Intro to R modeling - Session 2

Daniel Viana

February 1, 2018

Table of Contents

# Vectorizing

Vectorizing is when we use vector operations to perform simulations, instead of populating a matrix. This makes the simulation much faster, which is important when you are running a sensitivity analysis over a complicated model. For this we use vector operations from session 1. Lets review some important vector operations and vectorize our simple population growth model over many growth rates:

#Create a vector of a sequence from 1 to 5  
x=1:5  
x

## [1] 1 2 3 4 5

#Create a vector of a sequence from 6 to 10  
y=6:10  
y

## [1] 6 7 8 9 10

#Multiply two vectors  
  
z = x\*y  
z #This will do the following: 1\*6, 2\*7....

## [1] 6 14 24 36 50

#Divide two vectors  
z = x/y  
z #This will do the following: 1/6, 2/7....

## [1] 0.1666667 0.2857143 0.3750000 0.4444444 0.5000000

#Sum two vectors  
z= x + y  
z #This will do the following: 1+6, 2+7....

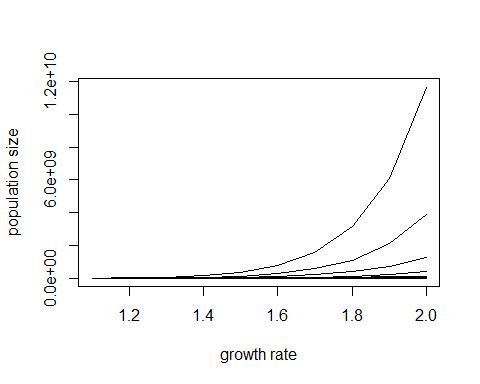
## [1] 7 9 11 13 15

#Another important vector operation is to multiply a vector by one value. This operation multiplies all elements of the vector by one value.  
z= x\*2  
z #This will do the following: 1\*2, 2\*2, 3\*2....

## [1] 2 4 6 8 10

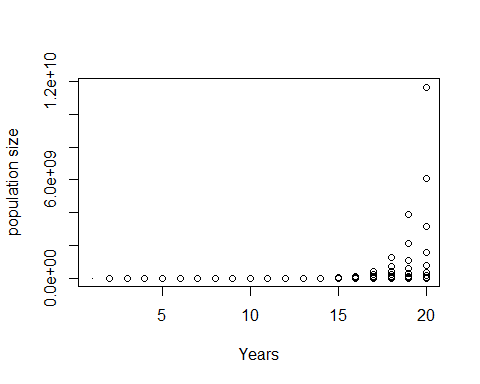
Now instead of using a nested loop, lets vectorize our example from Session 1

#First we create a vector to store the parameter values (in this case the growth rate)  
growth = seq(1.1, 2, by=0.1)  
  
#Let simulate over 20 years   
t=20  
  
#Then create a vector to store our populations  
pop = rep(0, length(growth))  
  
#Set initial population size (10)  
pop[]=10  
  
#Create a plot to demonstrate our population szies for each growth rate. We first need to plot our initial population sizes by each growth rate. Note that we need to know up front what will be the maximum population size for the years we simulate. This is easier when you set a carrying capacity for our population, which we can then set the ylim to K.  
plot(pop~growth, type="l", ylab="population size", xlab="growth rate", ylim=c(0, 11700000000))  
  
#Loop over the number of years we wish to simulate (t)  
for(n in 2:t){  
   
 #Lets first calculate how much our population will grow every time step   
 surplus = pop\*growth  
   
 #Now lets update our population every time step. This command will overwrite our vector of population sizes every time step. We will end up with a vector of population size at time t  
 pop = pop + surplus  
   
 #lets add a line to our plot of pop size at every time step to demonstrate the pop size vectors. The actual values should be the same as the matrix we constructed in session 1. In this plot, the first line will be our initial pop size (10) for all growth rate values, the second line will be our population at t=2, and so on.  
 lines(pop~growth)  
}

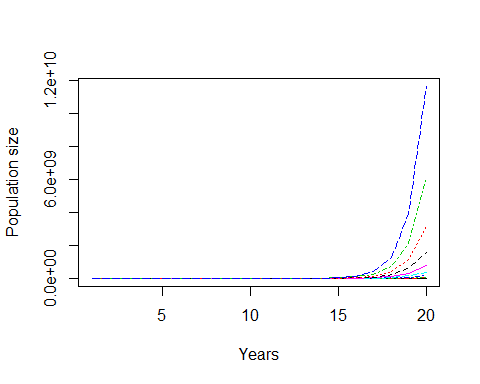


Note that we cannot graph population size over time as we did in session 1 because we only have a vector of the resulting pop size in year t. To do that we would have to create a matrix using the rbind function or construct a plot inside the loop. For example:

#First we create a vector to store the parameter values (in this case the growth rate)  
growth = seq(1.1, 2, by=0.1)  
  
#Let simulate over 20 years   
t=20  
  
#Then create a vector to store our populations  
pop = rep(0, length(growth))  
  
#Set initial population size (10)  
pop[]=10  
  
#Create a vector to store our population sizes  
out = pop  
  
#Create a plot for our population sizes in every time setp. Note that we need to create a vector with the same length of pop  
plot(rep(1, length(pop)), pop, type="l", ylab="population size", xlab="Years", ylim=c(0, 11700000000), xlim=c(1, t))  
  
#Loop over the number of years (rows) and growth rates (columns)  
for(n in 2:t){  
   
 #Lets first calculate how much our population will grow every time step   
 surplus = pop\*growth  
   
 #Now lets update our population every time step. This command will overwrite our vector of population sizes every time step. We will end up with a vector of population size at t=20  
 pop = pop + surplus  
   
 #Now we can add points to our plot for every time step  
 points(rep(n, length(pop)), pop)  
   
 #Alternatively, we can construct a matrix to store our population sizes from our vector out. This will add a line to out every time step of our loop.  
 out = rbind(out, pop)  
}



matplot(out, type="l", ylab="Population size", xlab="Years")



# Writing your own functions

Functions are critical to modeling in R. By turning your model code (like the population model we did in session 1) into a function, it allows you to easily modify and test parameters, perform optimization and make your programming more efficient and clean. Like every other function in R, each function you create is an independent little program, performing a few related tasks and returning the results. This makes complex problems easier to program, and makes it easier to see the logical flow of a large program.

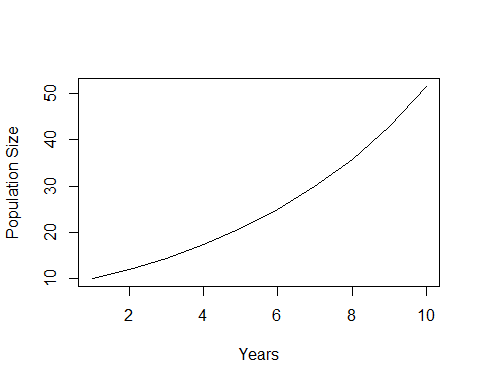
Suppose you want a function mypop to simulate a population growth model. You first need to specify the name of the function you wish to create (in this case mypop), and then use the function() command to create your function. Inside the parenthesis you will put the parameters of the model. In cases where you want a specific result (or results) from your function, you need the return() function in the end of your scrip to create mypop. The basic syntax is then:

mypop = function(parameters){  
commands  
return()  
}

The parameters specified in the function can be values or more complex objects, such as vectors or matrices. Similarly, we can set our function to return any one or combination of values or objects.

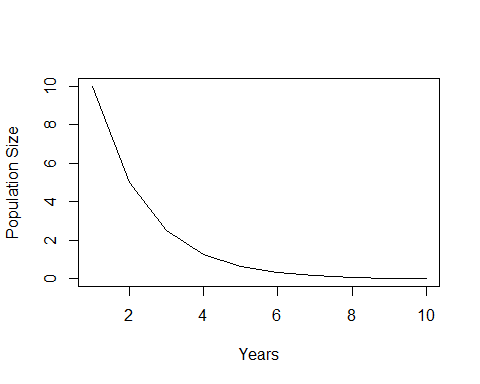
Lets go through an example. Lets simulate the same population growth model from Session 1, but now within a function:

#First we need to create our function, spcifying the parameters of our model. In this case our model has three parameters: g represents the population growth, N0 represents the initial population size, and t represents the number of years we wish to simulate  
  
mypop = function(g, t, N0){  
   
#Create a vector (pop) of zeros and size t   
pop = rep(0, t)  
  
#Set the initial population size  
pop[1]=N0  
  
#calculate population size at times 2 through 10, increasing by g every year  
for(n in 2:length(pop)){  
 pop[n]=g\*pop[n-1]  
}  
  
#Set the the returning object. In this case is going to be a vector of pop  
return(pop)  
}  
  
#To use this function you now need to specify the parameter values and name your results  
res = mypop(g=1.2, t=10, N0=10)  
  
#plot pop - note that is the same plot we creates in session 1  
plot(res, type="l", ylab="Population Size", xlab="Years")



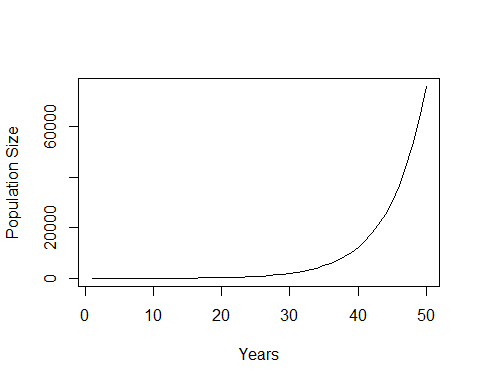
Now that we created our function, it is really easy for us to test different parameter values For example, now we want to test a different growth rate:

#Set g=0.5  
res = mypop(g=0.5, t=10, N0=10)  
  
#plot pop - note that now our population is decreasing  
plot(res, type="l", ylab="Population Size", xlab="Years")



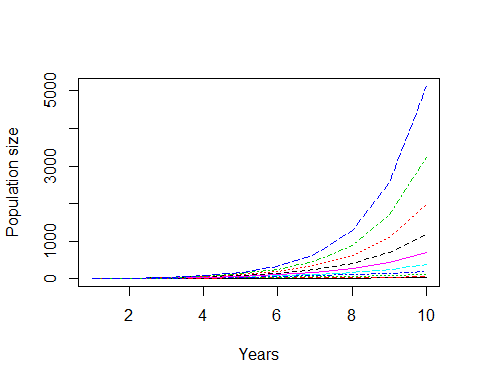
We can also change the number of years we wish to simulate (t=50):

#Set t=50  
res = mypop(g=1.2, t=50, N0=10)  
  
#plot pop - note that now our population is decreasing  
plot(res, type="l", ylab="Population Size", xlab="Years")



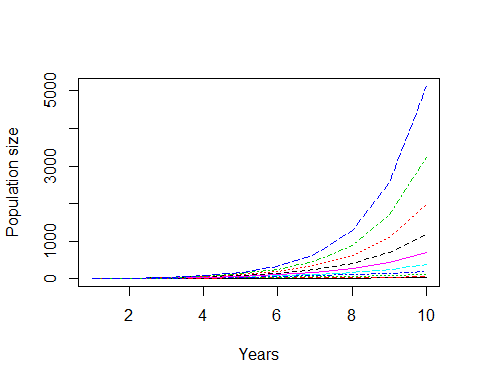
We can now easily create a matrix of our population sizes for different parameter values

#First, we need to create a vector of the growth rates we wish to simulate  
growth = seq(1.1, 2, by=0.1)  
t=10  
#Create a mtrix to store the values  
g.pop = matrix(0, t, length(growth))  
  
#now we run a for loop through all our population growth rates  
for(i in 1:length(growth)){  
 res = mypop(g = growth[i], t=10, N0=10)  
 g.pop[,i]=res  
}  
#In this loop, we first create a variable res to store our population for all growth rates. When i=1 res will be the vector of population size with a groth rate of growth[1] (which in this case is 1.1). Then we will store that vector in gpop. For i=2, res will be the vector of population size with a groth rate of growth[2] (which in this case is 1.2). Then we will bind this new vector with the previous one to form a matrix. This opreation will repeat for all values of growth. The result will be a matrix called g.plot with t rows and g columns.  
matplot(g.pop, type="l", ylab="Population size", xlab="Years")



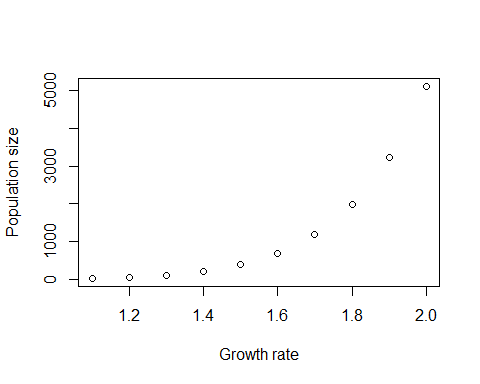
Another way of doing the same graph is to use the cbind function.

#First, we need to create a vector of the growth rates we wish to simulate  
growth = seq(1.1, 2, by=0.1)  
  
#now we run a for loop through all our population growth rates  
for(i in 1:length(growth)){  
 res = mypop(g = growth[i], t=10, N0=10)  
 if(i==1){g.pop = res}else(g.pop=cbind(g.pop, res))   
}  
#In this loop, we first create a variable res to store our population for all growth rates. When i=1 res will be the vector of population size with a groth rate of growth[1] (which in this case is 1.1). Then we will store that vector in gpop. For i=2, res will be the vector of population size with a groth rate of growth[2] (which in this case is 1.2). Then we will bind this new vector with the previous one to form a matrix. This opreation will repeat for all values of growth. The result will be a matrix called g.plot with t rows and g columns.  
matplot(g.pop, type="l", ylab="Population size", xlab="Years")



However, using the cbind function might be hard to keep track of the actual parameter in each column. To help you keep track it might be useful to use the rbind function instead, where you have a column to insert the actual parameter values. To test this, lets modify our function to return only the population size in the last year (t) and the parameters we are using.

#First lets redifine our function to return pop[50], g and t  
mypop.last = function(g, t, N0){  
   
#Create a vector (pop) of zeros and size t   
pop = rep(0, t)  
  
#Set the initial population size  
pop[1]=N0  
  
#calculate population size at times 2 through 10, increasing by g every year  
for(n in 2:length(pop)){  
 pop[n]=g\*pop[n-1]  
}  
  
#Set the the returning object. In this case is going to be a vector of pop[t], g and t  
return(c(pop[t], g, t, N0))  
}  
  
#Now lets test how growth rate will affect our final population size  
#First, we need to create a vector of the growth rates we wish to simulate  
growth = seq(1.1, 2, by=0.1)  
  
#now we run a for loop through all our population growth rates  
for(i in 1:length(growth)){  
 res = mypop.last(g = growth[i], t=10, N0=10)  
 if(i==1){g.pop2 = res}else(g.pop2=rbind(g.pop2, res))   
}  
#name the columns  
colnames(g.pop2)=c("pop", "g", "t", "N0")  
#In this case, in the first time step we are creating a vector of pop[50], g and t for growth[1]. In the second time step (i=2), we will create a matrix where the second row will be the values for growth[2]. The loop will keep adding rows to this matrix for all growth values. This can be a useful method when you are running a sensitivity analysis on a number of parameters.   
  
plot(g.pop2[,"pop"]~g.pop2[,"g"], ylab="Population size", xlab="Growth rate")



# Autoloading functions at startup

If you’ve written some functions that you would like to have available every time you use R , there’s a mechanism for having them load automatically at startup. All you need to do is write a script file with your function (or many functions) and save it in the folder Rstudio is connected. Then just add this function to the begging of your script. For example, if you saved your script in the “c:/src/R” location, and named your function “myfunction” you just use the command:

source(“c:/src/R/myfunction.R”)

Autoloading your functions can be useful when you want to run several functions in one script. This will make your script shorter and cleaner.

# Exercise 3

Write a function for the MPA model we constructed in exercise 2. For this model, our parameters will be , , , , , and . The parameter will be a value representing the percent in MPA of our model (from 0 to 100). Your should transform this MPA value into a vector of fishing mortalities, where patches designated as MPA have a value of . Set ; ; , , , and initial population of 50 individuals

**a**) Calculate yield and population sizes for all MPA values (0, 1, 2…, 100). Test MPA effectiveness for fishing mortality values for patches open to fishing of and .

Plot:  
1) Total population size (all patches) in year t for all MPA sizes and both fishing mortalities  
2) Total yield (all patches) in year t for all MPA sizes and both fishing mortalities

Answer:  
1) Does MPA improve system-wide fishing yield? 2) What is the optimal MPA size for both fishing mortalities?  
2) How does the answer from 2) change if ?

**b**) Calculate the optimal MPA size for different mobility values. Set for all patches open to fisheries and test a range of values (0, 0.1, 0.2…., 1) and MPA sizes (0, 1, 2…, 100). Optimal MPA size is defined as the percent in MPA that gives the maximum yield in year t.

Plot:  
1) optimal MPA size for different values of d

**c**) Now lets test for a different MPA location. Instead of the MPA be located on the edge of the string of patches (in the case of exercise 2 the patches on the right), lets set the MPA in the middle. Make the same plot of **b**.