R Programming Language

Notes for CS 6232: Data Analysis and Visualization
Georgia Tech (Dr. Guy Lebanon), Fall 2016
as recorded by Brent Wagenseller

Lesson Preview

R is good for data analysis as its fast to code in R, so you can get quick results

- That said, R itself is not as fast as, say, C or Java
- Ultimately, both R and a faster language (C or Java) are used in combination to take on big tasks

Goals

- Understand when to use R and when not to use it
- Understand R's basic syntax & write short programs
- Understand scalability issues and ways to resolve them

R, Python, and Matlab Similarities and Differences

The decision on which language to use should be based on the task at hand

Similarities

Characteristic	R	Python	Matlab
Run in interactive shell or graphical UI	x	x	x
Store and manipulate data as arrays	x	x	x
Many packages	x	x	x
Slower than C, C++	x	x	x
Interface with C++	x	x	x

Differences

Characteristic	R	Python	Matlab
Open source	x	x	
Ease of Contribution	x		
Quality of Contributions	x		
Suitable for Statistics	x		
Better Graphics Capabilities	x		

While all languages are slower than C/C++/Fortran, this can sometimes be overcome – specifically by interacting with native C++ code

Advantages of R

- R's standardized process of contributing packages has led to a large group of motivated contributors who contribute high quality packages
- R's syntax is more suitable for statistics and data as R was designed for this

- R has better graphics capabilities

Where are the languages popular?

- R is popular in statistics, bio-statistics, and social sciences
- Matlab is popular in Engineering and applied math
- Python is popular in web development and scripting

Running R

Two ways to Run R

- Interactively
  - Typing ‘R’ at the command line
  - R graphing application (Windows/MAC OXS)
  - RStudio
  - Within Emacs
- Non-interactively
  - This is mostly done with R scripts (foo.R)
  - Calling scripts within R
    R Command: source(“foo.R”)
  - From the terminal window / shell
    R CMD BATCH foo.R
    Rscript foo.R
  - Making an executable script
    MUST Include the following as the first line in the script
    #!/usr/bin/Rscript
    From there, just type this in the terminal: ./foo.R

More on RStudio

- GUI that makes using R interactively easy
- 4 panels
  - code
    top left
    tabbed
  - execution results / interactive R
    bottom left
  - graphs/help
    bottom right
  - history and workspace
    top right

General R Notes

Whitespaces are dropped

Semicolons aren’t required unless you want to put two commands in one line

Comments use a #

Variable types can be changed at runtime (so they are not statically typed)

R is a functional language, so functions must have at least a () at the end

Typically speaking, periods are simply part of the function/name and NOT an operator symbol

When passing to functions, R allows for the parameters to be out of order provided you explicitly say which variable in the function you are setting

- Otherwise – if you do not – they must be in the order in which they are defined

Helpful R commands

You can use help() to get help on any function

- For example, help("load") gives help on the load function

ls() lists variable names in workspace memory

save.image(file="R_workspace") saves all variables to the file

save.image(new.var, legal.var.name, file="R_workspace") saves specific variables to the file

load("R_workspace") loads all variables from the file

install.packages("ggplot2") installs packages (in this instance the ggplot2 package)

library(ggplot2) loads the package

To execute a command in the underlying shell you can use the system() command.

- For example, system("ls –la") executes the ls command in the shell

the c() function is read 'combine' or 'concatenate' and combines elements to make a vector

- For example, c(5,8,2) makes a vector / array with 3 elements: 5, 8, and 2

Scalars in R

numeric are the most common scalar type, and can represent floating point values

integer is less common in R

- Its possible to cast a numeric as an integer
    - If there exists a numberic 'c' it can be cast as an integer: c<- as.integer(b)

logical can be TRUE or FALSE, but it can be cast to a 1 or 0 with as.numeric

string is also possible, and either single or double quotes work

Factors

Factors are variables in R which take on a limited number of different values

Factors are similar to enums in C++ and Java

Factors can be ordered, but we have to define the ordering. Example

- current.season = factor("summer", levels=c("summer", "fall", "winter", "spring"), ordered=TRUE)
- The ordering is the default ordering in the concatenated vector of strings
- Ordering factors only make sense if data can be logically ordered
- Factors can also be unordered

Vectors Quiz 1 / 2

Fill in the blanks with the outcome of each 'R' command.

Purpose	Example	Outcome
concatenate	x = c(4,3,3,4,3,1)	x = 4 3 3 4 3 1
get length of vector or array	length(x)	length = 6
assign a boolean vector	y = vector(mode = "logical", length = 4)	y = FALSE FALSE FALSE FALSE
assign a numeric vector	z = vector(length = 3, mode = "numeric")	z = 0 0 0

Fill in the blanks with the outcome of each 'R' command.

Purpose	Example	Outcome
repeat value multiple times	q = rep(3.2, times = 10)	q = 3.2 3.2 3.2 3.2 3.2 3.2 3.2 3.2 3.2 3.2
load values in increments	w = seq(0, 1, by = 0.1)	w = 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
load values in equally spaced increments	w = seq(0, 1, length.out = 11)	w = 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

Using 'length.out' just means 'using the beginning and end inclusively, create 11 values that are equally spaced'

Creating sequences of values is important in creating grids that are useful for graphing data and functions

Comparison Commands Quiz

Fill in the boxes with the result of each example command.

Purpose	Example	Outcome
Boolean vector	w <= 0.5	w = TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
Checking for true elements	any(w <= 0.5)	TRUE
Checking for all true elements	all(w <= 0.5)	FALSE
Which elements are true	which(w <= 0.5)	1 2 3 4 5 6

```
w = 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Which seems to be very helpful as it returns the rows that match

Subset Commands Quiz

Purpose	Example	Outcome
Extracting entries	w[w <= 0.5]	0.0 0.1 0.2 0.3 0.4 0.5
Subset function	subset(w, w <= 0.5)	0.0 0.1 0.2 0.3 0.4 0.5
Zero out components	w[w <= 0.5] = 0	w = 0.0 0.0 0.0 0.0 0.0 0.0 0.6 0.7 0.8 0.9 1.0

```
w = 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

The brackets [] are used to limit the subset
Try to not use subset() and use the brackets instead
Its also possible to limit it AND set the matching elements at the same time

Creating Arrays Quiz

```
z = seq(1, 20,length.out = 20)
x = array(data = z, dim = c(4, 5))
```

Fill in the boxes with the values stored in the array.

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

The 'dim' simply means 'take the given vector and make it into a multidimensional array of 4 rows, 5 columns'

Reading Arrays Quiz

Given the following array, fill in the blanks with the results of each command.

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

```
x[2,3] = 10
x[2,] =  2  6 10 14 18
```

x[-1,] =

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

y = x[c(1,2), c(1,2)]

	[,1]	[,2]
[1,]	1	5
[2,]	2	6

Using a –X means ‘exclude that row / column’ (remember that the first number is the row and the second is the column)

- If left blank, it means include all

To include a specific list, use the c() operator

Manipulating Arrays Quiz

Given the following array, **determine the outcomes** of the following commands.

```
        [,1] [,2]
y = [,1]   1    5
    [,2]   2    6
```

`2 * y + 1`

	[,1]	[,2]
[1,]	3	11
[2,]	5	13

`y %*% y`

	[,1]	[,2]
[1,]	11	35
[2,]	14	46

Each operation is applied to each row separately

The %*% is a matrix multiplication operator – so basically matrix multiplication

Inner Product (Dot Product) and Transpose Quiz

Given the array 'x', **determine the outcome** of the following commands.

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

`x[1,] %*% x[1,]`

	[,1]
[1,]	565

`t(x)`

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

t() transposes the matrix

Outer Product Quiz

Given the array 'x', **determine the outcome** of the following commands.

```
outer(x[,1], x[,1])
```

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

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	2	3	4	
[2,]	2	4	6	8	
[3,]	3	6	9	12	
[4,]	4	8	12	16	
[5,]					

The outer product is a matrix whose (i,j) element is a product of the i^{th} component of the 1^{st} vector multiplied by the j^{th} component of the second

Concatenation Quiz

Determine the outcome of the following commands.

```
rbind(x[1,], x[1,])
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	5	9	13	17
[2,]	1	5	9	13	17

rbind means 'row bind' – its basically 'concatenate these things'

Determine the outcome of the following commands.

```
cbind(x[1,], x[1,])
```

	[,1]	[,2]
[1,]	1	1
[2,]	5	5
[3,]	9	9
[4,]	13	13
[5,]	17	17

cbind is similar, yet it makes columns instead

Lists Quiz

Given the following list command, fill in the blanks with the result of each command.

```
L=list(name = 'John', age = 55, no.children = 2, children.
ages = c(15, 18))
```

names(L)	name age no.children children.ages	L['name']	John
L[[2]]	55	L$children.ages[2]	18
L$name	John	L[[4]][2]	18

Double square bracket notation in lists returns the actual value – if we only used a single bracket it would return another list

Dataframes Quiz

Assume the following commands have been executed, fill in the blanks with the corresponding outputs

```
vecn = c("John Smith","Jane Doe")
veca = c(42, 45)
vecs = c(50000, 55000)
R = data.frame(name = vecn, age = veca, salary = vecs)
```

R

	name	age	salary
1	John Smith	42	50000
2	Jane Doe	45	55000

Dataframes are ordered lists sharing the same signature (data types)

Dataframes Modification Quiz

Given the following dataframe called 'R', fill in the blanks to reflect the changes made by the command:

```
names(R) = c("NAME", "AGE", "SALARY")
```

	name	age	salary
1	John Smith	42	50000
2	Jane Doe	45	55000

	NAME	AGE	SALARY
1	John Smith	42	50000
2	Jane Doe	45	55000

This simply changes the names of the columns

Datasets Quiz

Write the 'R' command that will perform the listed task

Task	Command
List the dimension (column) names	names(iris)
Show the first four rows	head(iris,4)
Show the first row	iris[1]
Sepal length of the first 10 samples	iris$Sepal.Length[1:10]
Allow replacing iris$Sepal.Length with shorter Sepal.Length	attach(iris, warn.conflicts = FALSE)

Use the Iris data set … data(iris)

Write the 'R' command that will perform the listed task

Task	Command
Average of Sepal.Length across all rows	mean(iris$Sepal.Length)
Means of all four numeric columns	colMeans(iris[,1:4])
Create a subset of sepal lengths less than 5 in the setosa species	subset(iris, Sepal.Length < 5 & Species == "setosa")
number of rows corresponding to setosa species	dim(subset(iris, Species == "setosa"))[1]
summary of the dataset iris	summary(iris)

Use the Iris data set … data(iris)

If Else

The if-else block is pretty standard. Example:

```
a = 10; b = 5; c = 1
if (a < b) {
      d = 1
} else if (a == b) {
      d = 2
} else {
      d = 3
}
print(d)
```

Note that logistical operators – And (&&), Or (||), equality (==), inequality(!=) are all the same

- BRENTS NOTE: The exception seems to be in brackets [] for dataframe subsets – there it seems to be only 1 & and one | for ‘and’ and ‘or’, respectively

For Loops

For loops are a bit different

```
#The format for a 'for' loop in R is:
#for(i in 1:100){
#    print("Hello World")
#    print(i*i)
#}

#Everything in the curly braces is executed 100 times.

#Write a 'for' loop that adds the numbers (num) 1 to 100 and
#stores it in a variable called 'sum'.

total = function(n){
    sum = 0

    ###Put the code for the 'for' loop here.
    for(i in 1:n) {
            sum = sum + i
    }

    print(sum)
    return(sum)
}

total(100)
```

Also we could have used a c() instead of 1:100 (but we would have had to list all the numbers)

- We also could have used
  for(i in seq(1,100,by = 1)) {

Note that the += trick does not work for the sum

Repeat Loops

It seems that a repeat loop in R is much like a for or while, except there is no

baked-in exit – you must put a ‘break’ in it
Could be used in place of a do…while loop

```
#A repeat loop must use a break statement to exit the loop.

#Using a repeat loop, write an 'R' program that
#subtracts the numbers (num) 100 to 1 from a variable called sum.
#If the sum becomes '0' or less, exit the repeat loop.
#Use a variable called 'num' for the numbers and 'sum' for the sum.

total = function(n){
    sum = 5050
    num = n

    ###Put the code for the 'repeat' loop here.
    repeat {
        sum = sum - num
        num = num - 1
        if (sum <= 0){
            break
        }
    }
    return(sum)
}

total(100)
```

WHILE Loops

While loops are very standard:

```
#Given two variables (a,b) and a sum=0, write a while loop to
#perform the following task:

#While b>a, increment the variables sum and 'a'
 and decrement the variable 'b'.
#a = 1, b = 10

total = function(){
    sum = 0
    a = 1
    b = 10

    #Put the while loop here
    while (b > a) {
        sum = sum + 1
        a = a + 1
        b = b - 1
    }

    return (sum)
    }

total()
```

Functions

Example

```
myPower = function(bas = 10, pow=2) {
        res = bas^pow
        return(res)
}
```

‘myPower’ is now the function name that can be called in other parts of the code

Note that the parameters have default values, so if a parameter is not supplied the default is used

Multiple ways to call the function

- myPower(2, 3)
- myPower (pow=3, bas=2)
  - If you are explicit with the variable, the order does not matter
- myPower(bas=3)

Functions Quiz

The given function is expecting variables to be in the order x,y,z

Fill in the blanks to call the function for each situation.

`Assume x=10, y=20, z=30`

Call foo with the variables in x,y,z order	foo(10,20,30)
Call foo with the variables in y,x,z order	foo(y=20, x=10, z= 30)
Call foo with the variables x and y set to default, z = 30	foo(z = 30)

Vectorized Code

R code can run slow, especially if it has many loops or iterations

- This is because R runs inside an interpreter
- A way around this is to write vectorized code

Vectorized code means we find another way to write loops

- For example, instead of writing

```
a=1:10000000; res = 0
for(e in a) res = res + e^2
```

- Instead, write

```
Sum(a^2)
```

External / Native API

Often, 10% of the code is responsible for 90% of the computing time; this is called the bottleneck

If we cannot vectorize the code, we can run the bottleneck in C/C++ instead and then call that from within R

```
dyn.load("fooC2.so") # load compiled C code

A = seq(0, 1, length = 10)
B = seq(0, 1, length = 10)
.Call("fooC2", A, B)
```

Newer packages: Rcpp, RcppArmadillo, RcppEigen

- The .Call knows to call the C code fooC2 with the given parameters
- fooC2 is already compiled code
- .Call is old, newer packages are Rcpp, RcppArmadillo and RcppEigen

This isn't a terrible tradeoff as the bottleneck is typically only a small amount of C code

Lesson Summary

We reviewed R

We noted that R can be slow, but we can get around it via vectorized code and by using external compiled code in C