LAB Fishy Matrices

ER Deyle

Fall 2024; Marine Semester Block 3

1. Introduction

In this lab, we will investigate Leslie matrix models of age structured populations.

Some of the goals of this lab are to:

- Provide another exercise in using loops to generate model time-series of populations.
- Build intuition about matrix multipication and eigenvectors.

By the end of this lab, you should be able to create and use `data.frame` structures.

Fisheries Context

Very few species of fish reproduce every year with complete turn-over of the population. Even in the case of salmon which are semelparous spawners (adults generally only spawn a single time and die after), the population at any one time is composed of multiple age classes because the life cycle takes more than a year to play out. Frasier River sockeye salmon (the inspiration for part of this lab), for example, generally take 4 years to complete their life cycle.

Leslie matrices are perhaps the simplest age-structured model. Suppose your population is composed of n age-classes. If the fecundity of each age class (number of Year 0 offspring created per individual), F_i and the survival rate of each age class, S_i, are both constant, then next year's abundance of each age class is:

$$\begin{aligned} N_1(t+1) &= \sum_{i=1}^{n} F_i N_i(t) \\ N_2(t+1) &= S_1 N_1(t) \\ &\vdots \\ N_i(t+1) &= S_{i-1} N_{i-1}(t) \\ &\vdots \end{aligned}$$

This is just a set of coupled linear equations in the n variables $\{N_1(t), N_2(t), ...N_n(t)\}$. As such we can rewrite it as a matrix equations.

$$\begin{bmatrix} N_1(t+1) \\ N_2(t+1) \\ N_3(t+1) \\ \vdots \\ N_n(t+1) \end{bmatrix} = \begin{bmatrix} F_1 & F_2 & F_3 & \cdots & F_n \\ S_1 & 0 & 0 & \cdots & 0 \\ 0 & S_2 & 0 & \cdots & 0 \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \cdots & S_{n-1} & 0 \end{bmatrix} \begin{bmatrix} N_1(t) \\ N_2(t) \\ N_3(t) \\ \vdots \\ N_n(t) \end{bmatrix}.$$

In the case of many species, the first few age-classes are not sexually mature. So commonly, $F_1 = 0$ and possibly more. In this lab, we'll start with a Leslie matrix representing a 4-year spawning salmon population. In that case, the only $F_i \neq 0$ is F_4.

$$\begin{bmatrix} N_1(t+1) \\ N_2(t+1) \\ N_3(t+1) \\ N_4(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & F_4 \\ S_1 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 \\ 0 & 0 & S_3 & 0 \end{bmatrix} \begin{bmatrix} N_1(t) \\ N_2(t) \\ N_3(t) \\ N_4(t) \end{bmatrix}.$$

Mathematical Approach

Up until now, we've represented the state of a population data as individual vectors. In this lab, we will instead be representing populations by multiple numbers. In an age-structured model, the population at any given time is represented by the abundances of indidividual age-classes. For an introduction to data-frames, we can run through the Data Carpentry ecology lesson:

https://datacarpentry.org/R-ecology-lesson/working-with-data.html

2. 4-year Salmon Runs

As mentioned above, let's begin with a Leslie matrix inspired by 4-year running salmon. Salmon, like many other fish, have very high mortality in their earliest life stages. This model assumes only 5 in 1000 fish survive their first year! Salmon also have very high fecundity.

$$M = \begin{bmatrix} 0 & 0 & 0 & 2000 \\ 0.005 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \end{bmatrix}$$

```
M_salmon_1 <- rbind(
  c(0,0,0,2000),
  c(0.005,0,0,0),
  c(0,0.2,0,0),
  c(0,0,0.5,0)
)
```

Practice matrix multipication

As described above, the convenience of a matrix model is that an iteration of the model (to advance the population "1 year" ahead) just involves multipication by the matrix. Recall in R that matrix multipication is denoted `%*%` not `*`.

TASK 1: Calculate the age distribution in Year 2 if the population in Year 1 is composed of a single age-class, for each of the four Year-1 populations below.

$$v_{ex1}(1) = \begin{bmatrix} 1000 \\ 0 \\ 0 \\ 0 \end{bmatrix}, v_{ex2}(1) = \begin{bmatrix} 0 \\ 1000 \\ 0 \\ 0 \end{bmatrix}, v_{ex3}(1) = \begin{bmatrix} 0 \\ 0 \\ 1000 \\ 0 \end{bmatrix}, v_{ex4}(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1000 \end{bmatrix}$$

TASK 2: Calculate the age distribution in Year 2 if the population in Year 1 is composed of equal age-classes.

$$v_{ex5}(1) = \begin{bmatrix} 1000 \\ 1000 \\ 1000 \\ 1000 \end{bmatrix}$$

> What do you notice comparing $v_{ex5}(2)$ to the results for the task before?

The above should demonstrate how a matrix is a *linear transformation*. That means that if you can write a vector v as the sum of two other vectors, u_1 and u_2, i.e. $v = u_1 + u_2$ that $\mathbf{M}v = \mathbf{M}u_1 + \mathbf{M}u_2$.

TASK 3: Calculate the age distribution in Year 4 for the above initial age distributions $v_{ex1}(1), v_{ex2}(1), v_{ex3}(1), v_{ex4}(1), v_{ex5}(1)$.

> Initially, please do this with sequential lines of code. e.g., for the first initial age distribution v_{ex1}, calculate $v_{ex1}(2)$ from $v_{ex1}(1)$, then $v_{ex1}(3)$ from $v_{ex1}(2)$, then $v_{ex1}(4)$ from $v_{ex1}(3)$. However, iterating a matrix model like this is also equivalent to $\mathbf{MMM}v_{ex1}(1)$. Ask R to calculate this in one line; you should get the same thing. Repeat this process for all five initial age distributions.

Simulation

Since we can advance the population a year, we can now use loops to simulate the population dynamics over multiple years.

We'll use a `data.frame` to store the age-structure population data. But there is a bit of issue going in between data formats for storing the age-structured population in a data.frame and using matrix multipication. Let's look at an example. We can store the same population data in both a vector and a data.frame:

```
v_salmon_ex6 <- c(1000,500,250,125)
df_salmon_ex6 <- data.frame(N_1=1000,N_2=500,N_3=250,N_4=125)
```

If you look in the "Environment" tab of Rstudio, you'll see that `v_salmon_ex6` is "num [1:4]" while `df_salmon_ex6` is "1 obs. of 4 variables". They contain the same numbers, but R treats them differently. Notice that `v_salmon_ex6` has a `length()` but no `dim()` while `df_salmon_ex6` has both.

```
dim(v_salmon_ex6)
```

```
## NULL
```

```
length(v_salmon_ex6)
```

```
## [1] 4
```

```
dim(df_salmon_ex6)
```

```
## [1] 1 4
```

```
length(df_salmon_ex6)
```

```
## [1] 4
```

This is because R is not actually treating the vector as a vector yet. Confusing, right? Why does this matter? Well, R will give a vector like `v_salmon_ex6` dimensions if it needs to for matrix multipication, but it will wait to decide if it's a row vector (i.e. dim(v) = c(1,4)) or a column vector (i.e. dim(v) = c(4,1)) based on which makes sense. We were using this before! What about the `data.frame` though?

```
M_salmon_1 %*% df_salmon_ex6
```

```
## Error in M_salmon_1 %*% df_salmon_ex6: requires numeric/complex matrix/vector arguments
```

R complains. The data.frame is not something it is willing to do matrix multipication on. That's because ultimately, a `data.frame` is a fancy kind of `list`.

```
typeof(df_salmon_ex6)
```

```
## [1] "list"
```

We need to convert the `data.frame` to a numeric matrix if we are going to use the matrix multipication operator `%*%`. R has a lot of functions for converting between data types, some of them are closely related and can be confusing. Type `?as.matrix`, `?as.vector`, and `?as.numeric`. If we use `as.matrix()`, we seem to get what we want.

```
as.matrix(df_salmon_ex6)
```

```
##       N_1 N_2 N_3 N_4
## [1,] 1000 500 250 125
```

However, note its dimensions:

```
dim(as.matrix(df_salmon_ex6))
```

```
## [1] 1 4
```

We have a 1 x 4 matrix, i.e. a row vector. What happens if we try to do the matrix multipication now?

```
M_salmon_1 %*% as.matrix(df_salmon_ex6)
```

```
## Error in M_salmon_1 %*% as.matrix(df_salmon_ex6): non-conformable arguments
```

Non-conformable arguments! That is because matrix multipication is only defined for an (n x p) matrix times a (p x m) matrix. We needed a column vector, not a row vector! Mathematically, that means we need the *matrix transpose* of the row vector. This is a very short function in R, just `t()`. Type `?t`.

```
M_salmon_1 %*% t(as.matrix(df_salmon_ex6))
```

```
##        [,1]
## [1,] 250000
## [2,]      5
## [3,]    100
## [4,]    125
```

This got us what we wanted. What happens if we use one of the other conversions, `as.numeric()`?

```
as.numeric(df_salmon_ex6)
```

```
## [1] 1000  500  250  125
```

```
dim(as.numeric(df_salmon_ex6))
```

```
## NULL
```

R will leave it like our original starting vectors- not set as either a row or column vector. (Try `as.vector()`, too, and see what happens. Is the result like `as.numeric()` or like `as.matrix()`?) Why does this all work out this particular way? Honestly, I couldn't tell you. But now that we know, we can write the code we want.

At any rate, you will want to (1) set your initial population up:

```
v_age_classes <- c(1000,500,250,125)
y_i <- 1
```

(2) create a one row data.frame to begin adding results to

```
df_salmon <- data.frame(Year = y_i,
                        N_1=v_age_classes[1],
                        N_2=v_age_classes[2],
                        N_3=v_age_classes[3],
                        N_4=v_age_classes[4])
```

and (3) write a `for` loop to iterate the population dynamics until the year is 100. Inside the `for` loop should look something like this:

```
  y_i <- y_i + 1
  v_age_classes <- M_salmon_1 %*% v_age_classes

  df_salmon_y_i <- data.frame(Year = y_i,
                              N_1=v_age_classes[1],
                              N_2=v_age_classes[2],
                              N_3=v_age_classes[3],
                              N_4=v_age_classes[4])
  df_salmon <- rbind(df_salmon,df_salmon_y_i)
```

TASK 4: Using a `for` loop, simulate 100 years of the salmon population with initial population distribution $v_{ex6} = [1000, 500, 250, 125]$. The result should be a data.frame with the first column denoting year, and four additional columns (one for the abundance of each year-class).

3. Modifying 4-year Salmon Runs

Frasier River salmon don't spawn at 4 years with complete fidelity, however. Every cohort, a fraction spend an extra year as juveniles and spawn at age 5. We can modify the Leslie matrix above to incorporate this detail.

```
M_salmon_2 <- rbind(
  c(0,0,0,2000*.9,2000),
  c(0.005,0,0,0,0),
  c(0,0.2,0,0,0),
  c(0,0,0.5,0,0),
  c(0,0,0,0.1,0)
)
```

Of course, we also need to add another element to the population vector for the age-5's and data.frame() as well. For simulating the modified Leslie matrix, we can start with no Age-5 salmon, but we still need to put in a 0. $v_{ex7} = [1000, 500, 250, 125, 0]$.

Simulation

TASK 5: Using a `for` loop, simulate 100 years of the salmon population with initial population distribution $v_{ex7} = [1000, 500, 250, 125, 0]$. The result should be a data.frame with the first column denoting year, and five additional columns (one for the abundance of each year-class).

What is happening different in this simulation?

Running it backwards.

We talked about the matrix inverse in class, a concept that is easier to explain than it is to calculate (even for a computer!). R can still compute the inverse, usually (assuming it's defined). It may not be obvious why, but **R** hides the matrix inverse under the general `solve()` function. In fact the matrix inverse version of `solve()` is actually a totally different function in the `Matrix` library. [**Tangent: R** will *overload* a function name if the base function is expecting a particular kind of input. In this case, the *base R* version of `solve()` does not expect matrix input. This lets **R** also put a method for `solve()` that accepts matrix input. Tangent over.]

Let's ask R what the inverse of our salmon models are.

```
solve(M_salmon_1)
```

```
##       [,1] [,2] [,3] [,4]
## [1,] 0e+00  200    0    0
## [2,] 0e+00    0    5    0
## [3,] 0e+00    0    0    2
## [4,] 5e-04    0    0    0
```

```
solve(M_salmon_2)
```

```
##       [,1] [,2] [,3] [,4] [,5]
## [1,] 0e+00  200    0    0    0
## [2,] 0e+00    0    5    0    0
```

```
## [3,] 0e+00    0    0    2    0
## [4,] 0e+00    0    0    0   10
## [5,] 5e-04    0    0    0   -9
```

TASK 6: Apply the inverse matrix to the final year of the simulations you ran before.

What value did you get?
Explain the result in terms of the basic concept of the matrix inverse.

4. Eigenvalues in action: stable age distributions of Leslie Matrices

Truthfully, the matrices above were very carefully chosen. Notice that the age distributions may shift, but the population as a whole seems stable. I engineered things that way. The fundamental issue with Leslie matrices is they don't incorporate *density dependence* in the dynamics. There is no carrying capacity, there is no compensatory mechanism to provide diminishing returns on reproduction and population growth. For example, lets expand on the salmon model to an iteroparous population, where individuals can reproduce (spawn) as they age through different classes.

```
M_notsalmon_1 <- rbind(
  c(0,0,0,2000*.5,2000),
  c(0.005,0,0,0,0),
  c(0,0.2,0,0,0),
  c(0,0,0.5,0,0),
  c(0,0,0,0.8,0)
)
```

TASK 7: Iterate the population dynamics of M_notsalmon_1 for 100 steps (years) using the same $v_{ex7} = [1000, 500, 250, 125, 0]$ as above.

Plot the abundance of Age-4 fish across the 100 year simulation. What happens?

The Leslie matrix is simulating exponential growth. There is an effective growth rate of the population hiding in the matrix causing this. It's the **dominant eigenvalue**. Let's ask R what the eigenvectors and eigenvalues of `M_notsalmon_1` are.

```
eigen(M_notsalmon_1)
```

```
## eigen() decomposition
## $values
## [1]  1.0585807+0.0000000i  0.2145442+0.9831614i  0.2145442-0.9831614i
## [4] -0.7438345+0.4393282i -0.7438345-0.4393282i
##
## $vectors
##                 [,1]                         [,2]                         [,3]
## [1,] 0.9999883077+0i -0.9999869722+0.0000000000i -0.9999869722+0.0000000000i
## [2,] 0.0047232502+0i -0.0010593218+0.0048544051i -0.0010593218-0.0048544051i
## [3,] 0.0008923742+0i  0.0008977351+0.0004113955i  0.0008977351-0.0004113955i
## [4,] 0.0004214956+0i  0.0002948109-0.0003922221i  0.0002948109+0.0003922221i
## [5,] 0.0003185364+0i -0.0002546762-0.0002954632i -0.0002546762+0.0002954632i
##                             [,4]                         [,5]
```

```
## [1,]  9.999818e-01+0.0000000000i  9.999818e-01+0.0000000000i
## [2,] -4.983398e-03-0.0029433254i -4.983398e-03+0.0029433254i
## [3,]  6.468553e-04+0.0011734422i  6.468553e-04-0.0011734422i
## [4,]  2.302896e-05-0.0007751776i  2.302896e-05+0.0007751776i
## [5,] -3.834250e-04+0.0006072489i -3.834250e-04-0.0006072489i
```

The largest real eigenvalue is about `1.0586`. If we store the results of `eigen` we can also access the eigenvector that corresponds to it.

```
eig_dec_M_notsalmon_1 <- eigen(M_notsalmon_1)
eig_vec_1_M_notsalmon_1 <- eig_dec_M_notsalmon_1$vectors[,1]
print(eig_vec_1_M_notsalmon_1)
```

```
## [1] 0.9999883077+0i 0.0047232502+0i 0.0008923742+0i 0.0004214956+0i
## [5] 0.0003185364+0i
```

Recall that when you multiply an eigenvector $\vec{e}$ of a matrix M by that matrix, the result will be $\vec{e}$ scaled by the corresponding eigenvalue. Thus, if we multiply this age distribution, `eig_vec_1_M_notsalmon_1` by the Leslie matrix model, each age distribution shoulod grow by a factor of approximately `1.0586`.

Also note, however, that R is returning complex numbers. If when you start with real-valued elements of a matrix, you can get complex numbers coming out (just like when you take square-roots of single numbers!). Any Leslie matrix will return a real-valued eigenvector (because of all those 0's they are very well behaved), but R will think of it as a complex number until you tell it not to. So we explicitly tell R just to take the "real part" with `Re()`.

```
N_eigenfish_1 <- Re(eig_vec_1_M_notsalmon_1)
N_eigenfish_2 <- M_notsalmon_1 %*% N_eigenfish_1

N_eigenfish_2/N_eigenfish_1
```

```
##          [,1]
## [1,] 1.058581
## [2,] 1.058581
## [3,] 1.058581
## [4,] 1.058581
## [5,] 1.058581
```

If you want to look at the ratio of age-classes across the years of the model, you can either add columns to the existing data frame with the ratios. Adding a column to a data frame is as easy as assigining data to a column that doesn't exist yet `my_data_frame$new_column <- data_to_go_in_new_column`. If you do this for each age class, adding a colum with the ratio of individuals in that age class to total individuals that year, you end up with 5 more columns. It would also be entirely reasonable to make a new data frame with just the `Year` column and the fraction of individuals in each age class.

```
df_salmon$frac_N_1 <- df_salmon$N_1/(df_salmon$N_1 + df_salmon$N_2 + df_salmon$N_3 + df_salmon$N_4 + df
df_salmon$frac_N_2 <- df_salmon$N_2/(df_salmon$N_1 + df_salmon$N_2 + df_salmon$N_3 + df_salmon$N_4 + df
df_salmon$frac_N_3 <- df_salmon$N_3/(df_salmon$N_1 + df_salmon$N_2 + df_salmon$N_3 + df_salmon$N_4 + df
df_salmon$frac_N_4 <- df_salmon$N_4/(df_salmon$N_1 + df_salmon$N_2 + df_salmon$N_3 + df_salmon$N_4 + df
df_salmon$frac_N_5 <- df_salmon$N_5/(df_salmon$N_1 + df_salmon$N_2 + df_salmon$N_3 + df_salmon$N_4 + df
```

TASK 8: Simulate the population for 100 years starting with the population distributed as the first eigenvector of `M_notsalmon_1`.

Plot the resulting time-series of the Age-3, Age-4, and Age-5 year classes together on the same plot. The abundances are all growing exponentially, but something is being preserved. Try plotting again but dividing each by the total number of individuals that year. [This is a good place to ask for help if you are unsure how to proceed].

The largest eigen-*value* of a Leslie matrix tells you a characteristic rate of exponential growth. The eigen-*vector* that corresponds to that value tell you the stable *age distribution* of the population. If you start the population with a different age-distribution than the stable age distribution, the population will tend to approach the stable age distribution through time, even as its overall growth is at an exponential rate.

TASK 9: Simulate the population for 100 years starting from at least two other initial distributions not equal to the eigenvector distribution above.

Plot the abudances of Age-3, Age-4, and Age-5 through time, but as before divide by the total number of individuals in that year. Is the population distribution tending towards our previous result?

Optional Exercises:

In the fisheries context, Leslie matrices aren't very usable off the shelf, since exponential growth isn't realistic behavior of real populations. However, the idea of static survival through adult phases is held onto, with a compensatory function like the Ricker model put into the fecundity. The original salmon matrix would then become something like:

$$M = \begin{bmatrix} 0 & 0 & 0 & \exp(7.6(1-N_4)) \\ 0.005 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \end{bmatrix}$$

When there are very few adults, the growth rate is similar to what we had before, $\exp(7.6) \approx 2000$, but as more adults are present in year y, fewer new Age-1's are created per adult in year $y+1$.

This is a model you can also simulate in the computer, but the matrix is changing slightly every year. You can work from your code before, but you will need a function to calculate M as a function of N. (And really it's just that one matrix element that needs to be recalculated).

Reflection Questions:

Why didn't our work from Sections 2 and 3 create exponential growth or decay? Check the eigenvalues!

Fisheries generally target adults above a certain length or age. How would you incorporate/represent fishing mortality into a Leslie matrix model? How do you think fishing on only the oldest fish would impact the stable age distribution of the modeled population?