ZO 5304 Population Ecology

Ana Payo Payo

2022-11-01

¡Hola!

Welcome to ZO 5304 Population Ecology practical! Today we will work through a series of exercises to estimate population size and vital rates in wild populations in R. R is a (free) software that many scientists used for statistical computing and graphics, e.g., analysing and plotting data. Many of you will be familiar with R and some might even be advanced users already. Because R is such an important tool for population ecologist, and because I want each of you to learn from this tutorial, please follow the sections according to your confidence with R. Consequently, learning outcomes of this tutorial will be different depending on your knowledge of R. Feel free to change levels along the tutorial.

- If this is your first-time using R, complete the sections marked by a 🐣 symbol. Also, make sure to ask demonstrators for help as learning to use R can be very challenging at first!
- If this is your second time using R, complete the sections marked by a 🐻 symbol. Also, make sure to ask demonstrators for help as learning to use R can be very challenging!
- If you have used R several times on your own but don't feel super confident yet, complete the sections marked by a 🐯 symbol.
- If you feel super confident using R, complete the sections marked by a 🦾 symbol.

Warm-up in R

🐣 This is my first-time using R.

🐻 This is my second time using R.

🐯 I have used R several times, but I don't feel super confident yet.

🦾 I feel super confident using R.

🐣 Let's start with a short introduction to R. We will use an online book written by Dr Alex Douglas and colleagues, available in the following link: https://intro2r.com/. If you are working on your own machine, start at chapter section 1.1 to install R and RStudio (Getting starter with R and RStudio). If you are on a Uni PC, go directly to 1.3 (RStudio Orientation). Go through sections 1.3,1.5, 1.7 and 1.10. Then proceed with completing section 1.14 (Exercise 1). Ask demonstrators for help at any time.

🐼Let's start with a short refresher of R basics. We will use an online book written by Dr Alex Douglas and colleagues, available here https://intro2r.com/. Since you have already used R once before and know how the interface looks like, jump to section 2. Go through each section and complete section 2.7 (Exercise 2). Ask demonstrators for help at any time.

🦁 Let's start with a short R refresher. We will use an online book written by Dr Alex Douglas and colleagues, available here https://intro2r.com/. Since you have already used R before, jump to section 3.7 (Exercise 3) and go through the exercises. Feel free to use any section of the book for support or ask a demonstrator if anything is unclear.

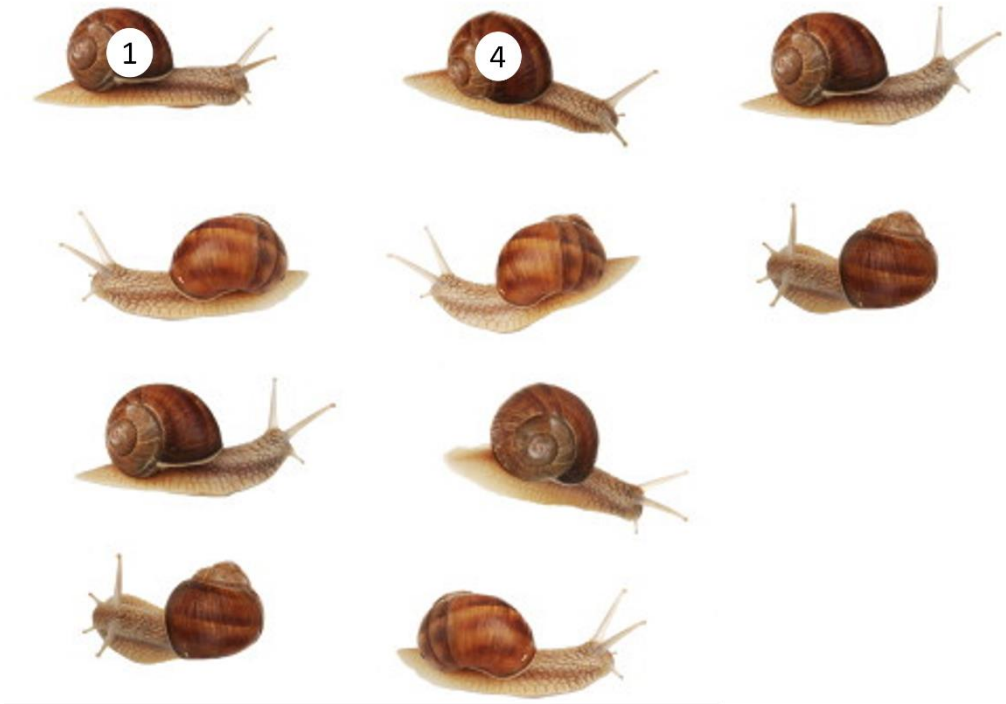
🐆Let's see if you R skills are up to the test! We will use an online book written by Dr Alex Douglas and colleagues, available here https://intro2r.com/. Since you are an experience R user, jump to section 4.6 (Exercise 4) and go through the exercises. Feel free to use any section of the book for support or ask a demonstrator if you need a refresher.

1.Estimating population size using capture-mark-release-recapture methods: The Lincoln-Petersen estimator

For most species, direct counts of population size are impossible because the species are secretive, cryptic, or inaccessible. However, total population size can be estimated (with confidence limits) from a statistical analysis of a smaller sample that is captured, marked, released, and recaptured later. A number of these 'capture-mark-release-recapture' (CMRR) methods are available at different levels of sophistication, and the choice of method should be matched to the ecology to the target organism and the resources of the investigator. The simplest method, the Lincoln-Petersen method, involves a single marking, and a single recapture. We want to estimate $\widehat{N_1}$ which is the population size at time t=1. To do that, in time 1, we capture, mark and release back to the population n_1 individuals. Then, we go back to the field in t=2 and we capture n_2 individuals. From those n_2 individuals we check how many m_2 individuals have one of the marks we put in t=1.

For instance, imagine we want to estimate the population of snails living in a given area of the forest. We go there one day (time 1), we capture, mark and release back to the population n_1=5 individuals.

Then, we go back to the field another day (t=2) and we capture n_2=10 individuals. From those n_2 individuals we see that m_2 =2 individuals are marked from our previous visit.



The method assumes that the proportion of individuals marked in a population at any time does not change. Therefore, the proportion of individuals marked in t1 should be the same to the proportion of individuals tagged in t2 and the following should be true:

$$\frac{n_1}{\hat{N}_1} = \frac{m_2}{n_2}$$

From which we can deduce that :

$$\hat{N}_1 = \frac{n_1 * n_2}{m_2}$$

This means that:

$$\hat{N}_1 = \frac{10 * 5}{2}$$
$$\hat{N}_1 = 25$$

Exercises:

You are a plant ecologist working on a rare prairie plant Mead's milkweed (*Asclepias meadii*), which does not produce above ground parts every year and lives in dense vegetation where nonflowering stems are hard to observe. In these cases, a count of the number of plants observed in a particular year may greatly underestimate the true population size, just as a count of animals in traps does not adequately estimate the total number of animals in an area. You are still determined to estimate population size and you decide to use the Petersen Lincoln estimator as a first approach. You decide to estimate population size by conducting two surveys during peak flowering season in early June. You visit the site first in 2020 and then in 2021. In 2020, you visit the study site and search systematically the entire prairie and mark those plants and you can identify 15 Mead's milkweed plants. In 2021 you return and conduct a more focused survey of known locations of stems in earlier years and

look for tagged plants. At that time, you find 13 Mead's milkweed stems of which 4 where already marked in 2020.

For questions 1-3:* If you are following the 🐡 or 🐻 steps use this shiny app to answer the questions.
* If you are following the 🦁& 🦡 write the code to answer questions above.

1. What is the value of n_1, n_2 and m_2 for the population of Mead's milkweed?
2. Estimate the proportion of Mead's milkweed plants that are marked (n_2/m_2).
3. Estimate the population size of Mead's milkweed plants at your study site ($\widehat{N_1}$).
For questions 4&5 you don´t need any code so all 🐡, 🐻, 🦁& 🦡 think about what we have explained before and justify your answers.

4. The Petersen Lincoln estimator assumes that:

- The population is closed. No individuals are leaving or arriving to the forest.
- All individuals have the same probability of being catch.
- Individuals don´t lose their marks.
- Random mixing.
- Samples are independent.

Natural systems not always fulfil all these conditions. Discuss briefly (1-2 lines each) which ones of the conditions above could be violated in the Mead's milkweed study system.
5. If your individuals were losing marks, would you underestimate or overestimate population size? Justify your answer in 1-3 lines.

Also, if sample size is small, the basic L-P estimator can be biased. A modified version that aims to reduce such biases was originally developed by Chapman (1951) and is commonly called the modified Petersen estimate or the Chapman estimator:

$$\hat{N}_1 = \frac{(n_1+1)(n_2+1)}{m_2+1} - 1$$

$$Var(\hat{N}_1) = \frac{(n_1+1)(n_2+1)(n_1-m_2)(n_2-m_2)}{(m_2+1)^2(m_2+2)} - 1$$

$$StDev(\hat{N}_1) = \sqrt{Var(\hat{N}_1)}$$

$$\hat{N}_1 \pm 1.96 StDev(\hat{N}_1)$$

2 Estimating vital rates

2.1 Estimating breeding success

Predictable food subsidies, together with the protection of suitable breeding areas in recent years, has led to an increase in large opportunistic gull populations such as Yellow-legged gulls (*Larus michahellis*). Their growth has caused social annoyance as well as concerns for public health and has triggered population control measures worldwide. For instance, Yellow-legged Gulls, (*Larus michahellis*) in the Mediterranean basin have been subject to numerous culling programs aimed at controlling their numbers (the symptom) by targeting either a reduction in adult survival rates or breeding performance. However, recent European Union (EU) environmental policies are beginning to target the causes rather than their symptoms. You are a PhD student looking at the effect of the Landfill Waste Council Directive (LWCD; European Commission 2008, EU 2009) on breeding success of Yellow legged gulls breeding in Dragonera Island (Mallorca). The changes in this directive aim to put an end to open-air landfill sites therefore they are expected to cause a progressive reduction of predictable subsidies and reduce the food availability for the gulls impacting their breeding success. To do that you go to the breeding colony in 2006 before the landfill was closed and once in 2012 once the landfill was closed. When you go there you collect data on the number of clutches of zero, one, two and three eggs. Since the nests are usually in uneven and covered with vegetation terrain you use the Petersen Lincoln method, and you conduct two transects in each visit. In the first transect you count the total number of nests of each size seen (n_1) and mark them by writing a x in with a pencil in one of the eggs. Then in the second transect you count the total number of nests seen (n_2) and the number of those nests that were already tagged (m_2).

You came back to the field with the following data for 2006, before the landfill was closed.

Clutch size	Total seen at t1 (n_1)	Total recaptured at t2 (m_2)	Total seen at t2 (n_2)	$\widehat{N}_{1,cs}$
0 eggs	18	3	4	$N_{1,0}$
1 egg	34	3	6	$N_{1,1}$
2 eggs	24	4	4	$N_{1,2}$
3 eggs	28	3	4	$N_{1,3}$

And the following data for 2012, before the landfill was closed.

Clutch size	Total seen at t1 (n_1)	Total recaptured at t2 (m_2)	Total seen at t2 (n_2)	$\widehat{N}_{1,cs}$
0 eggs	9	3	4	$N_{1,0}$
1 egg	11	3	4	$N_{1,1}$
2 eggs	17	3	5	$N_{1,2}$
3 eggs	21	2	5	$N_{1,3}$

- If you are following the 🐲 or 🐼 steps use this shiny app to answer question 6, then use a calculator to answer question 7.
- If you are following the 🐻 & 🐶 write the code to answer questions above.

Using the Petersen-Lincoln or the Chapman method discuss whether you think the landfill closure influenced breeding success. To do that you will need to:

6. Estimate total number of clutches of each size (0 eggs, 1 egg,2 eggs and 3 eggs). If you use the Petersen-Lincoln Method you should do:

$$\hat{N}_{1,cs} = \frac{n_{1,cs} * n_2}{m_2}$$

For each clutch size (CS)

7. Estimate fecundity by calculating:

Independently of what method you have used to estimate $\widehat{N}_{1,cs}$ then you should calculate fecundity before and after using the following expressions.

$$f = \frac{\sum_{n=0}^{3} N_{1,cs} * CS}{\sum_{n=0}^{3} N_1}$$

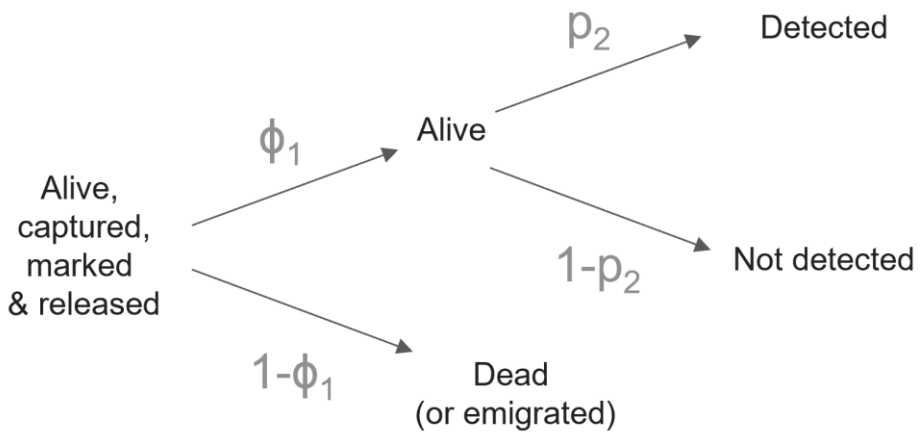
$$f = \frac{N_{1,0} * 0 + N_{1,1} * 1 + N_{1,2} * 2 + N_{1,3} * 3}{N_{1,0} + N_{1,1} + N_{1,2} + N_{1,3}}$$

3.2.2 Estimating survival

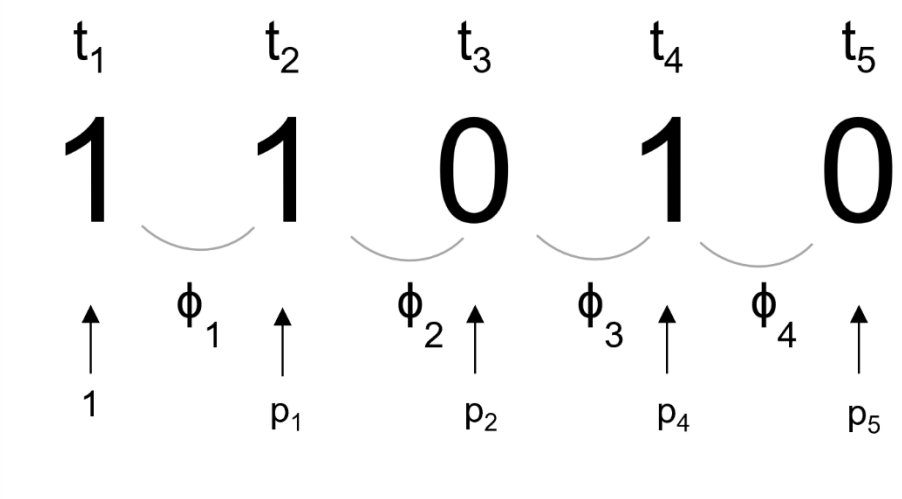
White-throated Dipper

Cormack-Jolly-Seber model

The Cormack-Jolly-Seber model allows to estimate apparent survival (ϕ) and recapture probability (p) in a given population.



- ϕ_t is the probability that a marked individual in the study population at sampling period t survives until period $t+1$ and remains in the population (does not permanently emigrate). Thus, apparent survival in year i relates to the interval between resighting (or capture) periods t and $t+1$. $1-\phi_t$ is the probability that individuals died and left the population (emigration). That is why ϕ_t is called apparent survival and it will always be lower than true survival.
- p_t is the probability that a marked individual in the study population at sampling period t is captured or observed during period t.
- χ_t is the probability that an individual alive and in the study population at sampling period t is not caught or observed again at any sampling period after period t. For a study with T sampling periods, X_1, and values for periods with $t<T$ can be obtained recursively as: $\chi_t=(1-\phi_1)+\phi_1(1-p_{t+1})\chi_{t+1}$



Assumptions:

- All marked individuals in the population at given time(t) have the same probability p of being captured or resighted.
- All marked individuals in the population at given time (t) have the same probability ϕ of survival until sampling period t+1.
- Marks are can´t be lost nor overlooked and are recorded correctly.
- Sampling periods are instantaneous.
- Individuals recaptured are released immediately.
- All emigration from the study area is permanent.
- The fate of each individual is independent of the fate of any other individual.

Answer the following questions:

8. 🐢, 🐼, 🐻& 🐊. Below we provide a diagram of fates for a 3-occasions recapture experiment.

ϕ_t: probability of survival between period t and t+1.

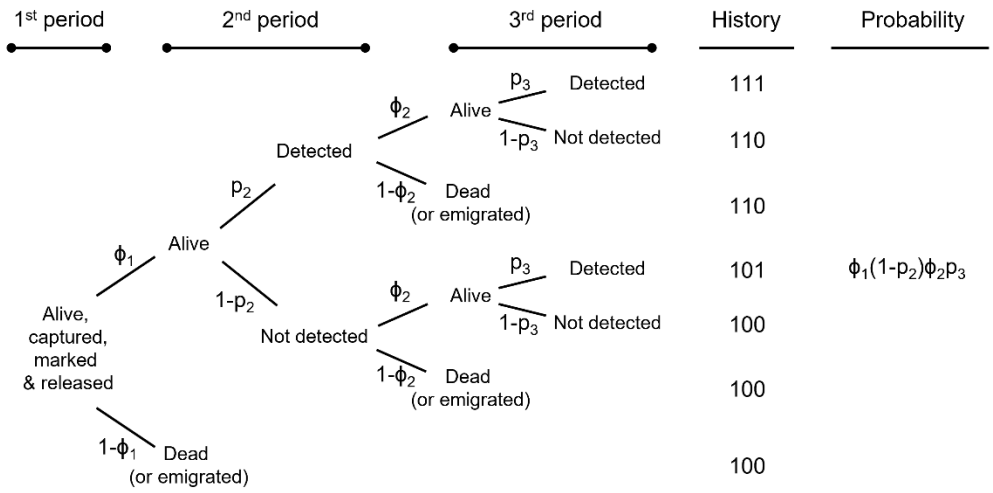
p_t: probability of detection at period t.

We provide you with the probability of the capture recapture history "101" as an example which is $\phi_1(1-p_2)\phi_2 p_3$. Look through the diagram of fates and formulate the probability for each one of the other 6 capture recapture histories.

Diagram of fates for a 3-occasions recapture experiment

ϕ_t : probability of survival between period *t* and *t+1*

p_t : probability of detection at period *t*



9. 🐢, 🐼 The code below estimates survival an recapture probability for White throated dippers. Run the code and get the survival and recapture probability. Then use the code to

estimate the probability of each one of the capture recapture histories above. Notice: your estimate of survival and recapture from the model will be constant so you will have to use the same value of ϕ and p for all time steps.

🐻& 🦡Run the code below and extract the survival and recapture estimates. Then, try to modify the code to produce time dependent survival and recapture estimates following https://github.com/papavientos/dipper-nimble/blob/main/dipper-NIMBLE.pdf. Then calculate the probability of each one of the capture recapture histories.

2.2.1 Bayesian capture-recapture inference with hidden Markov models

Requirements 1. Install R and RStudio.

https://rstudio-education.github.io/hopr/starting.html

2. Install Nimble following these guidelines: https://r-nimble.org/download

3.Then run the following code in R._ If that runs without error, you're all set. If not, please get in touch with us.

library(nimble) code <- nimbleCode({ y ~ dnorm(0,1) }) model <- nimbleModel(code) cModel <- compileNimble(model)

4. Install the following R packages: tidyverse, MCMCvis, here and nimbleEcology. You can install them all at once by running the following code in R:

install.packages(c("tidyverse", "MCMCvis", "here", "nimbleEcology"))

```
## Libraries----------
library(praise)
library(nimble)
library(tidyverse)
library(MCMCvis)
## Don't forget to run this line of code!!! MOST important line
# of the code.
praise()
## [1] "You are gnarly!"
## Loading dataset----------
dipper <- read_csv("dipper.csv") # data
sex <- ifelse(dipper$sex=="M",0 , 1) # vector sex

# Format data
y <- dipper %>%
  select(year_1981:year_1987) %>%
  as.matrix()
head(y)
##      year_1981 year_1982 year_1983 year_1984 year_1985 year_1986 year_1987
## [1,]         1         1         1         1         1         1         0
## [2,]         1         1         1         1         1         0         0
## [3,]         1         1         1         1         0         0         0
## [4,]         1         1         1         1         0         0         0
## [5,]         1         1         0         1         1         1         0
## [6,]         1         1         0         0         0         0         0
```

```
## CJS's models----------
## PHI(.) --------
# Survival and p are constant.

##### phi(.)p(.)----------
#define the model
hmm.phip <- nimbleCode({
  phi ~ dunif(0, 1) # prior survival
  p ~ dunif(0, 1) # prior detection
  gamma[1,1] <- phi      # Pr(alive t -> alive t+1)
  gamma[1,2] <- 1 - phi  # Pr(alive t -> dead t+1)
  gamma[2,1] <- 0        # Pr(dead t -> alive t+1)
  gamma[2,2] <- 1        # Pr(dead t -> dead t+1)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
  omega[1,2] <- p        # Pr(alive t -> detected t)
  omega[2,1] <- 1        # Pr(dead t -> non-detected t)
  omega[2,2] <- 0        # Pr(dead t -> detected t)

#Calculate the likelihood
for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})

#' Get the occasion of first capture for all individuals.
first <- apply(y, 1, function(x) min(which(x !=0)))


#' A list with constants.
my.constants <- list(N = nrow(y),
                     T = ncol(y),
                     first = first)

#' Now the data in a list. Note that we add 1 to the data to have 1 for non-detections and 2 for detections. You may use the coding you prefer of course, you will just need to adjust the $\Omega$ and $\Gamma$ matrices in the model above.
my.data <- list(y = y + 1)


#' Specify initial values. For the latent states, we go for the easy way, and say that all individuals are alive through the study period.
zinits <- y + 1 # non-detection -> alive
zinits[zinits == 2] <- 1 # dead -> alive
initial.values <- function() list(phi = runif(1,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)

#' Some information that we now pass as initial value info (observations of alive) are actually known states, and could also be passed as data – in which case the initial values have to be 0.
#' Specify the parameters we wish to monitor.
parameters.to.save <- c("phi", "p")
```

```
#' MCMC details.
n.iter <- 50000
n.burnin <- 10000
n.chains <- 2

#' At last, let's run nimble.
mcmc.phip <- nimbleMCMC(code = hmm.phip,
                    constants = my.constants,
                    data = my.data,
                    inits = initial.values,
                    monitors = parameters.to.save,
                    niter = n.iter,
                    nburnin = n.burnin,
                    nchains = n.chains)
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

Run the following code to see the results. MCMCsummary(mcmc.phip, round = 2) MCMCtrace(mcmc.phip, pdf = F)

```
#' Examine the results.
#MCMCsummary(mcmc.phip, round = 2)
#MCMCtrace(mcmc.phip, pdf = F)
```

Capture—mark—recapture models https://oliviergimenez.github.io/pubs/Grosbois%26Gimenez2010-bookAnders.pdf

https://oliviergimenez.github.io/hmm-cr-nimble-isec2022-workshop/

Population Modelling

What is a model?

A model is a simplified representation of reality.

What is population modelling?

Modelling is the construction and analysis of models that represent ecological processes. Models can be analytic or simulation-based. They are used to understand complex ecological processes and to predict how real systems might react to change.

Analytic models

These are mathematical models that have a closed form solution. This means that the solution can be expressed as a mathematical analytic function such as the Lotka Volterra model.

$$\frac{dx}{dt} = x(\alpha - \beta y),$$

$$\frac{dy}{dt} = -y(\gamma - \delta x).$$

x is the number of prey.
y is the number of some predator.
dy/dt and dx/dt represent the instantaneous growth rates of the two populations.
t represents time.
α, β, γ and δ are parameters describing the interaction between the two species.

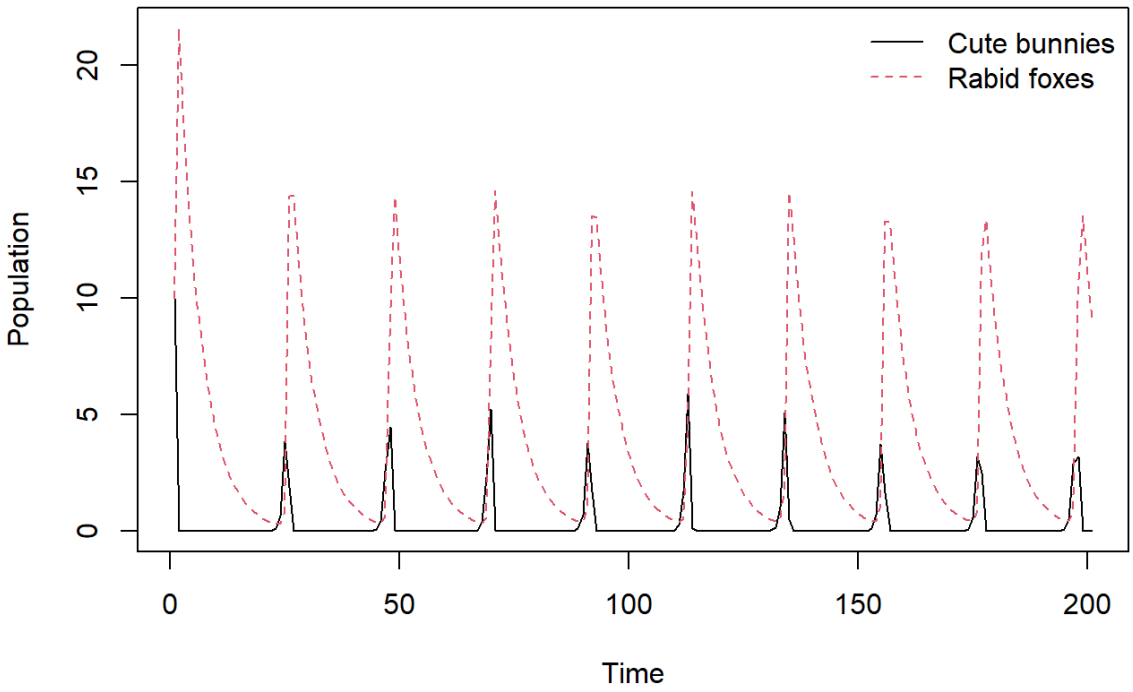
The code below allows you to replicate the dynamics of a population of prey and a population of predators. All 🐇, 🐼, 🐻 and 🦊 should do the following:
1 Run the following code that provides the solution of the Lotka Volterra equation for a given parametrization.

2 Change the running time, write down what you change it to and describe in 4 lines what happens to the prey predator dynamics.

3 Change α, β, γ and δ (one at a time), write down what you change it to each time and describe in 4 lines what happens to the prey predator dynamics.
4 Change the initial number of prey and predators x and y (one at a time) write down what you change it to each time and describe in 4 lines what happens to the prey predator dynamics.

Code:

```
library(deSolve)

LotVmod <- function (Time, State, Pars) {
  with(as.list(c(State, Pars)), {
    dx = x*(alpha - beta*y)
    dy = -y*(gamma - delta*x)
    return(list(c(dx, dy)))
  })
}
Pars <- c(alpha = 2, beta = .5, gamma = .2, delta = .6)
State <- c(x = 10, y = 10)
Time <- seq(0, 200, by = 1)
out <- as.data.frame(ode(func = LotVmod, y = State, parms = Pars, times = Time))
matplot(out[,-1], type = "l", xlab = "Time", ylab = "Population")
legend("topright", c("Cute bunnies", "Rabid foxes"), lty = c(1,2), col = c(1,2), box.lwd = 0,bty = "n")
```



Simulation-based models

These are a digital prototype of a system that can be used to test how it would react to different conditions.

Whenever natural scientists try to understand what they observe in nature, they make and interrogate models. Models are used to describe systems as well as to make new predictions and to generate ideas.

An example of simulation-based models are Individual based models (IBM's, also called 'agent-based models'). These models allow to simulate systems of discrete individuals in *silico* (i.e., in a computer simulation, developed by writing and running code). Here, we will represent individuals as discrete entities within the code, and to simulate changes in individual composition and individual properties by modelling biological processes as algorithms within the code.

IBMs are widespread and well-established tools for all types of questions involving biological modelling. Two properties of IBMs stand out to make them especially advantageous.

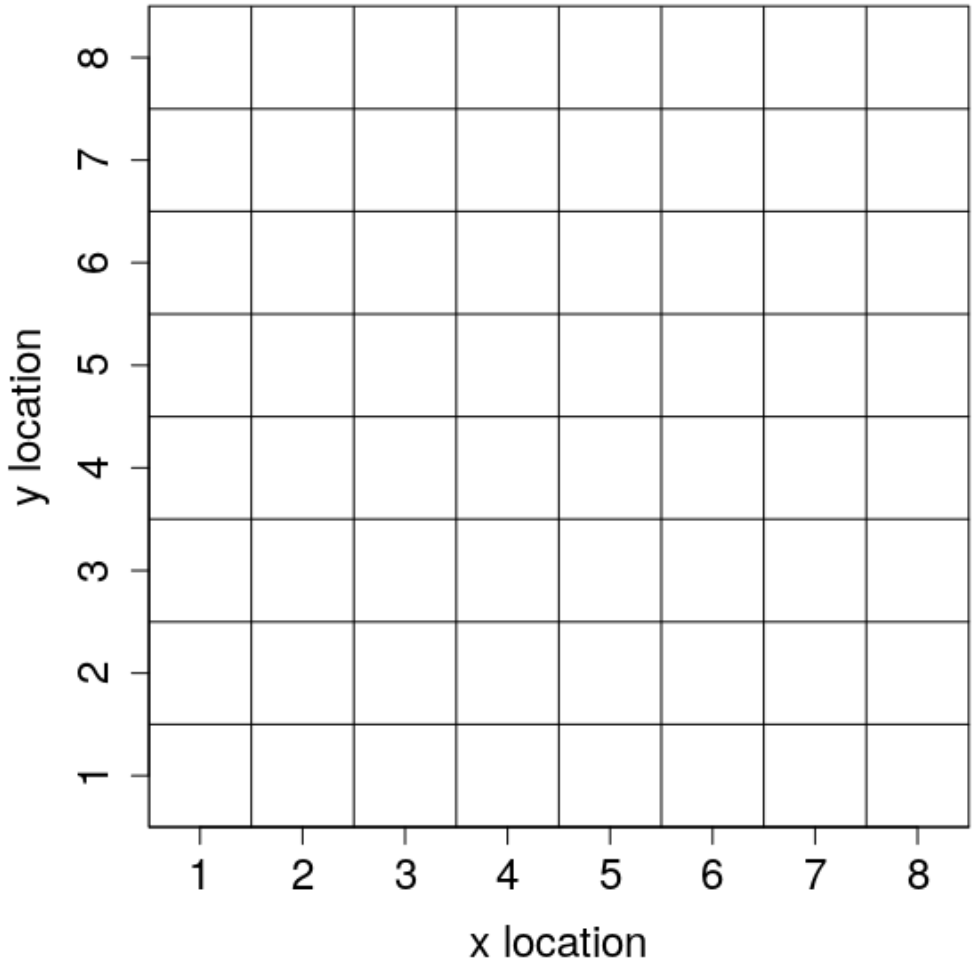
- The allow to implement among-individual variation – a highly important property for understanding biological populations – is easy to model with an IBM.
- And, because individuals are discrete, any biological processes being modelled are going to naturally include the kind of *stochasticity* * that is inherent to all biological populations.

Note**: *Stochasticity* refers to unpredictable events that can affect population and community dynamics are called stochastic processes.
Overall, IBMs are effective for addressing many (though, importantly, not all) questions relating to biological theory and predictive models.

Getting started with IBMs in R

In an IBM, discrete individuals are represented by some sort of data structure in code.

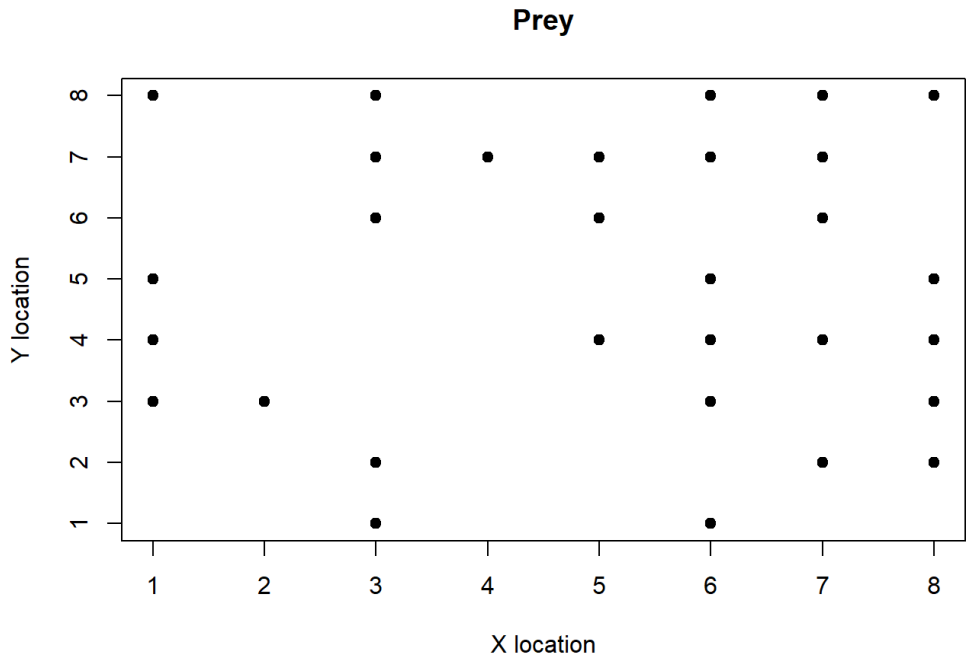
The code below creates a two-dimensional data array (inds) to model five individuals that have three characteristics each. We are going to create a population of prey and a population of predators that inhabit a landscape (lattice) of 8x8 squares.



Each individual is going to be given a body mass drawn from a normal distribution with mean 23 and standard deviation 3 that is stored in column 1. They are also given a starting location (x,y) that corresponds with columns 2 and 3. Then they are given a value of reproduction (number of offspring, column 4) and a column that records whether they are dead or not (column5).
Let's start by running the code below:

```
# ============================================================================
# Simulate predator-prey dynamics
```

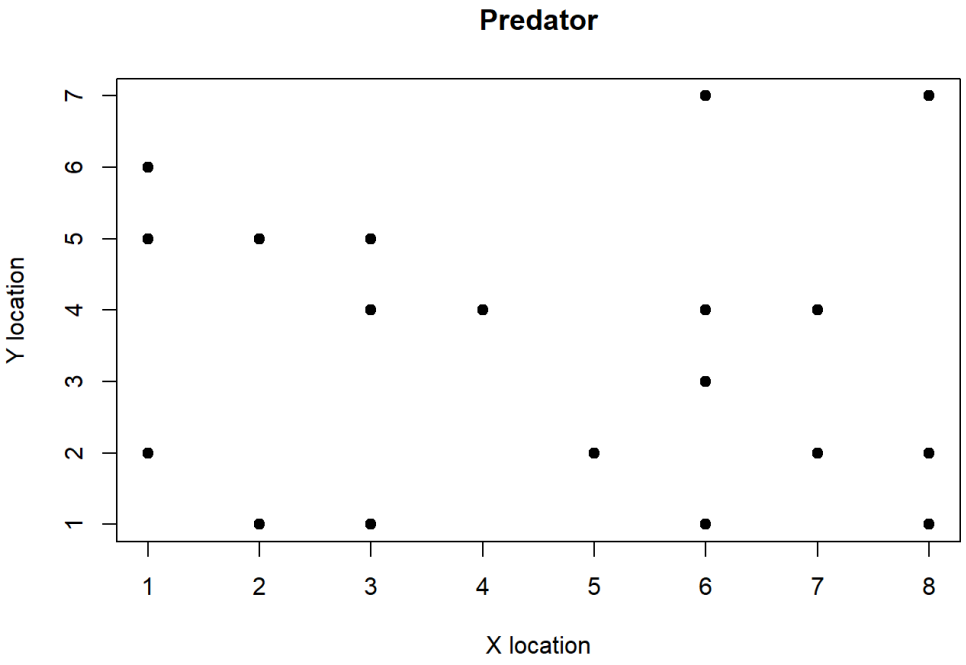
```r
# ========================================================================
# ----- Initialise individuals (prey)
inds          <- array(data = 0, dim = c(40, 5))
inds[,1]      <- rnorm(n = dim(inds)[1], mean = 23, sd = 3)
inds[,2]      <- sample(x = 1:8, size = dim(inds)[1], replace = TRUE)
inds[,3]      <- sample(x = 1:8, size = dim(inds)[1], replace = TRUE)
dim(inds)
## [1] 40  5
colnames(inds)<-c("body_mass", "x_loc", "y_loc", "repr", "death");
print(inds)
##       body_mass x_loc y_loc repr death
##  [1,]  24.83304     5     6    0     0
##  [2,]  22.54818     1     4    0     0
##  [3,]  26.24191     3     2    0     0
##  [4,]  23.52819     7     4    0     0
##  [5,]  23.04581     4     7    0     0
##  [6,]  23.43281     3     6    0     0
##  [7,]  22.13374     6     8    0     0
##  [8,]  23.47649     1     5    0     0
##  [9,]  24.42336     5     7    0     0
## [10,]  19.41873     8     2    0     0
## [11,]  19.91524     7     8    0     0
## [12,]  23.09564     6     7    0     0
## [13,]  22.78404     7     7    0     0
## [14,]  26.57972     2     3    0     0
## [15,]  19.95957     8     5    0     0
## [16,]  24.43444     4     7    0     0
## [17,]  23.27239     6     4    0     0
## [18,]  19.98176     6     7    0     0
## [19,]  26.24323     1     3    0     0
## [20,]  23.96657     3     1    0     0
## [21,]  22.71141     8     4    0     0
## [22,]  20.88945     6     1    0     0
## [23,]  21.68683     1     4    0     0
## [24,]  21.45100     5     4    0     0
## [25,]  28.74510     3     7    0     0
## [26,]  20.42008     7     8    0     0
## [27,]  20.11818     3     8    0     0
## [28,]  22.10621     7     7    0     0
## [29,]  16.80521     4     7    0     0
## [30,]  23.15678     8     8    0     0
## [31,]  19.89976     5     4    0     0
## [32,]  21.11161     3     2    0     0
## [33,]  31.15747     7     6    0     0
## [34,]  29.82129     6     3    0     0
## [35,]  25.76010     1     8    0     0
## [36,]  19.33704     6     5    0     0
## [37,]  17.27971     5     4    0     0
## [38,]  17.52547     3     7    0     0
## [39,]  15.92083     7     2    0     0
## [40,]  23.58212     8     3    0     0
plot(inds[,2],inds[,3], pch = 19, xlab="X location", ylab="Y location", main="Prey")
```



The plot above shows the spatial distribution of the individual prey in the first-time step of the simulation. The next bit of code does the same for the predators. Let's run it too.

```
# ----- Initialise individuals (predator)
pred        <- array(data = 0, dim = c(20, 5))
pred[,1]     <- rnorm(n = dim(pred)[1], mean = 40, sd = 3)
pred[,2]     <- sample(x = 1:8, size = dim(pred)[1], replace = TRUE)
pred[,3]     <- sample(x = 1:8, size = dim(pred)[1], replace = TRUE)
names(pred)<-c("body_mass", "x_loc", "y_loc", "repr", "death");
print(inds)
##      body_mass x_loc y_loc repr death
## [1,]  24.83304    5    6    0     0
## [2,]  22.54818    1    4    0     0
## [3,]  26.24191    3    2    0     0
## [4,]  23.52819    7    4    0     0
## [5,]  23.04581    4    7    0     0
## [6,]  23.43281    3    6    0     0
## [7,]  22.13374    6    8    0     0
## [8,]  23.47649    1    5    0     0
## [9,]  24.42336    5    7    0     0
## [10,]  19.41873    8    2    0     0
## [11,]  19.91524    7    8    0     0
## [12,]  23.09564    6    7    0     0
## [13,]  22.78404    7    7    0     0
## [14,]  26.57972    2    3    0     0
## [15,]  19.95957    8    5    0     0
## [16,]  24.43444    4    7    0     0
## [17,]  23.27239    6    4    0     0
## [18,]  19.98176    6    7    0     0
## [19,]  26.24323    1    3    0     0
## [20,]  23.96657    3    1    0     0
## [21,]  22.71141    8    4    0     0
## [22,]  20.88945    6    1    0     0
```

```
## [23,]  21.68683   1   4   0   0
## [24,]  21.45100   5   4   0   0
## [25,]  28.74510   3   7   0   0
## [26,]  20.42008   7   8   0   0
## [27,]  20.11818   3   8   0   0
## [28,]  22.10621   7   7   0   0
## [29,]  16.80521   4   7   0   0
## [30,]  23.15678   8   8   0   0
## [31,]  19.89976   5   4   0   0
## [32,]  21.11161   3   2   0   0
## [33,]  31.15747   7   6   0   0
## [34,]  29.82129   6   3   0   0
## [35,]  25.76010   1   8   0   0
## [36,]  19.33704   6   5   0   0
## [37,]  17.27971   5   4   0   0
## [38,]  17.52547   3   7   0   0
## [39,]  15.92083   7   2   0   0
## [40,]  23.58212   8   3   0   0
plot(pred[,2],pred[,3], pch = 19, xlab="X location", ylab="Y location", main="Predator")
```



The plot above shows the spatial distribution of the individual prey in the first-time step of the simulation.

Now, we are going to write three functions that will simulate what happens to the individuals in a natural system.

- Movement
- Reproduction
- Survival
- Predation

We are going to define a function that allows the individuals to move along the landscape that we have created. Notice that we don´t want our individuals to leave the study area that

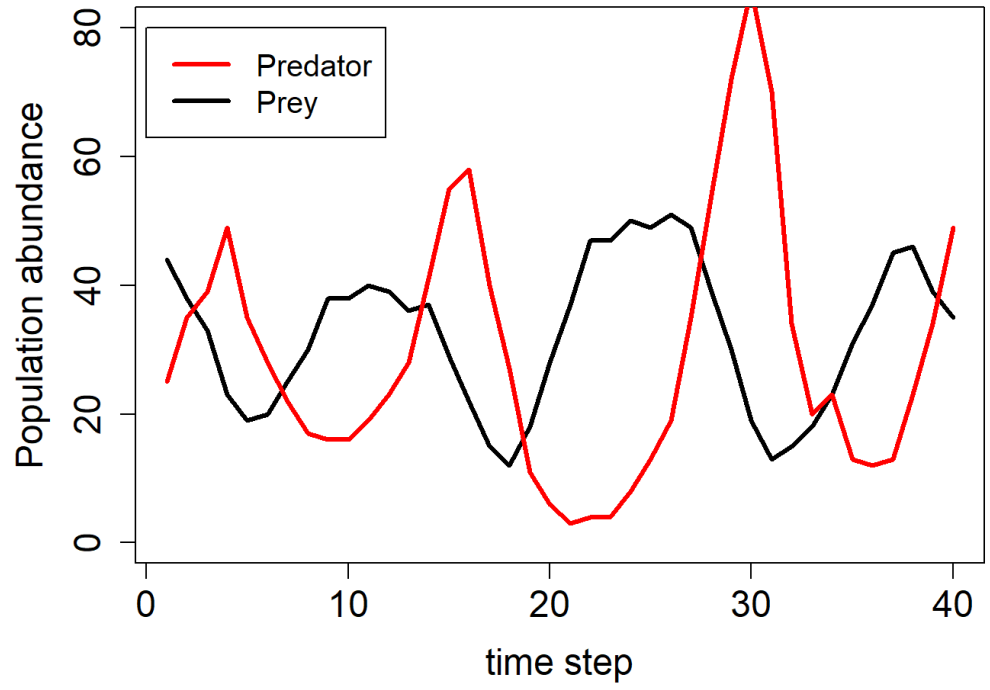
we have created. To avoid that happening we must implement what is called a reflecting boundary. This means that any individuals that the movement function takes outside the study area will be relocated again at the edge of the landscape. Run the next bits of code until you get a graph with the prey predator dynamics.

```
# ============================================================================
# Movement function
# ============================================================================
movement <- function(inds, xloc = 2, yloc = 3, xmax = 8, ymax = 8){
  total_inds  <- dim(inds)[1] # Get the number of individuals in inds
  move_dists  <- c(-1, 0, 1)  # Define the possible distances to move
  x_move      <- sample(x = move_dists, size = total_inds, replace = TRUE) #this finds the new x location for
the individual
  y_move      <- sample(x = move_dists, size = total_inds, replace = TRUE)#this finds the new x location for
the individual
  inds[, xloc] <- inds[, xloc] + x_move #this stores the new x location for the individual
  inds[, yloc] <- inds[, yloc] + y_move #this stores the new y location for the individual

  # =========  The reflecting boundary is added below

  for(i in 1:total_inds){             # For each individual i in the array
    if(inds[i, xloc] > xmax){         # If it moved passed the maximum xloc
      inds[i, xloc] <- xmax - 1     # Then move it back toward the centre
    }
    if(inds[i, xloc] < 1){           # If it moved below 1 on xloc
      inds[i, xloc] <- 2           # Move it toward the centre (2)
    }
    if(inds[i, yloc] > ymax){         # If it moved passed the maximum yloc
      inds[i, yloc] <- ymax - 1     # Then move it back toward the centre
    }
    if(inds[i, yloc] < 1){           # If it moved below 1 on yloc
      inds[i, yloc] <- 2           # Then move it toward the centre (2)
    }
  }
  # =========  Now all individuals should stay on the landscape
  return(inds)
}
# ============================================================================
# reproduction function
# ============================================================================
reproduction <- function(inds, lambda = 0.5, repr_col = 4){
  total_inds      <- dim(inds)[1] # Get the number of individuals in inds
  ind_cols        <- dim(inds)[2] # Total inds columns
  inds[, repr_col] <- rpois(n = total_inds, lambda = lambda)
  total_off       <- sum(inds[, repr_col])
  # ---- We now have the total number of new offspring now add to inds
  new_inds     <- array(data = 0, dim = c(total_off, ind_cols))
  new_inds[,1] <- rnorm(n = dim(new_inds)[1], mean = 23, sd = 3)
  new_inds[,2] <- sample(x = 1:8, size = dim(new_inds)[1], replace = TRUE)
  new_inds[,3] <- sample(x = 1:8, size = dim(new_inds)[1], replace = TRUE)
  # ---- Our new offspring can now be attached in the inds array
  inds <- rbind(inds, new_inds)
  return(inds)
}
# ============================================================================
# survival function
```

```r
# ============================================================================
survival <- function(inds, xlen = 8, ylen = 8, dcol = 5, xcol = 2, ycol = 3){
  for(xdim in 1:xlen){     # For each row `xdim` of the landscape...
    for(ydim in 1:ylen){  # For each col `ydim` of the landscape...
      # Get the total number of individuals on the landscape cell
      on_cell <- sum( inds[, xcol] == xdim & inds[, ycol]  == ydim)
      # Only do something if on_cell is more than one
      if(on_cell > 1){
        # Get all of the occupants on the cell
        occupants <- which(inds[, xcol] == xdim & inds[, ycol] == ydim)
        # Sample all but one random occupants to die
        rand_occ  <- sample(x = occupants, size = on_cell - 1)
        # Then add their death to the last column of inds
        inds[rand_occ, dcol] <- 1
      }
    }
  }
  return(inds)
}
# ============================================================================
# Predation function
# ============================================================================
predation <- function(pred, inds, xcol = 2, ycol = 3, rcol = 4, dcol = 5){
  predators  <- dim(pred)[1] # Predator number
  pred[, dcol] <- 1          # Assume dead until proven otherwise
  pred[, rcol] <- 0          # Assume not reproducing until proven otherwise
  for(p in 1:predators){      # For each predator (p) in the array
    xloc  <- pred[p, xcol] # Get the x and y locations
    yloc  <- pred[p, ycol]
    N_prey <- sum( inds[, xcol] == xloc & inds[, ycol] == yloc)
    # ----- Let's take care of the predator first below
    if(N_prey > 0){
      pred[p, dcol] <- 0  # The predator lives
    }
    if(N_prey > 1){
      pred[p, rcol] <- 1  # The predator reproduces
    }
    # ----- Now let's take care of the prey
    if(N_prey > 0){ # If there are some prey, find them
      prey <- which( inds[, xcol] == xloc & inds[, ycol] == yloc)
      if(N_prey > 2){ # But if there are more than 2, just eat 2
        prey <- sample(x = prey, size = 2, replace = FALSE)
      }
      inds[prey, dcol] <- 1 # Record the prey as dead
    }
  } # We now know which inds died, and which prey died & reproduced
  # ---- Below removes predators that have died
  pred       <- pred[pred[,dcol] == 0,] # Only survivors now left
  # ----- Below adds new predators based on the reproduction above
  pred_off    <- sum(pred[, rcol])
  new_pred    <- array(data = 0, dim = c(pred_off, dim(pred)[2]))
  new_pred[,1] <- rnorm(n = dim(new_pred)[1], mean = 23, sd = 3)
  new_pred[,2] <- sample(x = 1:8, size = dim(new_pred)[1], replace = TRUE)
  new_pred[,3] <- sample(x = 1:8, size = dim(new_pred)[1], replace = TRUE)
  pred       <- rbind(pred, new_pred)
  # ----- Now let's remove the prey that were eaten
  inds       <- inds[inds[,dcol] == 0,] # Only living prey left
```

```r
  # Now need to return *both* the predator and prey arrays
  pred_prey <- list(pred = pred, inds = inds)
  return(pred_prey)
}
# ---- Start the simulation as before
ts        <- 0
time_steps <- 40
inds_hist  <- NULL
pred_hist  <- NULL
while(ts < time_steps){
  pred          <- movement(pred)
  inds          <- movement(inds) # Note: I increased prey birth rate
  inds          <- reproduction(inds, lambda = 1.5)
  pred_prey_res   <- predation(pred = pred, inds = inds)
  pred          <- pred_prey_res$pred
  inds          <- pred_prey_res$inds
  inds          <- survival(inds)
  inds          <- inds[inds[, 5] == 0,] # Retain living
  ts            <- ts + 1
  inds_hist[[ts]] <- inds
  pred_hist[[ts]] <- pred
}
# ============================================================================
# Print the results
# ============================================================================
ind_abund <- array(data = NA, dim = c(40, 3))
for(i in 1:40){
  ind_abund[i, 1] <- i                  # Save the time step
  ind_abund[i, 2] <- dim(inds_hist[[i]])[1] # rows in inds_hist[[i]]
  ind_abund[i, 3] <- dim(pred_hist[[i]])[1] # rows in pred_hist[[i]]
}
colnames(ind_abund) <- c("time_step", "abundance", "predators")
print(ind_abund)
##      time_step abundance predators
##  [1,]         1        44        25
##  [2,]         2        38        35
##  [3,]         3        33        39
##  [4,]         4        23        49
##  [5,]         5        19        35
##  [6,]         6        20        28
##  [7,]         7        25        22
##  [8,]         8        30        17
##  [9,]         9        38        16
## [10,]        10        38        16
## [11,]        11        40        19
## [12,]        12        39        23
## [13,]        13        36        28
## [14,]        14        37        41
## [15,]        15        29        55
## [16,]        16        22        58
## [17,]        17        15        40
## [18,]        18        12        27
## [19,]        19        18        11
## [20,]        20        28         6
## [21,]        21        37         3
## [22,]        22        47         4
## [23,]        23        47         4
```

```
## [24,]    24    50     8
## [25,]    25    49    13
## [26,]    26    51    19
## [27,]    27    49    35
## [28,]    28    39    54
## [29,]    29    30    72
## [30,]    30    19    86
## [31,]    31    13    70
## [32,]    32    15    34
## [33,]    33    18    20
## [34,]    34    23    23
## [35,]    35    31    13
## [36,]    36    37    12
## [37,]    37    45    13
## [38,]    38    46    23
## [39,]    39    39    34
## [40,]    40    35    49
# ============================================================================
# Plot the results
# ============================================================================
par(mar = c(5, 5, 1, 1))
{plot(x = ind_abund[,2], type = "l", lwd = 3, ylim = c(0, 80),
    xlab = "time step", ylab = "Population abundance", cex.axis = 1.5,
    cex.lab = 1.5)
points(x = ind_abund[,3], type = "l", lwd = 3, col = "red")
legend(x = 0, y = 80, legend = c("Predator", "Prey"), col = c("red", "black"),
      cex = 1.25, lty = c("solid", "solid"), lwd = c(3, 3))
}
```



🐠, 🐻, 🦁 and 🦾 Now let's play a bit with the model and make some changes to see what happens.

5 Change the set of distances that the individuals can move.

We are going to define a function that allows the individuals to reproduce and generate offspring. To simulate birth in our population of individuals, we need some rule to determine when an individual reproduces. As a simplifying assumption, let as assume that all individuals in our model are female, and that the number of birth events for an individual is sampled from a Poisson distribution with 0.5 expected offspring per time step.

In R, we can sample some number of values from a Poisson distribution with a given rate parameter (λ, defining both the mean and variance of the Poisson distribution) using the rpois function. If we wanted to sample for five individuals with $\lambda = 0.5$, we could therefore use the code below.

```
rpois(n = 5, lambda = 0.5);
```

6 Try running the line of code above in your R console. Then try changing the number of individuals and then change the lambda value.

7 Now we are going to silence some of functions above to see what happens with the population.How do I silence a function? You can do that just by adding # in front of the line of code that you want to mute. For instance, if you want to silence movement you should do the following in your code:

```
# pred<- movement(pred) # inds<- movement(inds) # Note: I increased prey birth rate
```

Set all parameters to the original values of the code and:

8 Silence the movement function and run the whole code. Save the plot and describe what happens in 4 lines.

9 Silence the reproduction function and run the whole code. Save the plot and describe what happens in 4 lines.

10 Silence the predation function and run the whole code. Save the plot and describe what happens in 4 lines.

11 Silence the survival function and run the whole code. Save the plot and describe what happens in 4 lines.

12 Plot the spatial dynamics of the prey predator relationship.

13 Try to think of another biological process you could include in the list of functions, create a function, and modify the code to integrate it.