

# PhD Chapter 4: Simulation Framework

Erik Kusch; Anna Vinton

2022-06-24

## Contents

<b>Introduction</b>	<b>1</b>
Scientific Goals . . . . .	1
R Preparations . . . . .	1
<b>Building the Simulation</b>	<b>2</b>
One-Species Through Time . . . . .	2
Individuals in Simulation . . . . .	3
Spatial Component & Traits . . . . .	5
Adding a Second Species . . . . .	8

## 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 Preparations

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

```
library(ggplot2) # for visualisations
```

# 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**:

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

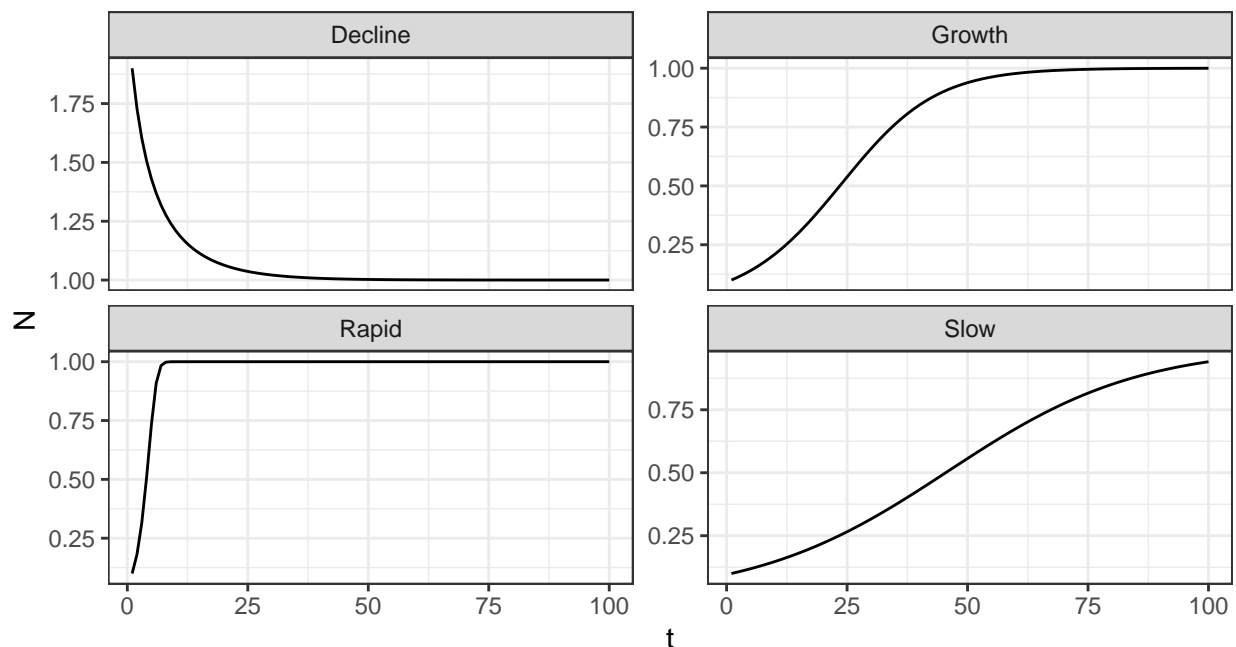
With this approach, a population will increase in size towards the carrying capacity if its starting population size ( $N_0$ )  $N_0 < k$ . If  $N_0 > k$  the population size will decrease towards the carrying capacity.

This function now needs to be iterated over time:

```
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:

```
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!

## Individuals 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?
```

```

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))
}

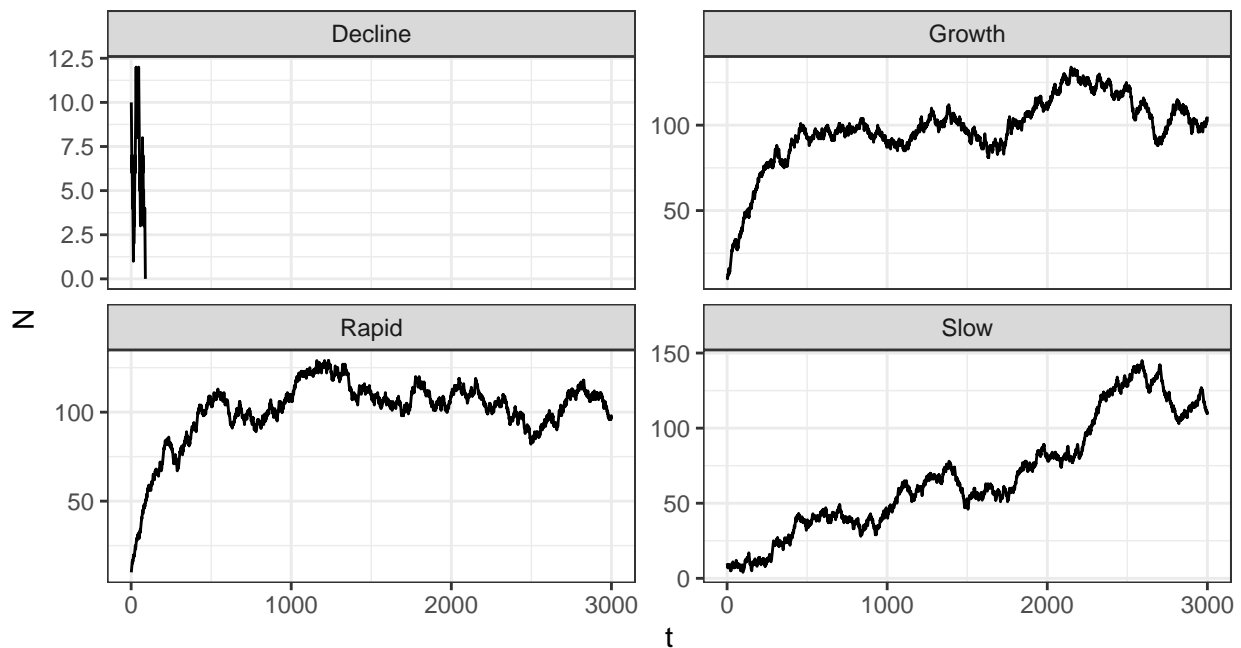
```

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

```

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 individual's trait value ( $Tr$ ) and the singular value of the environment ( $Env$ ). The severity of this effect is given by:

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

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:

```
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:

```
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:

```
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)

```

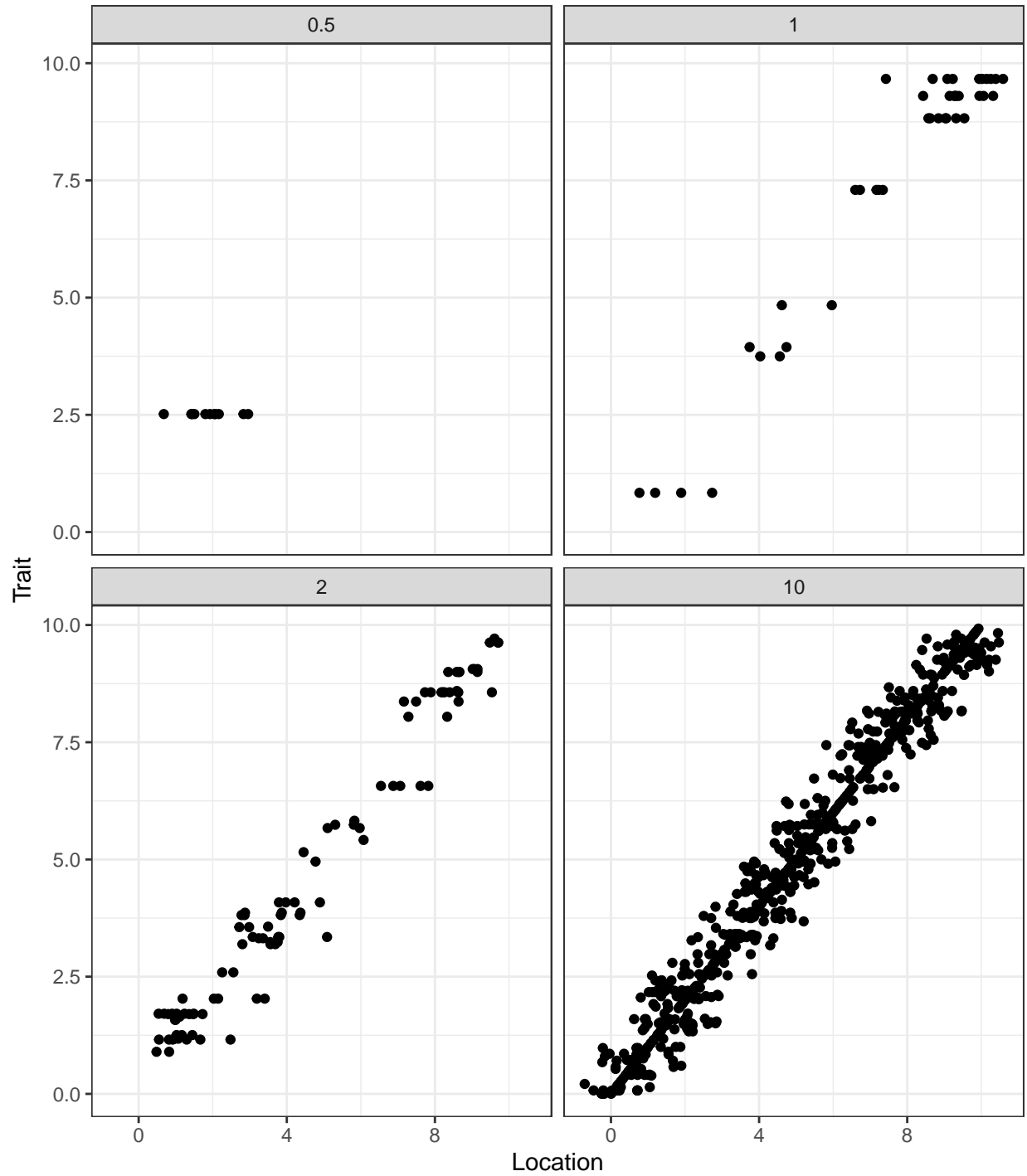
The big question we have to answer with this new simulation is how to parametrise the `sd` argument, so let's run the simulation with the same starting individuals with different values of `sd`:

```

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.

```

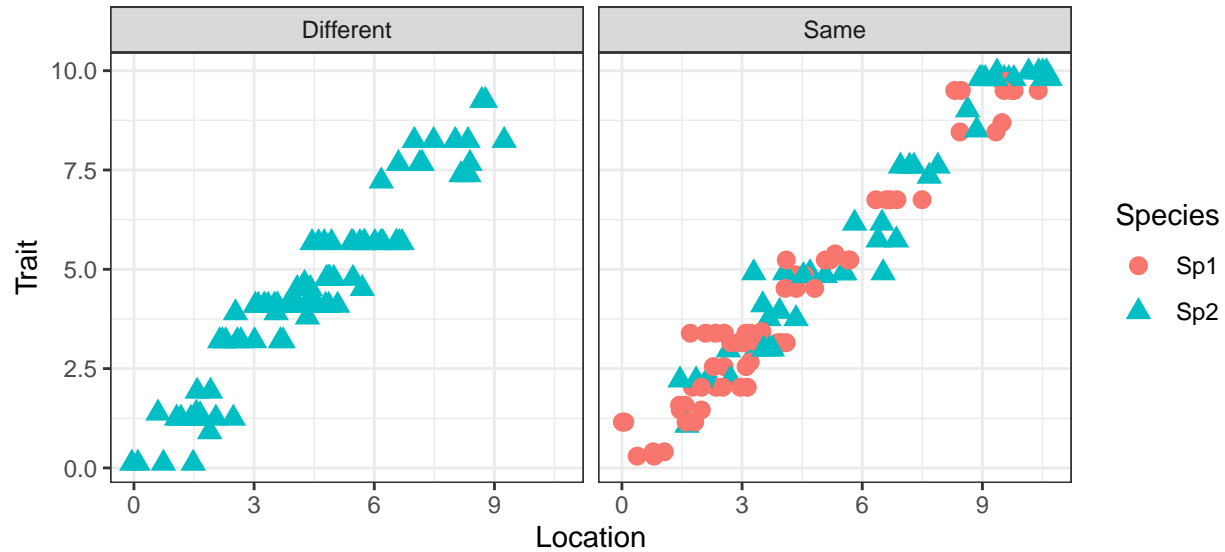
# individual data frame
ID_df = data.frame(ID = 1:1e3,
                  Trait = NA,
                  Location = NA,
                  Species = sample(c("Sp1", "Sp2"), size = 1e3, replace = TRUE))

# 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 use trait motifs for each species with overlapping distributions:

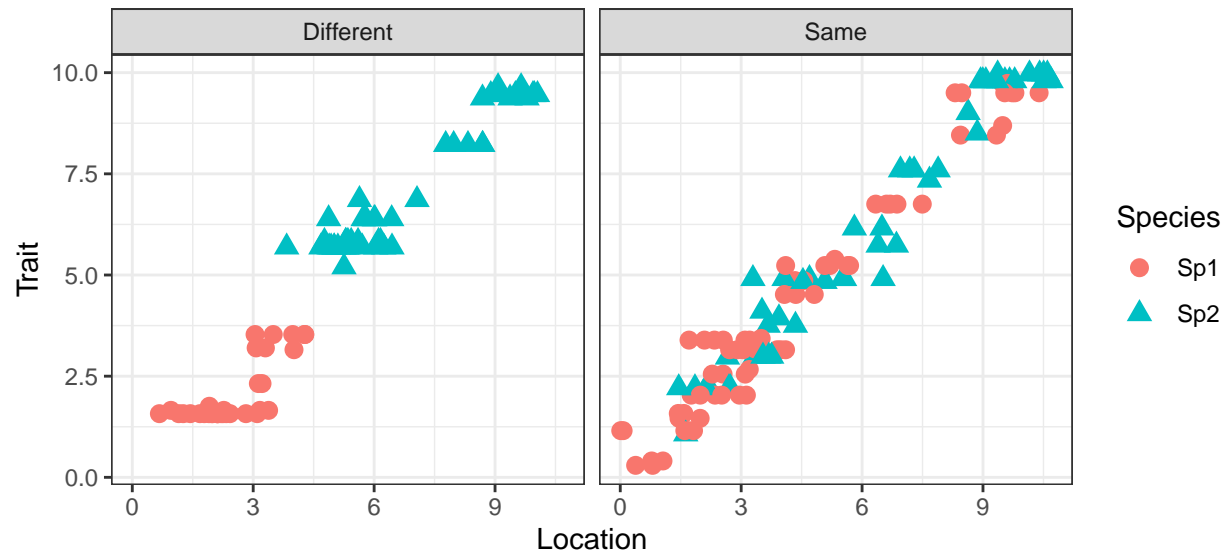
```
# individual data frame
ID_df = data.frame(ID = 1:1e3,
                   Trait = NA,
                   Location = NA,
                   Species = sample(c("Sp1", "Sp2"), size = 1e3, replace = TRUE))

# 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.