation
  ### data frame to hold population level data
  Pop_df <- data.frame(N = nrow(ID_df), t = NA)
  ### progress bar
  pb <- txtProgressBar(max = (tsteps-1), style = 3)
  ## simulation loop over time steps
  for(t in 1:(tsteps-1)){
  Pop_df$t[t] <- t # save time to population data frame
  ### vectors for storing birth and death probabilities for each individual
  birth_prob <- rep(NA, nrow(ID_df))
  death_prob <- rep(NA, nrow(ID_df))
  ### loop over all individuals alive at time t
  for(i in 1:nrow(ID_df)){
    birth_prob[i] <- b0
    death_prob[i] <- dt(b0, d0, k, Pop_df$N[t])
  }
  names(birth_prob) <- names(death_prob) <- ID_df$ID
  ### identify which event happens, only one birth or death per timestep
  EventSample_vec <- paste( # a vector of Birth/Death and individual ID
    rep(c("Birth", "Death"), each = nrow(ID_df)),
    names(birth_prob), sep="_")
  event <- sample( # sample from event possibilities according to probabilities
    EventSample_vec,
    size = 1,
    prob = c(birth_prob, death_prob)
  )
  ### event evaluation
  event_eval <- strsplit(event, split = "_") # identifiers for event
  event_EV <- event_eval[[1]][1] # what is happening?
```

```r
  event_ID <- event_eval[[1]][2] # which individual is it happening to?
  #### Birth event
  if(event_EV == "Birth"){ # currently, basically cloning
    append_df <- ID_df[ID_df$ID == event_ID, ] # subset for individual
    append_df$ID <- max(ID_df$ID)+1 # assign new ID
    ID_df <- rbind(ID_df, append_df) # copy ID row to ID_df
  }
  #### Death event
  if(event_EV == "Death"){
    ID_df <- ID_df[ID_df$ID != event_ID, ] # delete individual from ID_df
  }
  ### recalculate population size
  Pop_df <- rbind(Pop_df, data.frame(N = nrow(ID_df), t = t+1))
  ### update progress bar
  setTxtProgressBar(pb, t)
  ### check if population has gone extinct
  if(Pop_df$N[t+1] == 0){warning("Population went extinct"); break}
  } # end of simulation loop
  ## return of data objects
  return(list(Individuals = ID_df, Populations = Pop_df))
}
```

Again, we execute this function with a few different settings to see the effects they are having:

```r
ID_df <- data.frame(ID = 1:10, Sp = rep("Sp1")) # we run this simulation for 7 individuals
Sim_Growth <- SIM.Ind(tsteps = 3e3, b0 = 0.7, d0 = 0.3, k = 100, ID_df = ID_df) # growth
Sim_Decline <- SIM.Ind(tsteps = 3e3, b0 = 0.4, d0 = 0.6, k = 100, ID_df = ID_df) # decline
Sim_Rapid <- SIM.Ind(tsteps = 3e3, b0 = 0.9, d0 = 0.1, k = 100, ID_df = ID_df) # fast
Sim_Slow <- SIM.Ind(tsteps = 3e3, b0 = 0.55, d0 = 0.45, k = 100, ID_df = ID_df) # slow
```



Unsurprisingly, our simulations worked once more as expected!

Spatial Component & Traits

Next, we implement the effects of the environment onto the dynamic death rate. This effect is calculated from the mismatch of an individuals trait value (`Tr`) and the singular value of the environment (`Env`). The severity of this effect is given by:

$$\frac{e^{(Tr-Env)^2}}{sd} \tag{1}$$

As can be seen from the above, the effect of the mismatch is modulated by an additional argument - `sd`. This can be understood as the niche breadth of each individual. The larger this number, the lower the impact of environment-trait mismatch.

Consequently, we update the dynamic death rate function `dt()` as follows:

```r
dt <- function(b0, d0, k, N, Tr, Env, sd){d0 + N * exp((Tr-Env)^2)/sd * (b0 - d0)/k}
```

Now that we implement traits and environmental effects, we also need to derive environmental conditions. For the time being, we do so using a linear function in one dimension:

```r
env.xy <- function(x = NULL){x}
```

Let's put this all together and iterate over simulation steps. From now on, I will only highlight bits where a function and its argument changes when compared to previous versions:

```r
SIM.Ind <- function(tsteps = 3e3, d0 = 0.2, b0 = 0.8, k = 100,
                    sd = 1, # niche breadth
                    ID_df = NULL # now needs to contain locations and trait values
){
  ## object creation
  ### data frame to hold population level data
  Pop_df <- data.frame(N = nrow(ID_df),
                       Gridcell = NA, # gridcell binning is not implemented yet
                       t = NA)
  ### progress bar
  pb <- txtProgressBar(max = (tsteps-1), style = 3)
  ## simulation loop over time steps
  for(t in 1:(tsteps-1)){
  Pop_df$t[t] <- t
  ### vectors for storing birth and death probabilities for each individual
  birth_prob <- rep(NA, nrow(ID_df))
  death_prob <- rep(NA, nrow(ID_df))
  ### loop over all individuals alive at time t
  for(i in 1:nrow(ID_df)){
    birth_prob[i] <- b0
    death_prob[i] <- dt(b0 = b0, d0 = d0, k = k, N = Pop_df$N[t],
                        Tr = ID_df$Trait[i], # trait value of individual i
                        Env = env.xy(x = ID_df$Location[i]), # environment at location[i]
                        sd = sd)
  }
  names(birth_prob) <- names(death_prob) <- ID_df$ID
  ### identify which event happens, only one birth or death per timestep
  EventSample_vec <- paste(rep(c("Birth", "Death"), each = nrow(ID_df)),
                           names(birth_prob), sep="_")
  event <- sample(EventSample_vec, size = 1, prob = c(birth_prob, death_prob))
  ## event evaluation
  event_eval <- strsplit(event, split = "_")
  event_ID <- event_eval[[1]][2]
  event_EV <- event_eval[[1]][1]
  if(event_EV == "Birth"){
    append_df <- ID_df[ID_df$ID == event_ID, ]
    append_df$ID <- max(ID_df$ID)+1
    ## individuals now resettle somewhere around their parent-individual
    movement <- rnorm(1, 0, 1) # this needs more parametrisation in future versions
    append_df$Location <- append_df$Location+movement
    ID_df <- rbind(ID_df, append_df)
  }
  if(event_EV == "Death"){
    ID_df <- ID_df[ID_df$ID != event_ID, ]
  }
  ## recalculate population size
  Pop_df <- rbind(Pop_df, data.frame(N = nrow(ID_df), Gridcell = NA, t = t+1))
  setTxtProgressBar(pb, t)
  if(Pop_df$N[t+1] == 0){warning("Population went extinct"); break}
  }
  return(list(Individuals = ID_df, Populations = Pop_df))
}
```

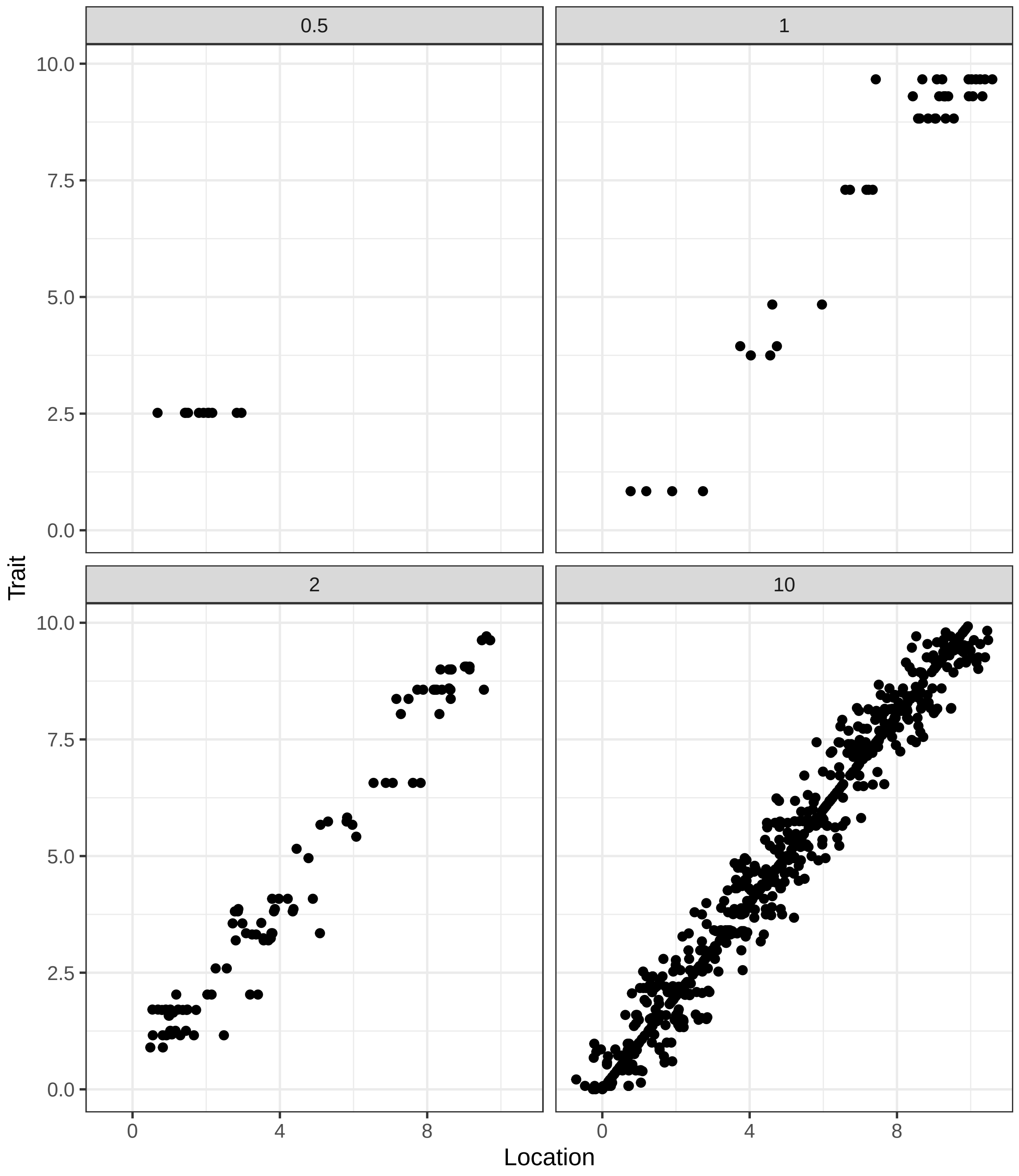
Now let's create a data frame with IDs, trait values, and locations. This time, we start with a much bigger starting population to make sure our randomly chosen traits and locations align for at least some individuals in such a way that our entire population doesn't collapse right away:

```
# data frame of individuals at start of simulation
ID_df = data.frame(ID = 1:1e3,
                   Trait = NA,
                   Location = NA)
# dummy values for traits
set.seed(42)
ID_df$Trait <- runif(n = nrow(ID_df), min = 0, max = 10)
# dummy locations
set.seed(42)
ID_df$Location <- runif(n = nrow(ID_df), min = 0, max = 10)
```

The big question we have to answer with this new simulation is how to parametrise the `sd` argument, so let's run the simulation with the same starting individuals with different values of `sd`:

```
SIM_0.5 <- SIM.Ind(sd = 0.5, # strong impact of environment
                     ID_df = ID_df)
SIM_1 <- SIM.Ind(sd = 1, # intermediate impact of environment
                     ID_df = ID_df)
SIM_2 <- SIM.Ind(sd = 2, # weaker impact of environment
                     ID_df = ID_df)
SIM_10 <- SIM.Ind(sd = 10, # very weak impact of environment
                     ID_df = ID_df)
```

To assess the effect, we now look at how strongly traits track the environment in the outputs



As we can see, setting `sd = 0.5`, all of our original individuals go extinct due to environment-trait mismatches except for one individual which ends up cloning itself and distributing itself around the same value in the environment as its trait (i.e. it's niche optimum). At `sd = 1`, some distinct individuals and trait profiles are present in our final population. As we increase `sd` further, the trait space and environmental spectrum are populated more densely.

Adding a Second Species

As we add a second species to our simulation, we do not take into account any specific interaction signs, strength, or spatial effect thereof. For now, our two species will simply be in competition with one another.

As a result, both our *dynamic death rate* `dt()` and *environment function* `env.xy()` remain unaltered:

```
dt <- function(b0, d0, k, N, Tr, Env, sd){d0 + N * exp((Tr-Env)^2)/sd * (b0 - d0)/k}
```

```
env.xy <- function(x = NULL){x}
```

Now on to implement our two species in the simulation. Again, I only annotate changes in the function:

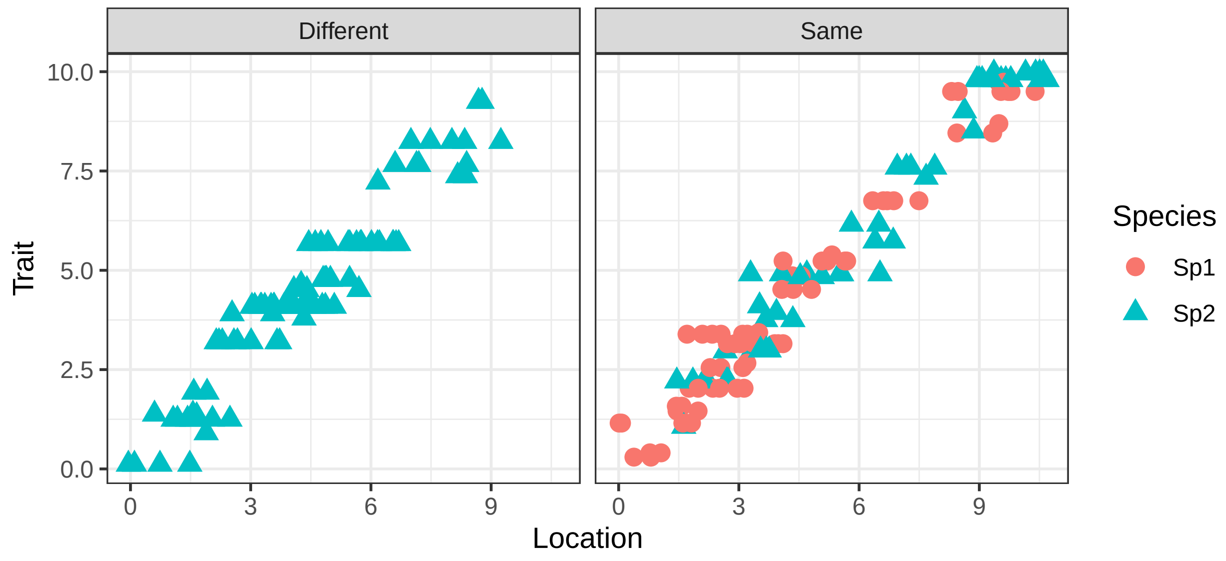
```
SIM.Comp <- function(tsteps = 3e3, d0 = 0.2, b0 = 0.8,
                     k_vec = 100, # carrying capacity for each species in ID_df
                     sd = 1,
                     ID_df = NULL # this now needs to hold different species
){
  ## object creation
  ### data frame to hold population level data
  Pop_df <- data.frame(N = nrow(ID_df),
                       Gridcell = NA, # gridcell binning is not implemented yet
                       t = NA)
  ### progress bar
  pb <- txtProgressBar(max = (tsteps-1), style = 3)
  ## simulation loop over time steps
  for(t in 1:(tsteps-1)){
  Pop_df$t[t] <- t
  ### vectors for storing birth and death probabilities for each individual
  birth_prob <- rep(NA, nrow(ID_df))
  death_prob <- rep(NA, nrow(ID_df))
  for(i in 1:nrow(ID_df)){
    birth_prob[i] <- b0
    death_prob[i] <- dt(b0 = b0, d0 = d0,
                        # select carrying capacity for species[i]
                        k = k_vec[which(names(k_vec) == ID_df$Species[i])],
                        N = Pop_df$N[t], # @Anna - should this not be the N of species i?
                        Tr = ID_df$Trait[i], Env = env.xy(x = ID_df$Location[i]), sd = sd)
  }
  names(birth_prob) <- names(death_prob) <- ID_df$ID
  ## event identification
  EventSample_vec <- paste(rep(c("Birth", "Death"), each = nrow(ID_df)),
                           names(birth_prob), sep="_")
  event <- sample(
    EventSample_vec,
    size = 1,
    prob = c(birth_prob, death_prob)
  )
  ## event evaluation
  event_eval <- strsplit(event, split = "_")
  event_ID <- event_eval[[1]][2]
  event_EV <- event_eval[[1]][1]
  if(event_EV == "Birth"){
    append_df <- ID_df[ID_df$ID == event_ID, ]
    append_df$ID <- max(ID_df$ID)+1
    movement <- rnorm(1, 0, 1) # this needs more parameterisation
    append_df$Location <- append_df$Location+movement
    ID_df <- rbind(ID_df, append_df)
  }
  if(event_EV == "Death"){
    ID_df <- ID_df[ID_df$ID != event_ID, ]
  }

  ## recalculate population size
  Pop_df <- rbind(Pop_df, data.frame(N = nrow(ID_df), Gridcell = NA, t = t+1))
  setTxtProgressBar(pb, t)
  if(Pop_df$N[t+1] == 0){warning("Population went extinct"); break}
  }
  return(list(Individuals = ID_df, Populations = Pop_df))
}
```

This time, I want to look at how different relative carrying capacities and trait expression between species affect the simulation.

Differing Carrying Capacities

First, let's alter carrying capacities while keeping everything else set to the default.

```
# individual data frame
ID_df = data.frame(ID = 1:1e3,
                   Trait = NA,
                   Location = NA,
                   Species = sample(c("Sp1", "Sp2"), size = 1e3, replace = TRUE))

# dummy values for traits
set.seed(42)
ID_df$Trait <- runif(n = nrow(ID_df), min = 0, max = 10)
# dummy locations
set.seed(42)
ID_df$Location <- runif(n = nrow(ID_df), min = 0, max = 10)

# same carrying capacity:
k_vec <- c(200, 200)
names(k_vec) <- c("Sp1", "Sp2")
SIM_2SpecSamek <- SIM.Comp(tsteps = 3e3, d0 = 0.2, b0 = 0.8,
                      k_vec = k_vec, # carrying capacity
                      sd = 1, ID_df = ID_df)

# different carrying capacity:
k_vec <- c(50, 200)
names(k_vec) <- c("Sp1", "Sp2")
SIM_2SpecDiffk <- SIM.Comp(tsteps = 3e3, d0 = 0.2, b0 = 0.8,
                      k_vec = k_vec, # carrying capacity
                      sd = 1, ID_df = ID_df)
```



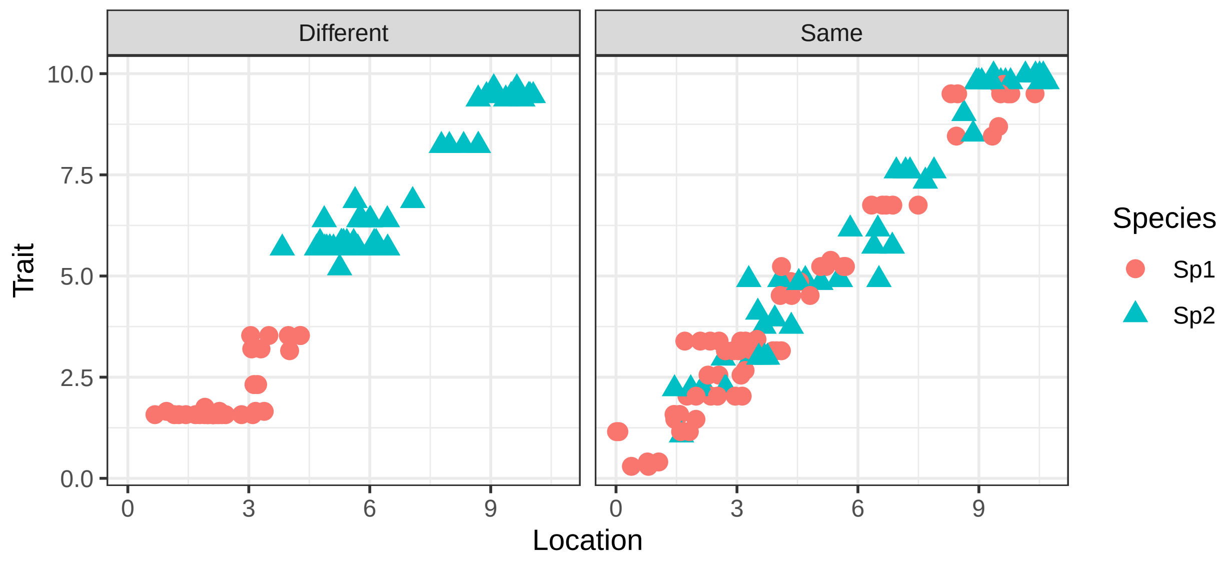
Neat! When we have a much higher carrying capacity for `"Sp2"` than `"Sp1"`, only species 2 ended up surviving the simulation. When there is no difference in carrying capacity, both species seem to come out of the simulation with roughly the same population sizes.

Differing Trait Expressions

Now let's assess how different trait motifs for each species affect the simulation outcome. Here, I se trait motifs for each species with overlapping distributions:

```
# individual data frame
ID_df = data.frame(ID = 1:1e3,
                   Trait = NA,
                   Location = NA,
                   Species = sample(c("Sp1", "Sp2"), size = 1e3, replace = TRUE))

# dummy values for traits
set.seed(42)
ID_df$Trait[ID_df$Species == "Sp1"] <- runif(n = length(ID_df$Trait[ID_df$Species == "Sp1"]),
                                             min = 0, max = 5.5)
ID_df$Trait[ID_df$Species == "Sp2"] <- runif(n = length(ID_df$Trait[ID_df$Species == "Sp2"]),
                                             min = 4.5, max = 10)
# dummy locations
set.seed(42)
ID_df$Location <- runif(n = nrow(ID_df), min = 0, max = 10)
# carrying capacity for each species
k_vec <- c(200, 200)
names(k_vec) <- c("Sp1", "Sp2")
SIM_2SpecDiffTr <- SIM.Comp(tsteps = 3e3, d0 = 0.2, b0 = 0.8,
                      k_vec = k_vec, # carrying capacity
                      sd = 1, ID_df = ID_df)
```

When plotting, I compare this simulation to the one ran for equal carrying capacities but equal, random traits above:



Cool. When we have species that prefer different environments (as defined by their trait-value), we see them being selected for in their preferred environments. When traits are random, so is space-utilisation by each species.

Binning by Gridcells & Limitting Environmental Space Range

To make this simulation output ready for network inference methodology, we need to represent the data on a grid-by-grid cell. Also, the movement of the offspring in previous iterations of this simulation is capable of pushing occurrences into environmental ranges we did not intend for (such as negative numbers). Here, I address both.

Implementing these "fixes" requires no change to our *dynamic death rate* `dt()` and *environment function* `env.xy()` and so they remain unaltered:

```r
dt <- function(b0, d0, k, N, Tr, Env, sd){d0 + N * exp((Tr-Env)^2)/sd * (b0 - d0)/k}
```

```r
env.xy <- function(x = NULL){x}
```

Now on to implement our grid-cell binning an environment limiting. Again, I only annotate changes in the function:

```r
SIM.Comp <- function(tsteps = 3e3, d0 = 0.2, b0 = 0.8,
                     k_vec = 100,
                     sd = 1,
                     ID_df = NULL,
                     Env_range = c(0, 10), # range of permitted environmental values
                     n_Grid = 100 # how many gridcells to draw
){
  ## object creation
  ### data frame that stores grid coordinates
  ### -> GridX holds the left-most starting-point of each gridcell
  grids_df <- data.frame(
    GridID = 1:n_Grid,
    GridX = seq(from = Env_range[1], to = Env_range[2], length = n_Grid+1)[-(n_Grid+1)]
  )
  ### data frame to hold population level data
  ### -> identify gridcell for each individual
  GridsID_vec <- sapply(ID_df$Location,
                        FUN = function(x){tail(which(x >= grids_df$GridX), 1)})
  ### -> store populations of each species binned by gridcell as a list element
  Pop_ls <- list(table(GridsID_vec, ID_df$Species))
  Pop_df <- data.frame(N = nrow(ID_df),
                       t = NA)
  ### progress bar
  pb <- txtProgressBar(max = (tsteps-1), style = 3)
  ## simulation loop over time steps
  for(t in 1:(tsteps-1)){
  Pop_df$t[t] <- t
  names(Pop_ls)[length(Pop_ls)] <- t
  ### vectors for storing birth and death probabilities for each individual
  birth_prob <- rep(NA, nrow(ID_df))
  death_prob <- rep(NA, nrow(ID_df))
  for(i in 1:nrow(ID_df)){
    birth_prob[i] <- b0
    death_prob[i] <- dt(b0 = b0, d0 = d0,
                        # select carrying capacity for species[i]
                        k = k_vec[which(names(k_vec) == ID_df$Species[i])],
                        N = Pop_df$N[t], # @Anna - should this not be the N of species i?
                        Tr = ID_df$Trait[i], Env = env.xy(x = ID_df$Location[i]), sd = sd)
  }
  names(birth_prob) <- names(death_prob) <- ID_df$ID
  ## event identification
  EventSample_vec <- paste(rep(c("Birth", "Death"), each = nrow(ID_df)),
                           names(birth_prob), sep="_")
  event <- sample(EventSample_vec, size = 1, prob = c(birth_prob, death_prob))
  ## event evaluation
  event_eval <- strsplit(event, split = "_")
  event_ID <- event_eval[[1]][2]
  event_EV <- event_eval[[1]][1]
  if(event_EV == "Birth"){
    append_df <- ID_df[ID_df$ID == event_ID, ]
    append_df$ID <- max(ID_df$ID)+1
    movement <- rnorm(1, 0, 1) # this needs more parametrisation
    newloc <- append_df$Location+movement
    ## ensuring species don't disperse beyond the environmental limit
    newloc <- ifelse(newloc<Env_range[1], Env_range[1], newloc)
    newloc <- ifelse(newloc>Env_range[2], Env_range[2], newloc)
    append_df$Location <- newloc
    ID_df <- rbind(ID_df, append_df)
  }
  if(event_EV == "Death"){ID_df <- ID_df[ID_df$ID != event_ID, ]}
  ### -> identify gridcell for each individual
  GridsID_vec <- sapply(ID_df$Location,
                        FUN = function(x){tail(which(x >= grids_df$GridX), 1)})
  Pop_ls <- c(Pop_ls,
              list(table(GridsID_vec, ID_df$Species)))
  names(Pop_ls)[length(Pop_ls)] <- t+1
  ## recalculate population size
  Pop_df <- rbind(Pop_df, data.frame(N = nrow(ID_df), t = t+1))
  setTxtProgressBar(pb, t)
  if(Pop_df$N[t+1] == 0){warning("Population went extinct"); break}
  }
  return(list(Individuals = ID_df, Populations = Pop_df,
              GriddedPopulations = Pop_ls # returning the gridded population data
              ))
}
```

Now, let’s see how this works out. Here, I use the exact same simulation specification as I have done in the previous development step when I investigated different trait expressions per species. I do so to check whether I can see how the initially uniformly distributed species settle into their respective “optimal” grid cells over time:

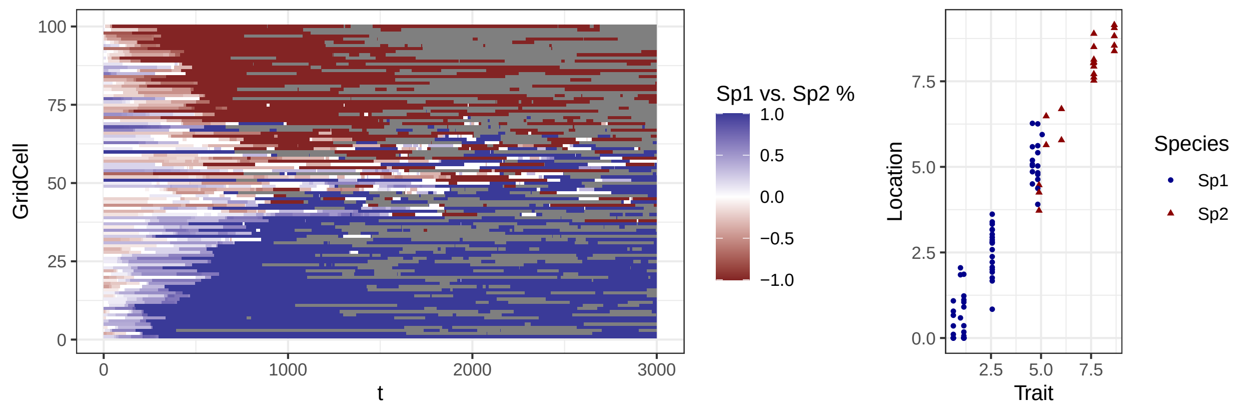
```
# individual data frame
ID_df = data.frame(ID = 1:1e3,
                   Trait = NA,
                   Location = NA,
                   Species = sample(c("Sp1", "Sp2"), size = 1e3, replace = TRUE))

# dummy values for traits
set.seed(42)
ID_df$Trait[ID_df$Species == "Sp1"] <- runif(n = length(ID_df$Trait[ID_df$Species == "Sp1"]),
                                             min = 0, max = 5.5)
ID_df$Trait[ID_df$Species == "Sp2"] <- runif(n = length(ID_df$Trait[ID_df$Species == "Sp2"]),
                                             min = 4.5, max = 10)
```

```
# dummy locations
set.seed(42)
ID_df$Location <- runif(n = nrow(ID_df), min = 0, max = 10)
# carrying capacity for each species
k_vec <- c(200, 200)
names(k_vec) <- c("Sp1", "Sp2")
SIM_Grid <- SIM.Comp(tsteps = 3e3, d0 = 0.2, b0 = 0.8,
                     k_vec = k_vec, sd = 1, ID_df = ID_df,
                     Env_range = c(0, 10), # range of permitted environmental values
                     n_Grid = 100 # how many gridcells to draw
                    )
```

Now I extract the data and calculate the difference in population size (as percentages) between the two species in each grid cell at each time step (a value of 100 identifies a grid cell entirely populated by species 1):

```
## extracting the data
GridPop_ls <- lapply(names(SIM_Grid$GriddedPopulations), FUN = function(x){
  ## difference per cell in percent
  DiffPerc <- (SIM_Grid$GriddedPopulations[[x]][,1]-
                 SIM_Grid$GriddedPopulations[[x]][,2])/
      (SIM_Grid$GriddedPopulations[[x]][,1]+
         SIM_Grid$GriddedPopulations[[x]][,2])
  names(DiffPerc) <- as.numeric(rownames(SIM_Grid$GriddedPopulations[[x]]))
  ## data frame for reporting results
  Report_df <- data.frame(
    GridCell = 1:100,
    DiffPerc = NA,
    t = x)
  ## filling data frame with results
  Report_df$DiffPerc[match(names(DiffPerc), Report_df$GridCell)] <- DiffPerc
  Report_df
  }
  )
## creating final data frame for plotting
GridPop_df <- do.call(rbind, GridPop_ls)
GridPop_df$t <- as.numeric(GridPop_df$t)
GridPop_df$GridCell <- as.numeric(GridPop_df$GridCell)
```

Finally, I plot the grid cells through time side-by-side with the individuals of the final time step of the above simulation:



They agree with each other, but now we can also track the populations through time and space!

Gillespie Algorithm

So far, every iteration of our simulation framework evaluated one event per time-step. That is not realistic - multiple events happen to multiple individuals roughly at the same time and not in neat intervals. The number of events that happen scales with population size - the more individuals, the more events happen. But how do we determine just how many events happen at each time step? Simple. We don't. Instead, we make use of the Gillespie formulation which can be used to approximate the time which each event takes to complete. This has been implemented in a previous study by Anna.

Implementing this has nothing to do with our *dynamic death rate* `dt()` and *environment function* `env.xy()` and so they remain unaltered:

```
dt <- function(b0, d0, k, N, Tr, Env, sd){d0 + N * exp((Tr-Env)^2)/sd * (b0 - d0)/k}
```

```
env.xy <- function(x = NULL){x}
```

To implement the Gillespie algorithm, biological time in the simulation is advanced by a random intervals drawn from an exponential distribution whose rate is equal to 1/E where E is the sum of the rates of all possible events.

This leads to many more simulation timesteps being run than before (each event is still one simulation step - we also refer to this as simulation time). So, we have to decide at what biological time intervals to sample and until which biological time to run our simulation. Consequently, our function gets two new arguments (`t_inter` and `t_max`) and loses the previous `tsteps` argument - we will run simulations until the biological time in `t_max` is reached (unless all our species go extinct before then).

Lastly, I didn't like how much migration potential individuals received on birth so far. As a matter of fact, I commented in earlier code version that this mechanisms required further parametrisation. I have consequently established the `migration` argument in the `SIM.Comp()` function call. This numeric argument controls the standard deviation of the mean-0 normal distribution from which the migration direction and magnitude is drawn.

Again, I only annotate the new bits to our simulation framework

```
SIM.Comp <- function(d0 = 0.2, b0 = 0.8,
                     k_vec = 100,
                     t_max = 100, # now defining until what biological time to run
                     t_inter = 0.5, # at what time intervals to sample
                     sd = 1,
                     ID_df = NULL,
                     Env_range = c(0, 10),
                     n_Grid = 100,
                     migration = 0.1
){
  ## object creation
  ### data frame that stores grid coordinates,
  grids_df <- data.frame(
    GridID = 1:n_Grid,
    GridX = seq(from = Env_range[1], to = Env_range[2], length = n_Grid+1)[-(n_Grid+1)]
  )
  ### data frame to hold population level data
  GridsID_vec <- sapply(ID_df$Location,
                        FUN = function(x){tail(which(x >= grids_df$GridX), 1)})
  ### populations of each species binned by gridcell as a list element
  Pop_ls <- list(table(GridsID_vec, ID_df$Species))
```

```r
Pop_df <- data.frame(N = nrow(ID_df),
                     t = NA)
### setting star times
t <- 1 # start time at 1
Pop_df$t[nrow(Pop_df)] <- 1
names(Pop_ls)[length(Pop_ls)] <- 1
### progress bar
pb <- txtProgressBar(max = t_max, style = 3)
## simulation loop over time steps
while(t < t_max){ # now a while loop
## vectors for storing birth and death probabilities for each individual
birth_prob <- rep(NA, nrow(ID_df))
death_prob <- rep(NA, nrow(ID_df))
for(i in 1:nrow(ID_df)){
  birth_prob[i] <- b0
  death_prob[i] <- dt(b0 = b0, d0 = d0,
          k = k_vec[which(names(k_vec) == ID_df$Species[i])],
          N = nrow(ID_df), # @Anna - should this not be the N of species i?
          #' like so: #sum(ID_df$Species == ID_df$Species[i])
          Tr = ID_df$Trait[i], Env = env.xy(x = ID_df$Location[i]), sd = sd)

}
names(birth_prob) <- names(death_prob) <- ID_df$ID
## event identification
EventSample_vec <- paste(rep(c("Birth", "Death"), each = nrow(ID_df)),
                         names(birth_prob), sep="_")
event <- sample(
  EventSample_vec,
  size = 1,
  prob = c(birth_prob, death_prob)
)
## event evaluation
event_eval <- strsplit(event, split = "_")
event_ID <- event_eval[[1]][2]
event_EV <- event_eval[[1]][1]
if(event_EV == "Birth"){
  append_df <- ID_df[ID_df$ID == event_ID, ]
  append_df$ID <- max(ID_df$ID)+1
  movement <- rnorm(1, 0, migration) # new function argument
  newloc <- append_df$Location+movement
  ## ensuring species don't disperse beyond the environmental limit
  newloc <- ifelse(newloc<Env_range[1], Env_range[1], newloc)
  newloc <- ifelse(newloc>Env_range[2], Env_range[2], newloc)
  append_df$Location <- newloc
  ID_df <- rbind(ID_df, append_df)
}
if(event_EV == "Death"){
  ID_df <- ID_df[ID_df$ID != event_ID, ]
}

## Gillespie time
### identify by how much time advances
tadvance <- rexp(1, rate = sum(c(birth_prob, death_prob)))
```

```r
  t <- t + tadvance
  ### record data only if interval is met
  if(t - Pop_df$t[nrow(Pop_df)] >= t_inter){
    GridsID_vec <- sapply(ID_df$Location,
                          FUN = function(x){tail(which(x >= grids_df$GridX), 1)})
  Pop_ls <- c(Pop_ls,
              list(table(GridsID_vec, ID_df$Species))
              )
  names(Pop_ls)[length(Pop_ls)] <- t
  ### recalculate population size
  Pop_df <- rbind(Pop_df, data.frame(N = nrow(ID_df), t = t+1))
  }
  ## update progress
  setTxtProgressBar(pb, t)
  if(nrow(ID_df) == 0){warning("All species went extinct"); break}
  }
  return(list(Individuals = ID_df, Populations = Pop_df,
              GriddedPopulations = Pop_ls
              ))
}
```

Now to run this simulation. This time, I increase the value in `sd` to make the environment less harsh on our species, but I also establish initial trait values for each species no longer as drawn randomly from a uniform distribution, but from normal distributions. I also increase the carrying capacities for both species (they are still equal):

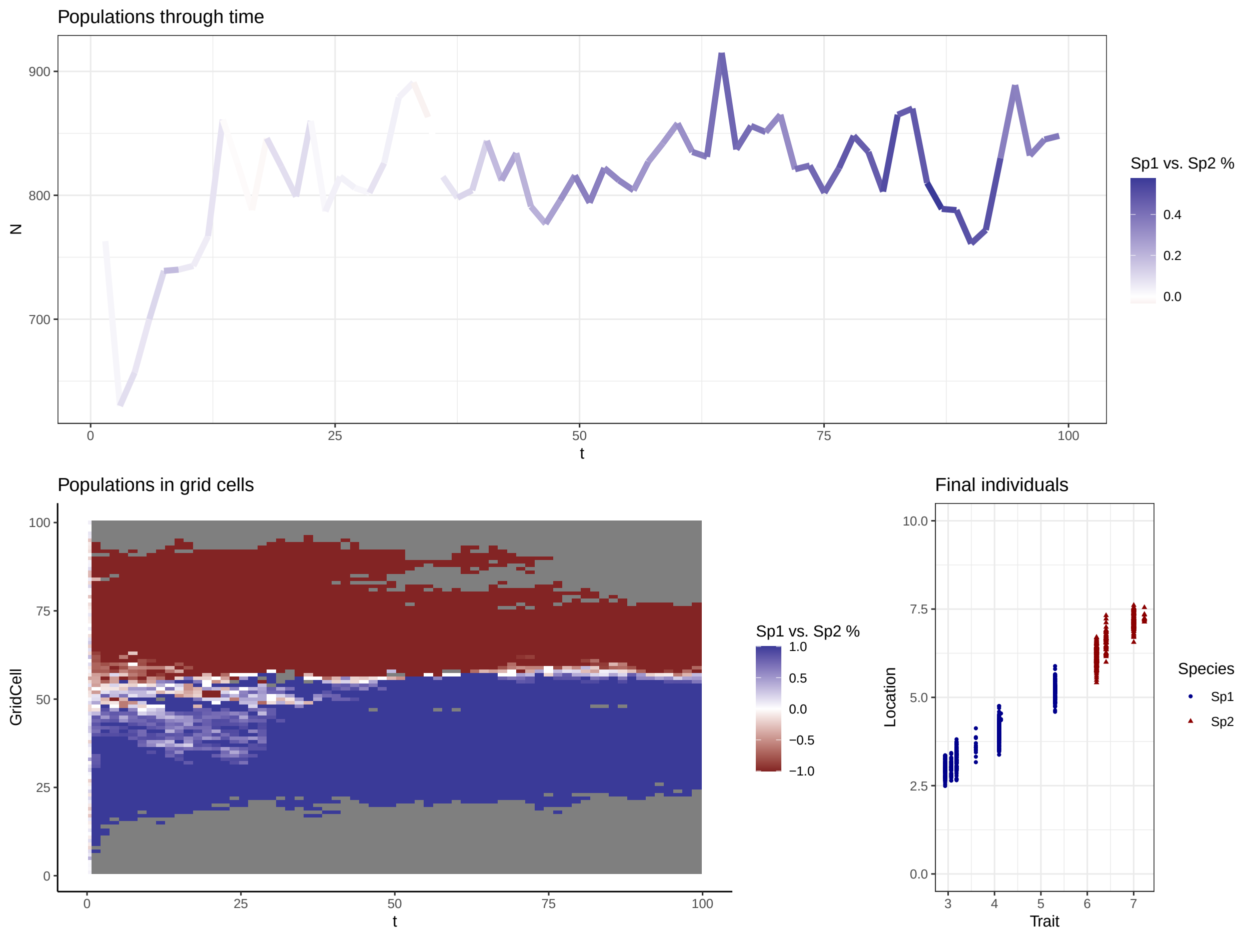
```r
# individual data frame
ID_df = data.frame(ID = 1:4e3,
                   Trait = NA,
                   Location = NA,
                   Species = sample(c("Sp1", "Sp2"), size = 4e3, replace = TRUE))

# dummy values for traits
set.seed(42)
ID_df$Trait[ID_df$Species == "Sp1"] <- rnorm(n = length(ID_df$Trait[ID_df$Species == "Sp1"]),
                                             mean = 3.5, sd = 1)
ID_df$Trait[ID_df$Species == "Sp2"] <- rnorm(n = length(ID_df$Trait[ID_df$Species == "Sp2"]),
                                             mean = 6.5, sd = 1)
# dummy locations
set.seed(42)
ID_df$Location <- runif(n = nrow(ID_df), min = 0, max = 10)
# carrying capacity for each species
k_vec <- c(400, 400)
names(k_vec) <- c("Sp1", "Sp2")
# running simulation
SIM_Gillespie <- SIM.Comp(d0 = 0.2, b0 = 0.8,
                    t_max = 200, # now defining until what biological time to run
                    t_inter = 0.5, # at what time intervals to sample
                    k_vec = k_vec,
                    sd = 3, # higher habitat suitability
                    ID_df = ID_df,
                    Env_range = c(0, 10), n_Grid = 100,
                    migration = 0.1 # shorter migration distances
                   )
```

Ok… this ran for a long time. That makes sense though - we simulated many more iterations (i.e. longer simulation time) than before.

Now on to extract the relevant data (this is just a beefier version than what I did in the previous simulation development section):

```r
## extracting the data
GridPop_ls <- lapply(names(SIM_Gillespie$GriddedPopulations), FUN = function(x){
  ## difference per cell in percent
  if(ncol(SIM_Gillespie$GriddedPopulations[[x]]) == 2){
   DiffPerc <- (SIM_Gillespie$GriddedPopulations[[x]][,1]-
                  SIM_Gillespie$GriddedPopulations[[x]][,2])/
      (SIM_Gillespie$GriddedPopulations[[x]][,1]+
         SIM_Gillespie$GriddedPopulations[[x]][,2])
  }
  if(ncol(SIM_Gillespie$GriddedPopulations[[x]]) == 1){
    if(colnames(SIM_Gillespie$GriddedPopulations[[x]]) == "Sp2"){
      DiffPerc <- rep(-1, nrow(SIM_Gillespie$GriddedPopulations[[x]]))
    }
    if(colnames(SIM_Gillespie$GriddedPopulations[[x]]) == "Sp1"){
      DiffPerc <- rep(1, nrow(SIM_Gillespie$GriddedPopulations[[x]]))
    }
  }
  names <- as.numeric(rownames(SIM_Gillespie$GriddedPopulations[[x]]))
  names(DiffPerc) <- names

  Report_df <- data.frame(
    GridCell = 1:100,
    DiffPerc = NA,
    t = x)
  Report_df$DiffPerc[match(names(DiffPerc), Report_df$GridCell)] <- DiffPerc

  ## absolute population size for entire environment
  Abs_df <- data.frame(Sp1 = 0,
             Sp2 = 0,
             DiffPerc = 0,
             N = 0,
             t = x)
  Pops <- colSums(SIM_Gillespie$GriddedPopulations[[x]])
  Abs_df[,match(names(Pops), colnames(Abs_df))] <- Pops

  if(sum(!is.na(Abs_df[,1:2])) == 2){
   DiffPerc <- (Abs_df[1]- Abs_df[2])/(Abs_df[1] + Abs_df[2])
  }else{
    if(!is.na(Abs_df[,1])){
      DiffPerc <- 1
    }
    if(!is.na(Abs_df[,2])){
      DiffPerc <- -1
    }
  }
  Abs_df$DiffPerc <- as.numeric(DiffPerc)
  Abs_df$N <- sum(Abs_df[,1:2], na.rm = TRUE)

  ## returning data
  list(Diff = Report_df,
       Abs = Abs_df)
  }
  )
## creating final data frame for plotting
GridPop_df <- do.call(rbind, lapply(GridPop_ls, "[[", 1))
GridPop_df$t <- as.numeric(GridPop_df$t)
GridPop_df$GridCell <- as.numeric(GridPop_df$GridCell)
AbsPop_df <- do.call(rbind, lapply(GridPop_ls, "[[", 2))
AbsPop_df <- data.frame(sapply(AbsPop_df, as.numeric))
```

Finally, I plot the total population size through time (coloured by percentages of the population size of each species), the relative populations of our species (relative to each other) in each grid cell through time, and the location and trait expression of our final individuals:



Final Simulation Framework

Rationale

Input

Output

Parameters / Function Arguments