# Roc Individual Based Stochastic Model

Therese Donovan

2020-02-27

## Overview

This script is a translation of the Roc individual-based stochastic model we built in class to R. This model could be a simple R script, but I have chosen to build it in RMarkdown so that we can weave narrative along with calls to R.

We've been working on the Roc population model for two weeks now. Let's begin by reviewing that model.

## The Roc Model ——

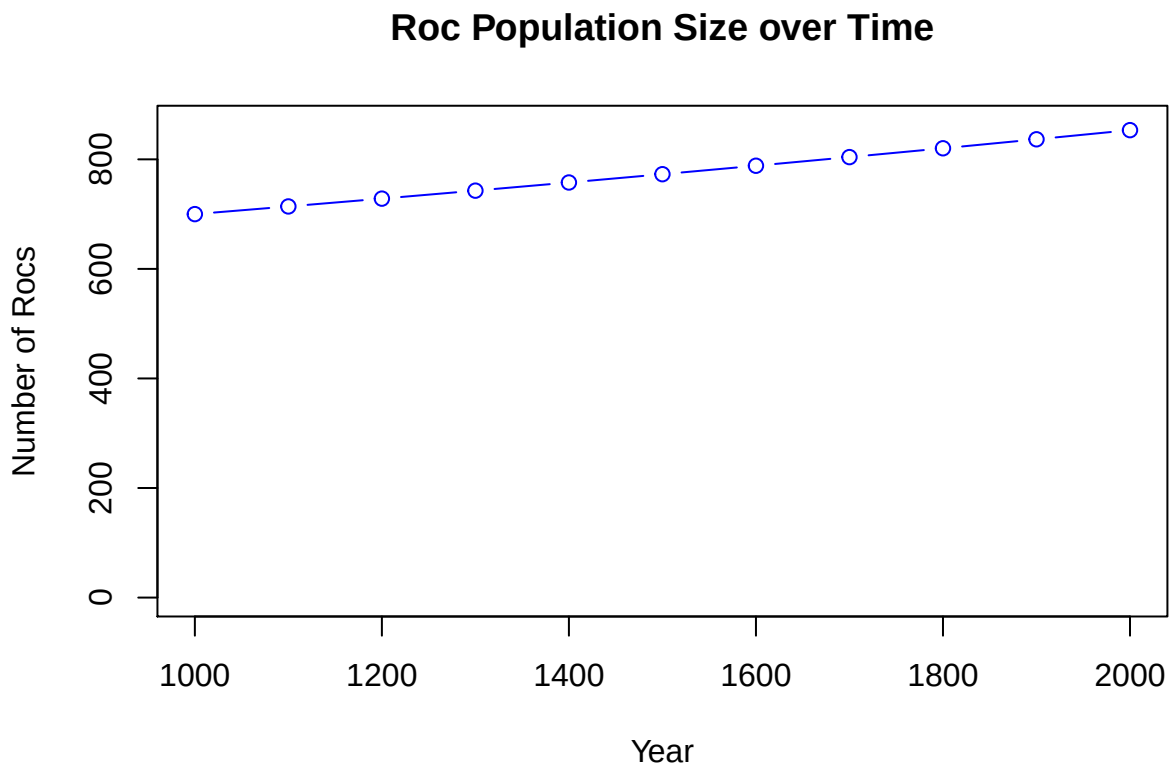
Our Roc model is a function called `popRoc()`. Here's the original function; it has four arguments named *bullets*, *clutch.size*, *starting.pop*, and *plot.traj*. Notice that the *plot.traj* argument has been set to TRUE in the function definition. This means that this argument has a default value of 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 popRoc.
# it has four arguments: bullets, clutch.size, starting.pop, and plot.traj
# it returns the vector of population sizes of Rocs with each time step.

popRoc <- function(bullets, clutch.size, starting.pop, plot.traj = TRUE){

  # set the trajectory with the seq() function
  years <- seq(from = 1000, to = 2000, by = 100)

  # create a vector called rocs that will store our population trajectory
  rocs <- vector(mode = "numeric", length = length(years))

  # set the starting population
  rocs[1] <- starting.pop

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

      # this is your main roc model
      # in words: rocs in time step y is the is the number of rocs in the previous
      # time step minus those hit by bullets; this result is multiplied by clutch size.
      rocs[y] <- (rocs[y - 1] - bullets) * clutch.size
    }

  # plot a line graph
  if (plot.traj == TRUE) {
  plot(x = years, y = rocs,
       type = "b",
```

```
       col = "blue",
       xlab = "Year", ylab = "Number of Rocs",
       ylim = c(0, max(rocs) + 10),
       main = "Roc Population Size over Time")
  }

  # return the vector of population sizes
  return(rocs)

} # end of function
```

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:
popRoc(bullets = 0, clutch.size = 1.02, starting.pop = 700, plot.traj = TRUE)
```




```
##  [1] 700.0000 714.0000 728.2800 742.8456 757.7025 772.8566 788.3137 804.0800
##  [9] 820.1616 836.5648 853.2961
```

This model works fine, as long as the population size is large enough and that the input rates, such as clutch.size, can realistically apply to a given population.

But, many of the species we work with in conservation, such as endangered species, have low population sizes. This is the case of our contemporary Roc population, which numbers 2 in total. When you deal with small population sizes, the law of averages (such as average clutch.size) fails to capture what actually happens and may lead you astray: a single Roc cannot produce 1.03 chicks (the average clutch size of the population). Instead, a single Roc can lay 0, 1, or 2 eggs. When we get into the realm of identifying the exact number of offspring that are produced, or whether an individual lives or dies, we are no longer invoking population averages. Instead, we've entered the realm of individual based modeling, and in particular the world of demographic stochasticity. With individual based models, we let individuals reproduce an integer number of offspring (0, 1, 2, . . .) and we let them live or die in a time step (0 or 1). We can track each individual by

name if needed, or we can simply count total births and deaths as we'll do here.

## Building a Roc Individual Based Model (IBM) ——

We will modify our `popRoc()` function fairly substantially to account for current conditions: Our new function will be called `popRocIBM()` to keep it separate from the original `popRock()` model. This model will include demographic stochasticity. Step 1 is to get rid of arguments we no longer need, including *bullets*.

Step 2 will involve an important change to the *clutch.size* argument: Instead of entering a population clutch size (rate), we will now need some emperical data that contains reproduction information. In other words, *clutch.size* should be replaced by *clutch.data*. In the lecture video, Tony presented a dataset of clutch size information. Let's make this dataset now as a dataframe in R. A dataframe is really a list of vectors, each of the same length. Each vector can hold only one type of data. For instance, one vector may hold integers, another may hold characters, and so on. You can safely imagine a spreadsheet that holds a dataset, where each column is a vector.

```
# use the data.frame() function to create a dataframe
data <- data.frame("eggs" = c(0:2), "nests" = c(12, 79, 18))

# have a look at the structure
str(data)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ eggs : int  0 1 2
##  $ nests: num  12 79 18
```

```
# show the data
data
```

```
##   eggs nests
## 1    0    12
## 2    1    79
## 3    2    18
```

This is an empirical dataset. "Empirical implies that the information is based on experience, and data is information we gather about something. Thus the information acquired by scientists through experimentation and observation is called empirical data." – https://study.com/academy/lesson/empirical-data-definition-example.html

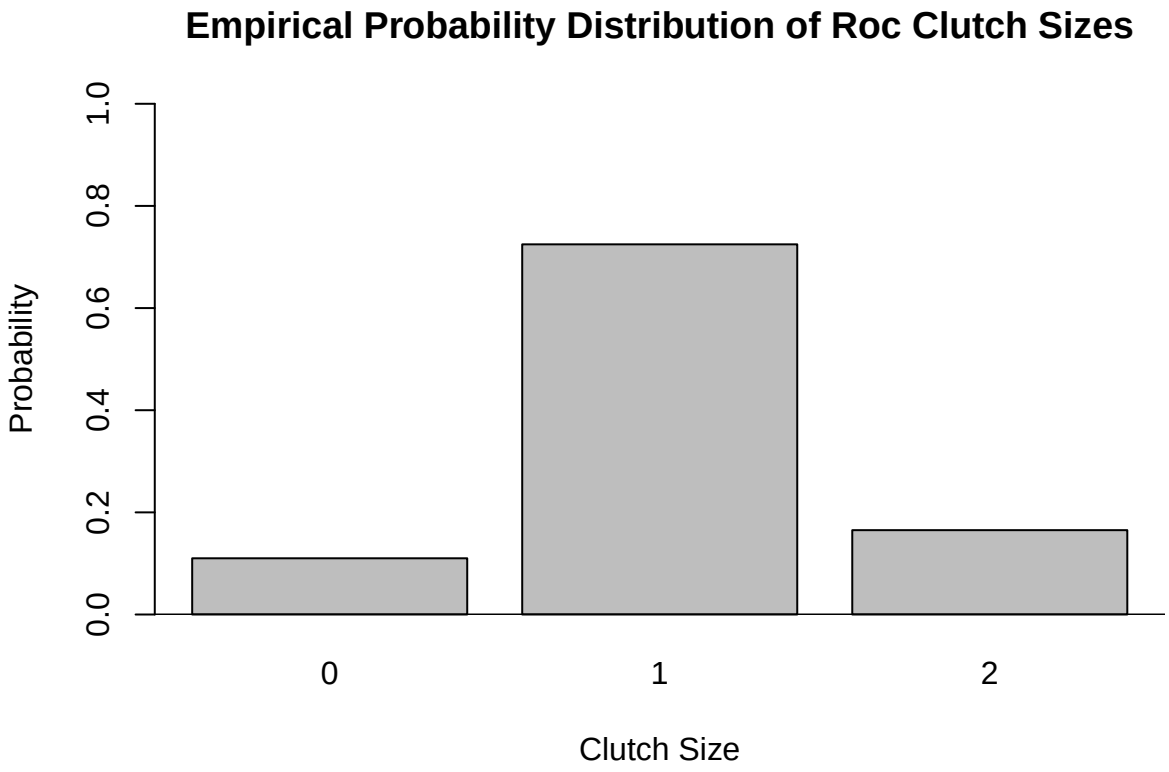
At this point, we can compute the probability of each clutch outcome. Recall that with dataframes, a single column can be created (or accessed) by typing in the name of the dataframe, followed by the $ symbol and the column name.

```
# add a column called probability, which is the nest column divided by the total nests
data$probability <- data$nests/sum(data$nests)

# have a look
data
```

```
##   eggs nests probability
## 1    0    12   0.1100917
## 2    1    79   0.7247706
## 3    2    18   0.1651376
```

```
# make a histogram of the data
barplot(data$probability,
        main = "Empirical Probability Distribution of Roc Clutch Sizes",
        ylab = "Probability",
```

```
        ylim = c(0,1),
        names.arg = as.character(data$eggs),
        xlab = "Clutch Size")

abline(h = 0)
```




Next, we can add in cumulative probability vector with the `cumsum()` function.

```
data$cumprob <- cumsum(data$probability)

# have a look
data
```

```
##   eggs nests probability   cumprob
## 1    0    12   0.1100917 0.1100917
## 2    1    79   0.7247706 0.8348624
## 3    2    18   0.1651376 1.0000000
```

This is known as an "empirical probability distribution" becomes it is from an empirical dataset. That is, it is not from a probability distribution function such as the normal distribution or binomial distribution.

We can use this distribution to assign a clutch size for each individual in the population. We assume that each individual has an 11% chance of having 0 offspring, a 72% of having 1 offspring, and a 17% chance of having 2 offspring. In our spreadsheet, we drew a random number between 0 and 1 for each individual. In R, this can be done with the `runif()` function, which means draw a random number from a uniform distribution.

```
rand <- runif(n = 1, min = 0, max = 1)
rand
```

```
## [1] 0.8356676
```

Tony talked about how you would find where this random number falls in the cumulative distribution, and return the number of offspring for that individual. The `cumsum()` function can be used to create this

distribution:

```
data$cumulative <- cumsum(data$probability)

# have a look
data
```

```
##   eggs nests probability   cumprob cumulative
## 1    0    12   0.1100917 0.1100917  0.1100917
## 2    1    79   0.7247706 0.8348624  0.8348624
## 3    2    18   0.1651376 1.0000000  1.0000000
```

Cumulative distributions always end with the number 1. We would now need a way to have R find where 0.8356676 falls in the cumulative distribution, and return the number of eggs for that individual.

In R, the `sample()` function allows us to do this easily. Take a look at the help page of this function. This function has a few required arguments:

- $x$ is the vector of items that you draw samples from. Here, we want to draw a number of eggs.
- *size* is the number of samples you want to draw.
- *prob* - the probability associated with each item of $x$. That is, the probability of have 0 eggs, 1 egg, or 2 eggs.
- *replace* - should the sampling be done with replacement? For this model, yes.

```
# draw 1 sample
sample(x = data$eggs, size = 1, prob = data$probability, replace = TRUE)
```

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

```
# draw another sample
sample(x = data$eggs, size = 1, prob = data$probability, replace = TRUE)
```

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

```
# draw 5 samples
sample(x = data$eggs, size = 5, prob = data$probability, replace = TRUE)
```

```
## [1] 1 1 0 1 1
```

You've probably guessed that the *size* argument relates to the number of Rocs in our population. If our current population is 5 individuals, each individual will produce 0, 1, or 2 eggs following our empirical probability distribution.

Our new function, `popRocIBM()`, is really quite simple. The user will pass in a dataframe with clutch size information (*clutch.data*), the *starting.pop* size, and will identify if they'd like a trajectory plot or not (*plot.traj*). The function is shown below; note that many elements of our original `popRoc()` function appear once again:

```
popRocIBM <- function(clutch.data,
                      starting.pop,
                      plot.traj = TRUE
                      ){

  # create the empirical probability distribution
  clutch.data$probability <- data$nests/sum(data$nests)

  # set the trajectory with the seq() function
  years <- seq(from = 1000, to = 2000, by = 100)

  # create a vector called rocs that will store our population trajectory
```

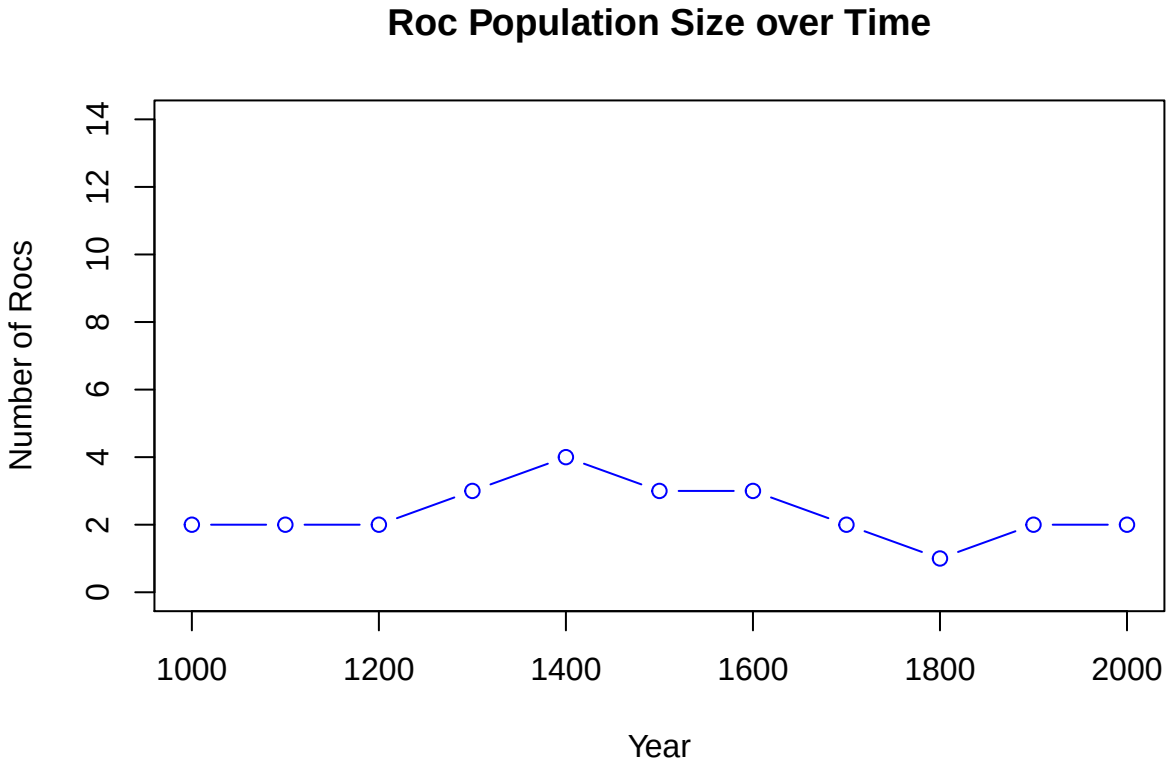
```
  rocs <- vector(mode = "numeric", length = length(years))

  # set the starting population
  rocs[1] <- starting.pop

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

      # this would be your main roc model if population size was not a problem
      # rocs[y] <- (rocs[y - 1]) * clutch.size

      # this is the individual-based approach
      rocs[y] <- sum(sample(x = clutch.data$eggs,
                            size = rocs[y - 1],
                            prob = clutch.data$probability,
                            replace = TRUE))
    }  # end of loop

  # plot a line graph
  if (plot.traj == TRUE) {
  plot(x = years, y = rocs,
       type = "b",
       col = "blue",
       xlab = "Year", ylab = "Number of Rocs",
       ylim = c(0, max(rocs) + 10),
       main = "Roc Population Size over Time")
  }

  # return the vector of population sizes
  return(rocs)


} # end of function
```

That's it! This function:

- asks the user to provide clutch data in the form of a dataframe. You would need to tell your user that your function expects this dataframe to have columns that are named "eggs" and "nests" at a minimum. The function uses these names in its calculations, so if the names are off, the function will throw an error;
- asks the user to provide the starting population size;
- computes the empirical probability distribution of clutch sizes;
- creates two vectors that will hold the years and the Roc population sizes;
- sets the first element of the **rocs** vector to the starting population size;
- initiates a loop to loop through years. For a given year, the total offspring is computed by using the `sample()` function to draw samples from the empirical probability distribution (one for each Roc), and then sums them.
- creates a plot if the user sets *plot.traj* to TRUE (which is the default).

Let's test it!

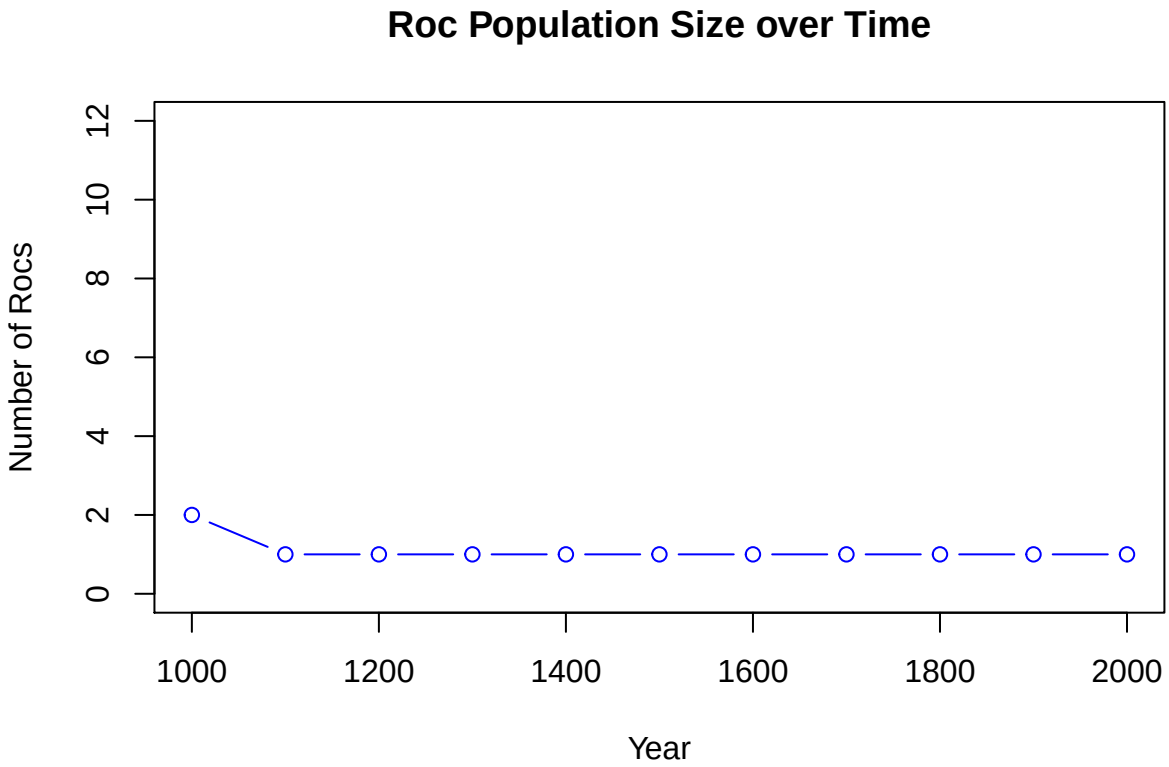
```
popRocIBM(clutch.data = data, starting.pop = 2, plot.traj = TRUE)
```




```
##  [1] 2 2 2 3 4 3 3 2 1 2 2
```

Do it again!

```
popRocIBM(clutch.data = data, starting.pop = 2, plot.traj = TRUE)
```




```
##  [1] 2 1 1 1 1 1 1 1 1 1 1
```

## Running Multiple Trials

These are just two possible realizations out of many alternative realizations. As with all simulations that involve stochasticity, you will want to run this model many times and summarize the results. Why not do with with a new function?

```
simIBM <- function(trials, clutch.data, starting.pop, plot.traj = FALSE){

  # set up a dataframe to store the results
  results <- data.frame("Trial" = integer(), "Rocs" = integer())

  # loop through the trials
  for (i in 1:trials) {

    # run popRocIBM to get a new realization
    result.i <- popRocIBM(clutch.data = clutch.data,
                          starting.pop = starting.pop,
                          plot.traj = plot.traj)

    # create a 1 row dataframe holding the result
    data.i <- data.frame("Trial" = i, "Rocs" = tail(result.i, n = 1))

    # add the result to the results dataframe
    results <- rbind(results, data.i)

  } # end of for loop

  return(results)

} # end of function
```

Don't forget you can always add a `browser()` call to the top of the function, and use R Studio's debug interface to step through your function. Honestly, there is no better way to see what is going on! I had to use `browser()` to fix a bug in my first attempt. And then again in my second attempt . . .

Let's test our function as it stands so far:

```
my.little.sim <- simIBM(trials = 10, clutch.data = data, starting.pop = 2, plot.traj = FALSE)

# have a look
my.little.sim
```

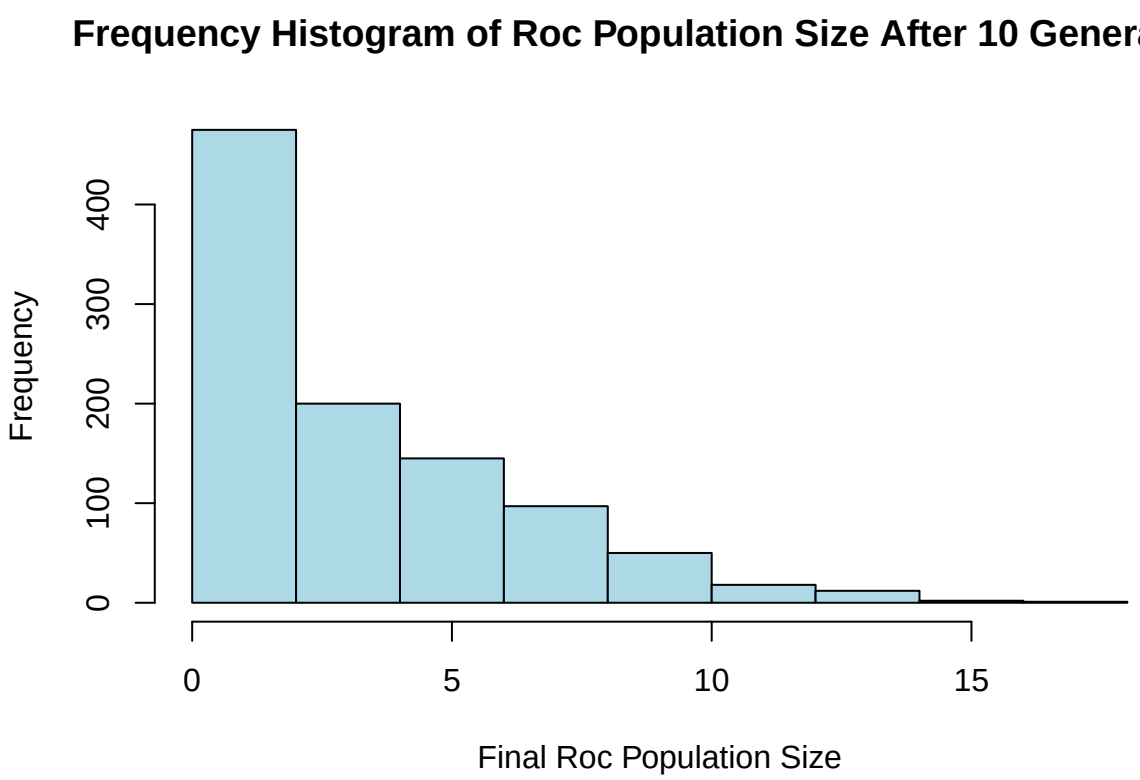
```
##    Trial Rocs
## 1      1    8
## 2      2    0
## 3      3    1
## 4      4    3
## 5      5    4
## 6      6    1
## 7      7    4
## 8      8    3
## 9      9    0
## 10    10    6
```

Our function appears to be working. At this point, we can run many trials, and summarize our results in a way that is meaningful.

```r
# run the simulation with 1000 trials
my.big.sim <- simIBM(trials = 1000, clutch.data = data, starting.pop = 2, plot.traj = FALSE)

# produce some summary statistics, like quantiles (I love these)
quantile(my.big.sim$Rocs)
```

```
##   0%  25%  50%  75% 100%
##    0    1    3    5   18
```

```r
# how about a histogram?
hist(my.big.sim$Rocs,
     xlab = "Final Roc Population Size",
     ylab = "Frequency",
     main = "Frequency Histogram of Roc Population Size After 10 Generations",
     col = "lightblue",
     border = "black")
```




## Reflection

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

- What information should you present to the new-age Caliph?
- Why is it important to run multiple trials?
- If Rocs could live for more than 1 year, how would you handle the survival part of the equation?
- How do you get from your model world back to the real world?