Intro to modeling in R - Session 1

Daniel Viana

January 25, 2018

Table of Contents

# Vectors

Vectors are 1 dimensional arrays of numbers.

Here are examples of ways in which vectors can be created

#Create a vector of a sequence from 1 to 5  
x=1:5  
x=c(1,2,3,4,5)  
x=c(1:3, 4, 5)  
x=seq(1, 5, by=1)  
x

## [1] 1 2 3 4 5

#Create a vector of zeros with length 5  
x=rep(0, 5)  
x

## [1] 0 0 0 0 0

#Create an empty vector with length 5  
x=vector(length=5)  
x

## [1] FALSE FALSE FALSE FALSE FALSE

## Vector adressing

Often it is necessary to extract a specific entry or other part of a vector. This is done using brackets, for example:

#Create a vector of a sequence from 1 to 5  
x=1:5  
  
#Extract first element of the vector  
x[1]

## [1] 1

#Extract fifth element of the vector  
x[5]

## [1] 5

#Extract elements 1 to 3 of the vector  
x[1:3]

## [1] 1 2 3

#Extract elements 1, 3 and 5 of the vector  
x[c(1,3,5)]

## [1] 1 3 5

#We can also set specific values within a vector. For example, we can set the value for the third element of vector x as 0  
x[3] = 0  
x

## [1] 1 2 0 4 5

## Vector operations

We can perform operations with vectors such as divide, sum, multiply. When operating two vectors of the same size, it will apply the operation to elements in the same position.

Here are some examples

#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

#When vectors have different sizes, r will repeat values of the smallest vector and the resulting vector will have the same size as the largest vector. For example:  
x=1:5  
w=6:12  
k = x + w

## Warning in x + w: longer object length is not a multiple of shorter object  
## length

k

## [1] 7 9 11 13 15 12 14

#Notice that it worked well for 1:5. However, for elements 6 and 7 it summed with the first two elements of x (k[5:7] = c(11+1, 12+2)). Even though r gives you a warning, it does the calculations anyway.

# Matrices

Matrices are 2 dimensional arrays of numbers. All columns in a matrix must have the same mode (numeric, character, etc.) and the same length

Here is an example

#Create a matrix with 2 rows and 3 columns  
X=matrix(c(1,2,3,4,5,6),2,3)  
#This takes the values 1 to 6 and reshapes them into a 2 by 3 matrix  
X

## [,1] [,2] [,3]  
## [1,] 1 3 5  
## [2,] 2 4 6

#Note that values in the data vector are put into the matrix column-wise, by default. You can change this by using the optional parameter byrow. For example  
X=matrix(c(1,2,3,4,5,6),2,3, byrow=TRUE)  
X

## [,1] [,2] [,3]  
## [1,] 1 2 3  
## [2,] 4 5 6

#You can also create a matrix of zeros with 2 rows and 3 columns  
X=matrix(0,nrow = 2, ncol= 3)  
X

## [,1] [,2] [,3]  
## [1,] 0 0 0  
## [2,] 0 0 0

## Matrix adressing

Often it is necessary to extract a specific entry or other part of a matrix. Like in vectors, this is done using brackets. However, this time we need to specify two elements, where the first element represents the rows and the second represents the columns, for example:

#Create a matrix with 2 rows and 3 columns  
X=matrix(c(1,2,3,4,5,6),2,3)  
X

## [,1] [,2] [,3]  
## [1,] 1 3 5  
## [2,] 2 4 6

#First column of matrix  
X[,1]

## [1] 1 2

#First row of matrix  
X[1,]

## [1] 1 3 5

#Second element of the first column  
X[2,1]

## [1] 2

#Change value of the second element of the first column   
X[2,1]=200  
X

## [,1] [,2] [,3]  
## [1,] 1 3 5  
## [2,] 200 4 6

#We can name the columns of the matrix (note that we need to use quotation marks)  
colnames(X) = c("a", "b", "c")  
  
#Now we can also address the matrix by the name to extract the second element of column "a"  
X[2, "a"]

## a   
## 200

## cbind and rbind

Vectors of same size can be combined to form matrices. We can use the cbind function to combine by columns and the rbind function to combine by rows, for example:

#Create a vector of zeros and length 5  
x=rep(0, 5)  
  
#Create a vector of ones and length 5  
y=rep(1, 5)  
  
#Combine by rows  
rbind(x,y)

## [,1] [,2] [,3] [,4] [,5]  
## x 0 0 0 0 0  
## y 1 1 1 1 1

#Combine by columns  
cbind(x,y)

## x y  
## [1,] 0 1  
## [2,] 0 1  
## [3,] 0 1  
## [4,] 0 1  
## [5,] 0 1

# Arrays

Arrays are similar to matrices but can have more than two dimensions. This can be useful when we want two matrices to be connected somehow.

#Create an array of zeros consisting of two matrices with 10 rows and 3 columns  
x = array(0, dim=c(10, 3, 2))  
  
#we can name the columns (like in matrices)  
colnames(x) = c("a", "b", "c")  
x

## , , 1  
##   
## a b c  
## [1,] 0 0 0  
## [2,] 0 0 0  
## [3,] 0 0 0  
## [4,] 0 0 0  
## [5,] 0 0 0  
## [6,] 0 0 0  
## [7,] 0 0 0  
## [8,] 0 0 0  
## [9,] 0 0 0  
## [10,] 0 0 0  
##   
## , , 2  
##   
## a b c  
## [1,] 0 0 0  
## [2,] 0 0 0  
## [3,] 0 0 0  
## [4,] 0 0 0  
## [5,] 0 0 0  
## [6,] 0 0 0  
## [7,] 0 0 0  
## [8,] 0 0 0  
## [9,] 0 0 0  
## [10,] 0 0 0

#Once we name the columns, we can address each column using the column name  
#For example, we can set the first element of column "a" of the first matrix to one   
x[1,"a",1] = 1  
#Now we can extract the first row of the first matrix  
x[1,,1]

## a b c   
## 1 0 0

# Data frames

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.). One example where you would need a data frame is when you categorize your data somehow.

For example:

#Create a data frame with one numeric and one factor columns  
x = data.frame(1:3,c("a", "b", "c"))  
#Name the data frame  
names(x) = c("ID","categories")  
x

## ID categories  
## 1 1 a  
## 2 2 b  
## 3 3 c

#We can address data frames using a dollar sign ($)  
x$ID

## [1] 1 2 3

# Lists

A list allows to create a variety of unrelated objects under one name. To address elements in the list we need to use either two brackets or address by the name of each element. For example:

#Create a list with factors, a vector and a matrix  
x <- list(name=c("Fred", "Joe"), vector=1:5, matrix=matrix(1,2,3))  
x

## $name  
## [1] "Fred" "Joe"   
##   
## $vector  
## [1] 1 2 3 4 5  
##   
## $matrix  
## [,1] [,2] [,3]  
## [1,] 1 1 1  
## [2,] 1 1 1

#Extract first element  
x[[1]]

## [1] "Fred" "Joe"

#Extract element "name"  
x["name"]

## $name  
## [1] "Fred" "Joe"

#We can also use a dollar sign ($) to address lists, in this case to extract the element "name"  
x$name

## [1] "Fred" "Joe"

# Iteration (“loops”)

## For-loops

Loops make it easy to do the same operation over and over again.  
- Solving a model to make population forecasts  
- Simulating a model multiple times with different parameter values

A for loop runs for a specified number of steps:

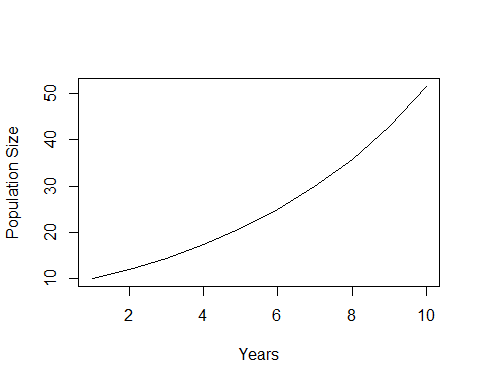
for (**variable** in **vector**) {  
commands  
}

What a loop does is go through each element of the vector you specify. So if you set a vector c(1,2,3), the loop will first use first the first element of the vector (in this case 1), then the second (2) until the last element (3). Here is an example where we use a loop to simulate a population over time:

#First we need to create a vector for the number of times (or years) you wish to simulate, which in this case is t=10  
  
#Define t  
t=10  
  
#Create a vector (pop) of zeros and size t   
pop = rep(0, t)  
  
#Set the initial population size  
pop[1]=10  
  
#calculate population size at times 2 through 10, increasing by 20% every year  
for(n in 2:length(pop)){  
 pop[n]=1.2\*pop[n-1]  
 print(n)  
}

## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7  
## [1] 8  
## [1] 9  
## [1] 10

#The first time through the loop, n=2. The second time through, n=3. When it reaches n=10, the for-loop is finished and R starts executing any commands that occur after the end of the loop. The result is a table of the population size in generations 2 through 10.  
  
#plot pop  
plot(pop, type="l", ylab="Population Size", xlab="Years")



## Nested loops

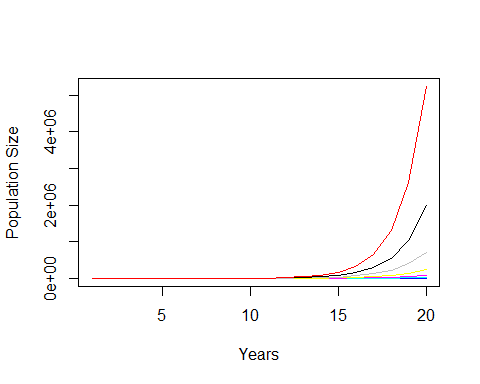
Several for loops can be nested within each other. This can be used in more complicated loops over several vectors and matrices.

Here is an example where we simulate a population over time for different parameter values:

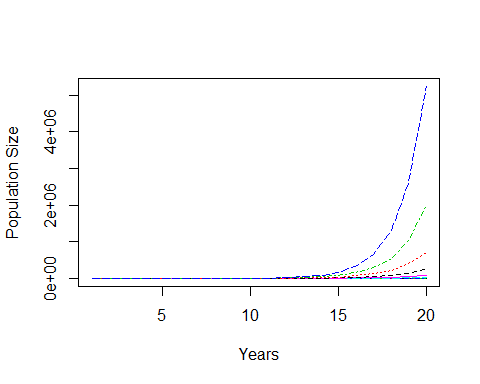
#First we create a vector to store the parameter values (in this case the growth rate)  
growth = seq(1.1, 2, by=0.1)  
  
#Then we create a matrix for the number of years you wish to simulate (in this case t=20) and number of parameters you wish to test, where the rows represent the number of years and the columns the number of parameters.  
#First set the number of years  
t=20  
#Then create the matrix to store our population  
pop = matrix(0, t, length(growth))  
  
#Set initial population size (10) for the first element of all the columns in our matrix  
pop[1,]=10  
pop

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## [1,] 10 10 10 10 10 10 10 10 10 10  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [3,] 0 0 0 0 0 0 0 0 0 0  
## [4,] 0 0 0 0 0 0 0 0 0 0  
## [5,] 0 0 0 0 0 0 0 0 0 0  
## [6,] 0 0 0 0 0 0 0 0 0 0  
## [7,] 0 0 0 0 0 0 0 0 0 0  
## [8,] 0 0 0 0 0 0 0 0 0 0  
## [9,] 0 0 0 0 0 0 0 0 0 0  
## [10,] 0 0 0 0 0 0 0 0 0 0  
## [11,] 0 0 0 0 0 0 0 0 0 0  
## [12,] 0 0 0 0 0 0 0 0 0 0  
## [13,] 0 0 0 0 0 0 0 0 0 0  
## [14,] 0 0 0 0 0 0 0 0 0 0  
## [15,] 0 0 0 0 0 0 0 0 0 0  
## [16,] 0 0 0 0 0 0 0 0 0 0  
## [17,] 0 0 0 0 0 0 0 0 0 0  
## [18,] 0 0 0 0 0 0 0 0 0 0  
## [19,] 0 0 0 0 0 0 0 0 0 0  
## [20,] 0 0 0 0 0 0 0 0 0 0

#Loop over the number of years (rows) and growth rates (columns)  
for(n in 2:nrow(pop)){  
 for(j in 1:ncol(pop)){  
 pop[n, j] = pop[n-1, j]\*growth[j]  
 }  
}  
#The loop starts over the rows of pop, and immediately a loop over the columns is started. To fill in the matrix we need to consider all possible values for the pair (row, col). So for row=1, we need to consider all columns (size ncol(pop) = 10) . Then for row=2 we also need to consider all columns, and the same for row=3. That's what the nested for-loops accomplish. In this case the order in which you perform the loops does not matter becasue the columns are unrelated to each other. In some cases when all columns are conected (such as an age structured model, populations conected through dispersal) the order does matter. For example, if these populations where conected through dispersal you would not want to run all the elements in the first column, then all the elements in the second column, and so on. In this case you need to run the first row though all columns, then the second row through all columns, and so on (like in the example above).   
  
#We can also use loops to plot population size vs years for all parameter values  
plot(pop[,1], type="l", ylim=c(10, max(pop)), ylab="Population Size", xlab="Years")  
for(j in 2:length(growth)){  
 lines(pop[,j], col=j)  
}



#Another way to plot multiple curves on one graph is with matplot. It works like plot but matplot(x) plots each column of a matrix as a separate curve. For example:  
matplot(pop, type="l", ylab="Population Size", xlab="Years")



## While-loops

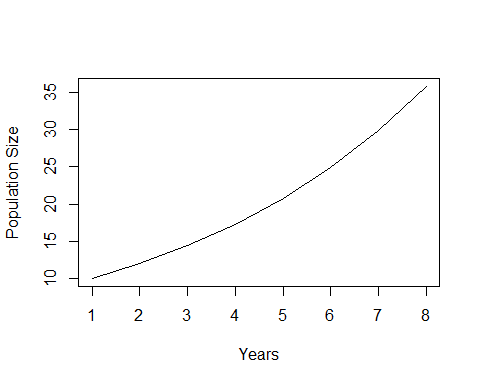
A while loop lets an iteration continue until some condition is satisfied. The format is:

while(condition){  
commands  
}

The loop repeats as long as the condition remains true. Since we don’t know in advance how many iterations will occur, we can’t create in advance a vector to hold the results. Instead, a vector of results is built up one step at a time. This can be useful when your model has some kind of a threshold value.

Here is the same example simulating a population growth model through time using while loops:

#Set initial population size and the first n value  
pop=10  
n=1  
  
#Run loops to increase population size by 20% every year until pop reaches a population size of 30  
while(pop[n]<30){  
 pop[n+1]=1.2\*pop[n]  
 n=n+1  
}  
#Note that for while loops you need an extra line of code to make sure that n values are increasing after each loop (n=n+1)  
  
#plot pop  
plot(pop, type="l", ylab="Population Size", xlab="Years")



#Note that simulations stoped right after satisfying the set condition. In this case, you end up with a population size above 30 because the loop stops after it reaches the set threshold. In other words, the loop condition is checked before the commands in the loop are executed. In generation 7 the population size is 29, so the condition is satisfied and the loop runs through one more time. After that the population size is 35, so the condition is not satisfied, so the lopp stops.

# Branching - “if” statements

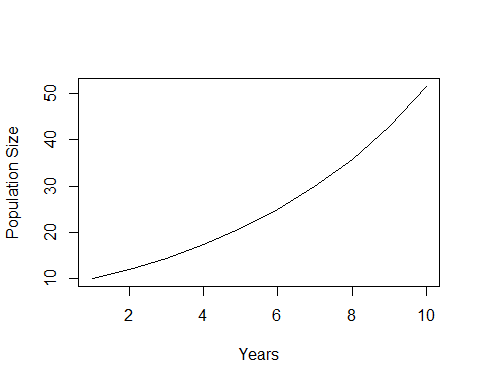
Allows the “rules” for state variable dynamics depend on the current values of state variables. In other words, if statements allows for commands to be executed only if a certain condition is met.

The basic format is:

if(condition) {some commands}else{some other commands}

Here is an example where we set the initial population size using the “if” function:

#First we create a vector to store our population  
pop= rep(0, 10)  
  
#Use an "if" statement to include the initial population size  
for(n in 1:length(pop)){  
 if(n==1){pop[n]=10}else{pop[n]=1.2\*pop[n-1]}  
}  
#The "if" function will apply pop=10 when the value of n is 1 and the actual equation to the remaining values of n. Note that we need two equal signs (==) to find identical numbers.   
  
#Now we can plot our population over time  
plot(pop, type="l", ylab="Population Size", xlab="Years")

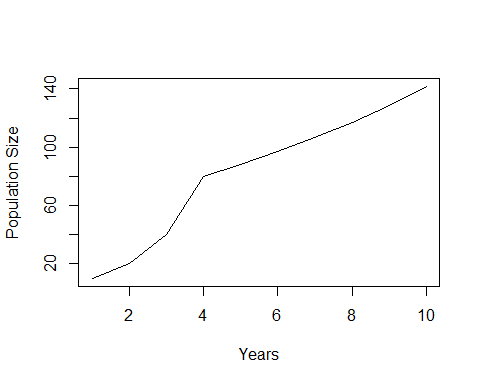


## Nested “if” statements

More complicated decisions can be built up by nesting one “if” block within another.

here is an example where we use the “if” function to set the initial population size and set different growth rates to simulate population size over time:

#First we create a vector to store our population  
pop= rep(0, 10)  
  
#Now we use an "if" statement to include the initial population size and change population growth after year 5  
for(n in 1:length(pop)){  
 if(n==1){pop[n]=10}else{  
 if(n<5){pop[n]=2\*pop[n-1]}else{  
 pop[n]=1.1\*pop[n-1]}  
 }  
}  
#Here the if function will apply pop=10 when the value of n is 1, and the actual equation to the remaining values of n. The difference now is that we inserted another if function to tell our model to use a growth rate of 2 for values of n lower than 5 (excluding 1) and a gorwth rate of 1.1 for all other values.  
  
#Now we can plot our population over time  
plot(pop, type="l", ylab="Population Size", xlab="Years")



# Exercises

## Exercise 1

Write a script file that simulates a fish population growth model over time. We will use a simple logistic model where the number of fish this year are the number of fish last year, plus the population growth minus the fish that died from fishing mortality. The growth of the population will be density dependent, meaning that the growth rate will be affected by population size, where smaller populations grow faster that large populations. In the absence of fishing, the population will grow until it reaches a certain carrying capacity. Let denote the number of individuals at time ( = 1, 2, 3, . . ., ), let be the species growth rate, be the carrying capacity, and be the fishing mortality, according to the following equation:

**a**) Write your program to start with 50 individual at time , iterate the model up to =50. Write your program so that , , and are parameters, in the sense that the first line of your script file reads , and .

Plot:  
1) the population size over time  
2) the yield over time

Answer:  
1) What is the total yield in 50 years (sum of all years)?  
2) What is the yield and population size in year 50?

**b**) Now test different values of fishing mortality (), ranging from 0 to 1 (0, 0.1, 0.2…., 1). Note that should now be a vector. You should end up with a matrix where the columns are the different fishing mortality and the rows are the years you iterate your model.

Plot:  
1) the population size over time for each fishing mortality  
2) the yield over time for each fishing mortality  
3) Yield in year 50 for each fishing mortality  
4) Total yield for each fishing mortality

Answer:  
1) Which fishing mortality gives you the highest total yield?  
2) How does answer from 1) changes if you model only 3 years?  
3) Which fishing mortality gives you the highest equilibrium yield (t = 50)?  
4) What happens to answer 3) if you change the growth rate to 0.2?

## Exercise 2

Write a script file that simulates a population growth model distributed into interconnected patches. We will use the same base model from exercise 1. However, this time we will model different populations in a series of patches along a line (denoted by L). These population will be connected through dispersal, with individuals emigrating to other patches and immigrating from other patches. Let denote the number of individuals in patch ( = 1, 2,.., ) at time ( = 1, 2, 3, . . ., ), let be the species growth rate, be the carrying capacity, be the fishing mortality of patch , be the emigration to patch , and be the immigration from patch , according to the following equation:

Where:

Where is the dispersal rate. Also, lets assume reflecting boundaries: no fish disperse out of the system. That is, there is no leftward dispersal out of patch 1 and no rightward dispersal out of patch L. Therefore, the equations for boundary patches (1 and ) will be different.

Write your program to start with 50 individuals in each patch at time , iterate the model up to =50. Use the following fishing mortalities: in the left half of the patches, and in the right half of the patches. Write your program so that , , and are parameters, in the sense that the first line of your script file reads ; ; , .

**a**) Run your model using for loops over a matrix of population sizes (with columns and rows)

Graph:  
1) the total population size (total number in all patches) over time  
2) the total yield over time 3) average population size over time for patches with and   
4) average yield over time for patches with and   
5) Population size in year 50 for each patch  
6) Yield in year 50 for each patch  
7) Total yield (all years) in each patch  
8) Total population size (all years) in each patch

**b**) Now instead of looping over a matrix, Vectorize. To vectorize, L is a one-line statement (or vector) of the form . You will still use for loops to vectorize, but instead of using matrices you will use vectors. This will make your simulation run much faster. For t=1 you will have a vector (length L) with initial population sizes. Next time step you will overwrite your vector with the new population sizes expected in t=2. And you will repeat this operation for all time steps. You will end up with a vector of the population size in year 50. Repeat all the graphs from **a**.

**c**) Now lets introduce marine reserves into the system. First, lets use , to simulate 100 patches. Now each patch will have either (open to fishing) or (marine reserves). To simplify, lets make patches in the left to be open to fishing and patches in the right to be set as marine reserves. Lets start setting 50% of the patches as a marine reserve.

Graph:  
1) Population size in year 50 in each patch  
2) Yield in year 50 in each patch  
3) Total yield (all years) in each patch  
4) Total yield (all years and all patches) for reserves of 10%, 50% and 80%  
5) Total population size in year 50 (all patches) for reserves of 10%, 50% and 80%