Elephant Age-Based Population Model

Therese Donovan

2020-03-15

Overview

This script is a translation of the elephant age-based population model we did with spreadsheets. The model is a *female-only* population model that has a post-breeding census. That is, we count individuals immediately after the birthday (and all animals are assumed to be born on the same day in a birth pulse). Thus, our count includes 0, 1, 2, . . . 59 year olds. We assume that all individuals that are age 12 or older can reproduce.

The Elephant Model ——

Our Elephant model will be a function called `popElephant()`. We certainly aren't required to build our model as a function. We could just have a script where you create new objects at the top of your script, and then use those objects later in the script. However, we already know that our model has inputs (shaded green in the spreadsheet) and outputs (shaded blue in the spreadsheet), and that we want to be able to quickly use our model to perform experiments with it. A function is perfect for that.

If you recall, our model has several inputs. Here, we will let the `popElephant()` function have six arguments: *starting.pop* (a vector of starting female population sizes by age class), *ci* (the calving interval; the number of years between subsequent births), *offspring.sex.ratio* (the proportion of the offspring that are female), *s.calf* (the survival rate of calves), *s.adult* (the survival rate of adults), , and *plot.traj* (TRUE or FALSE; should a plot of the population trajectory be displayed). Notice that the *plot.traj* argument has been set to TRUE in the function definition. This means that this argument has a default value of TRUE. The other three arguments do not have default values. The user of the function will need to provide these values.

```
# create a new function call popElephant.
# it returns the vector of total population sizes of Rocs with each time step.
# never forget to look for each newly created object in the environment!

popElephant <- function(starting.pop,
                        ci,
                        offspring.sex.ratio,
                        s.calf,
                        s.adult,
                        plot.traj = TRUE){

  # convert the calving interval to a fecundity rate
  fecundity = 1/ci

  # create a matrix that will store the population size by age
  # the rows are the years
  # the columns give the age classes from 0 to 59
  pop <- matrix(data = NA, nrow = 80, ncol = 60)

  # add column names to avoid confusion
```

```
  colnames(pop) <- as.character(0:59)

  # set the initial population vector
  pop[1,] <- starting.pop

  # a loop for generating the population through time
    for (y in 2:nrow(pop)) {

      # create a vector of survival rates
      # the first entry is the calf survival rate
      # the next 59 entries are the adult survival rates for age classes 1:60
      # the final adult age class has a survival rate of 0; they drop out of the model
      survival.rate <- c(s.calf, rep(s.adult, times = 58), 0)

      # compute survivors in next time step; notice this is length 61
      survivors <- pop[y - 1, ] * survival.rate

      # add the survivors to the population matrix; notice the offset here
      pop[y, 2:60] <- survivors[1:59]

      # add in  the total births; 27.871
      indices <- match(x = as.character(12:59), table = colnames(pop))
      pop[y, "0" ] <- sum(pop[y, indices]) * fecundity * offspring.sex.ratio

    } # end of loop


  # summarize the results
  total.pop <- rowSums(pop, na.rm = T)

  # get growth rate for each year:  lambda = n(t+1)/n(t)
 lambda <- total.pop[-1]/head(total.pop, -1)

  # plot a line graph
  if (plot.traj == TRUE) {
  plot(x = 1:nrow(pop), y = total.pop,
       type = "b",
       col = "blue",
       xlab = "Year", ylab = "Number of Female Elephants",
       ylim = c(0, max(total.pop) + 10),
       main = "Total Female Elephant Population Size over Time")



 }

  # return the vector of population sizes
  return(list(total.pop = total.pop, lambda = lambda))

} # end of function
```

The starting population vector is something that the user of the function (you!) will need to provide. You might recall that we our spreadsheet model started with 40 0-year-olds, 20 individuals for age classes 1-5, and 4 individuals for age classes 6-60. Hand-typing 61 entries can be time-consuming. Here, we'll make use

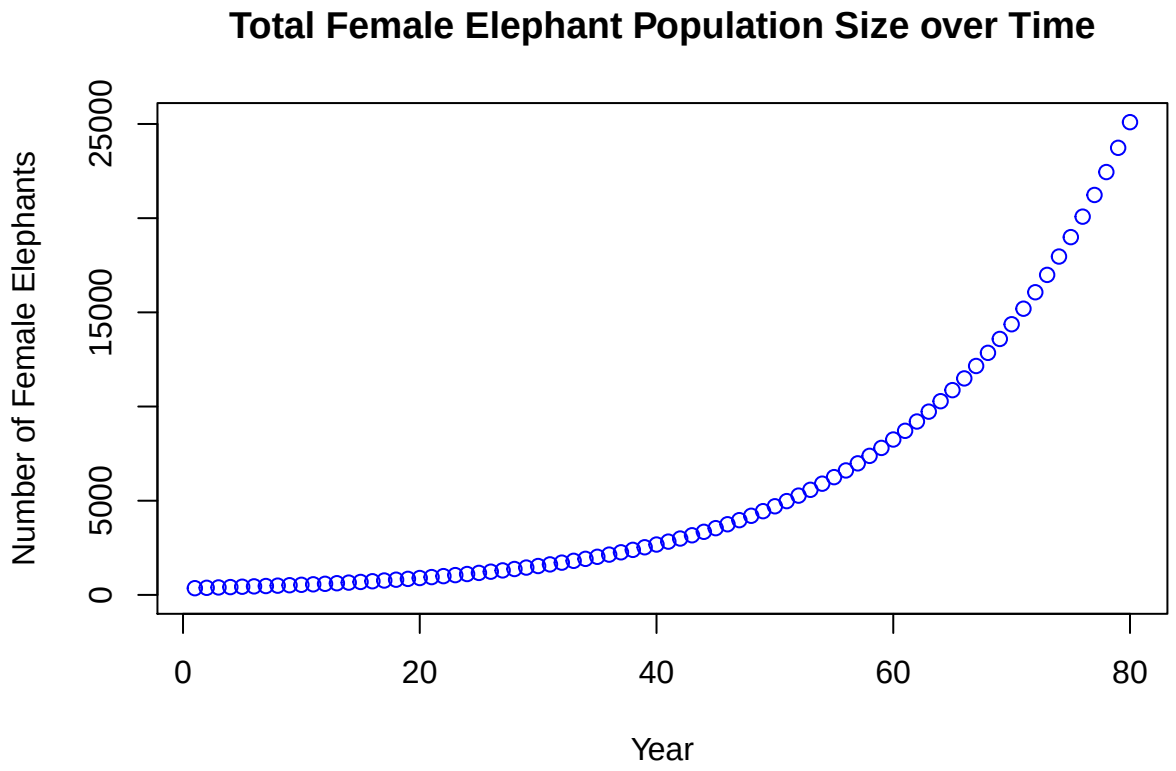
of the `rep()` function to repeat values a certain number of times, like this:

```
c(40, rep(20, times = 5), rep(4, times = 54))
```

```
##  [1] 40 20 20 20 20 20  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
## [26]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
## [51]  4  4  4  4  4  4  4  4  4  4
```

To test your function, call it (just like Excel) and send in some values for the arguments:

```
# to run this function, do something like this:
popElephant(starting.pop = c(40, rep(20, times = 5), rep(4, times = 54)),
            ci = 3.1,
            offspring.sex.ratio = 0.5,
            s.calf = 0.9,
            s.adult = 0.99,
            plot.traj = TRUE)
```




```
## $total.pop
##  [1]   356.0000   375.5381   395.4545   414.9351   433.9866   452.6155
##  [7]   470.8284   491.0372   512.9595   536.5373   561.7141   588.4342
## [13]   618.9826   651.8632   687.1009   724.6145   764.3245   806.1530
## [19]   850.0239   896.1752   944.8059   996.1054  1050.2539  1107.4227
## [25]  1168.0782  1232.4878  1300.9193  1373.6276  1450.8545  1532.8292
## [31]  1619.7689  1711.9195  1809.5504  1912.9526  2022.4382  2138.3378
## [37]  2261.0398  2390.9633  2528.5580  2674.3022  2828.7011  2992.2843
## [43]  3165.6047  3349.2412  3543.8024  3749.9282  3968.2931  4199.6075
## [49]  4444.6254  4704.1475  4979.0252  5270.1634  5578.5236  5905.1268
## [55]  6251.0552  6606.7653  6983.0396  7381.2004  7802.6443  8248.8457
## [61]  8710.9643  9205.1166  9728.4551 10282.8360 10870.2218 11492.6872
## [67] 12152.4266 12850.1783 13588.0976 14368.4944 15193.8417 16066.7840
## [73] 16988.6069 17962.7978 18992.4372 20080.8226 21231.4799 22448.1756
```

```
## [79] 23734.9301 25095.8000
##
## $lambda
##  [1] 1.054882 1.053035 1.049261 1.045914 1.042925 1.040239 1.042922 1.044645
##  [9] 1.045964 1.046925 1.047569 1.051915 1.053120 1.054057 1.054597 1.054802
## [17] 1.054726 1.054420 1.054294 1.054265 1.054296 1.054360 1.054433 1.054772
## [25] 1.055141 1.055523 1.055890 1.056221 1.056501 1.056718 1.056891 1.057030
## [33] 1.057143 1.057234 1.057307 1.057382 1.057462 1.057548 1.057639 1.057734
## [41] 1.057830 1.057922 1.058010 1.058091 1.058165 1.058232 1.058291 1.058343
## [49] 1.058390 1.058433 1.058473 1.058511 1.058547 1.058581 1.056904 1.056953
## [57] 1.057018 1.057097 1.057186 1.056022 1.056728 1.056853 1.056986 1.057123
## [65] 1.057263 1.057405 1.057417 1.057425 1.057432 1.057441 1.057454 1.057374
## [73] 1.057344 1.057321 1.057306 1.057301 1.057306 1.057321 1.057336
```

Now we're talking. This graph summarizes the results of our age-based population model for female elephants.

Explore Some More

Reflection

Now that you have an R version of Roc flu model, ponder these questions:

- What information should you present to the Caliph?
- Why is it important to run multiple trials?
- How do you get from your model world back to the real world?