Intro to R: Week 7

Topics Covered: Matrix operations, understanding a script, review

This week, you will need the following packages:

Task 1: Matrix operations
This example is based on Leslie matrices, which can be used to model population growth. It is not crucial to know how they work to understand the assignment.

Step 1.1 Create a Leslie matrix (`L`) and an initial population vector (`n0`) that look like the following:

$$L = \begin{bmatrix} 0 & 3 & 1 \\ 0.5 & 0 & 0 \\ 0 & 0.8 & 0 \end{bmatrix}$$

$$n_0 = \begin{bmatrix} 20 \\ 0 \\ 0 \end{bmatrix}$$

Step 1.2 We will follow this population for 30 generations. Set a variable `n_gen <- 30`. *Note that `L`, `n0`, and `n_gen` will be the only variables you will need to edit for this script.*

Step 1.3 Use the function `length()` to set a variable `a_class` equal to the number of age classes in the population vector.

Step 1.4 With Leslie matrices, it is possible to obtain the population vector for the next generation (time t+1) by multiplying the Leslie matrix and the current population vector (`L`×`n0`). Test that you can get an answer for `L`×`n0` that has the same structure as vector `n0`.

Step 1.5 Initialize a matrix (`pop`) that will be used to store our population vectors as we calculate them. Make `pop` a 3×30 (`a_class` × `n_gen`) matrix filled with NAs.

Step 1.6 FOR loop: set up a loop to calculate `n` at time t+1 by multiplying `L` by `n` at time t. Note: you should store the vectors in `pop`, but start at column 2. (Save column 1 for your initial population vector).

Step 1.7 IF statement: add a statement to your loop that stores your initial population vector (n) in the first column.

Step 1.8 Now, convert the transpose of `pop` into a data frame called `pop.df`, rename the columns "Babies", "1 yo", and "2 yo", and add a column with the generations 0 to `n_gen` - 1.

Step 1.9 Use the function `melt()` and `ggplot()` to plot your results. Assign a different color to each line.

Task 2: Understanding a script
Imagine that you are given the following model by a graduate student who just left for a month-long cruise. You cannot wait until s/he is back to start on your work. The script contains a code to model the predation on mice by a cat

Step 2.1 Copy-paste the following code into a script (you should use the .Rmd file provided), make sure it runs, and try to understand it. *Note: be careful when you play with numbers. Mice reproduce fast, and your model can quickly take a lot of time to run. If you get bored, just abort and try again with lower values.*

```r
## Assumptions (based on some quick Googling):

# the cat is introduced only after a certain number of mice were seen
# mice are mature after 50 days (not counted in population before)
# average 6-8 mice per litter, will use 7 in the model
# 5-10 litters per year, will assume one every 50 days in the model
# death caused by cat only
# 50% of mice are female at all time
# half of the mice give birth on day 0 (thus mature on day 50)

# input:
n.mice0 <- 100  # number of mice on day 0
p.eaten <- 0.1  # probability of cat killing each mouse per day
max.mice <- 3  # maximum number of mice the cat can kill per day
n.days <- 365  # days to run model for
n.runs <- 10  # number of model runs

# model
d.new <- seq(50, n.days, 50)  # days when the most recent litter matures
pop <- as.data.frame(matrix(NA, nrow = n.days+1, ncol = n.runs))
names(pop) <- paste(rep("run", n.runs), seq(1:n.runs))

for (r in 1:n.runs){

  n.mice <- n.mice0

  for (d in 0:n.days){

    ## Cannot have negative number of mice:
    if (n.mice < 0){
      n.mice <- 0
    } # end if

    ## Are there new mice born?

    if (any(d == d.new)){
      n.mice = n.mice + floor(n.mice/2)*7;  # one litter for each female present
    }  # end if

    ## Do the mice survive the day?

    dead.mice <- sample(1:(1/p.eaten), n.mice, replace=TRUE)

    # kills as many mice as there are 1s, or from one to max number of mice
    n.mice <- n.mice - min(sample(max.mice, 1), length(which(dead.mice==1)))

    # store number of mice every day:
    pop[d+1, r] <- n.mice

  }  # end d

}  # end r
```

```r
# add a column with day value
pop$days <- 0:n.days

# melt data and obtain mean and SD for each day
pop.m <- melt(pop, id.vars=c("days"))
summary <- summarize(group_by(pop.m, days), Mean=mean(value), SE=sd(value))

# plot data
ggplot(summary, aes(x=days, y=Mean))+
  geom_errorbar(aes(ymin=Mean-SE, ymax=Mean+SE), color="purple")+
  geom_line()+
  xlab("Days")+ # change x-axis label
  ylab("Number of mice")+ # change y-axis label
  theme_bw()+ # get rid of gray background
  theme(panel.grid.major=element_blank(), # get rid of major
      panel.grid.minor=element_blank()) # and minor grid lines
```

Questions to help you think about the code: *you can focus on the mechanics of the code (what does each line do?) and/or the thinking behind it (why is it using such and such function?)*

- What is n.runs?
- What does the purple area on the graph represent? How do you change it to a different color?
- Why is `n.mice <- n.mice0` necessary? Why is it in the `r`-loop (first loop)?
- Why is the function `floor()` used in this statement `floor(n.mice/2)*7`?
- What does `sample(1:(1/p.eaten), n.mice, replace=TRUE)` do? *More on this next week.*
- What statement determines how many mice are killed by the cat every day?
- How would you model the population growth of these mice when there is no cat?
- What happens when you are melting and summarizing the data?

Task 3: Review
Since the beginning of this course, you have seen a lot of material. Here are some exercises to refresh your memory. Some will be easier than others.

Step 3.1 Use the function `sample()` to create a vector `s` with 100 numbers, each randomly selected from 1 to 5 (use the help menu to make it work). *More on random numbers next week.*

Step 3.2 In which positions can you find a value equal to 5? Use these to index `s` to make sure that you are correct. Calculate the percentage of values that are equal to 5? Are any values in `s` equal to 10? Repeat the first exercises for the the values that are not equal to 5?

Step 3.3 Run the top part of the code below to generate data, then calculate the mean of the column `coral`.

```r
# Generate data
islands <- c("Kingman", "Palmyra", "Tabuaeran", "Kiritimati")
inhabited <- c(FALSE, FALSE, TRUE, TRUE)
coral <- c(44, 20, 20, 15)
fish <- c(5, 3, 2, 1)
island.df <- data.frame(islands, inhabited, coral, fish)
```

Step 3.4 Write a FOR loop with 100 iterations that prints "I am here, in loop number", and the number of the loop every 5 loops (i.e. at iteration 5, 10, etc.)

Step 3.5 Write a function for each of the two population models we worked with today. Do not include the melting and/or plotting of the data in your functions, simply generate a population vector using the required inputs.

Step 3.6 Run the top part of the code below to generate data, then convert each row to a time format.

```r
Year <- rep(2012, 24)
Month <- rep(seq(1:12), each = 2)  # what does each do?
Day <- rep(c(1, 15), 12)

time.df <- data.frame(Year, Month, Day)
```

Step 3.7 Modify some of the `ggplot()` codes used this week. For instance, customize the colors of the lines and the location of the legend in the plot for the Leslie matrix population. Plot the three lines in three different plots.