Lecture 8

R programming debugging

GEOG 489 SPRING 2020

R Programming Structure

Recursion

A recursive function calls itself. This can be a very powerful solution to various problems.

The basic notion of a recursive function is:

For a problem you are trying to solve of type X:

1) Break the original problem of type X into one or more smaller problems of type X.
2) Within f(), call f() on each of the smaller problems.
3) Within f(), consolidate the results of step 2 to solve the original problem.

Quiz 3

Write a **recursive function** to calculate the factorial of a positive integer number. Factorial of a positive integer number is defined as the product of all the integers from 1 to that number.

Hint: For example, the factorial of 5 (denoted as 5!) will be 5! = 1*2*3*4*5 = 120. This problem of finding factorial of 5 can be broken down into a sub-problem of multiplying the factorial of 4 with 5 (namely 5! = 5*4!). More generally, n! = n*(n-1)!

```
recursive.factorial <- function(x) {

if (x == 0)    return (1)

else           return (x * recursive.factorial(x-1))

}
```

Debugging

You are already becoming experts on debugging code, but if you are still struggling, the main thing to remember is the concept of CONFIRMATION.

Don't assume something your code is going to work the way you want it to:

Test it!

Debugging

To begin with, always have small test cases that you have figured out the answer to ahead-of-time, e.g. confirm the results independently.

If you write your code in a modular fashion, your top level function should contain a very small number of commands, most of which are function calls. This will allow you to test the internal functions one at a time.

Debugging

```
findruns <- function(x,k)
{
    n <- length(x)
    runs <- NULL
    for(i in 1:(n-k))
    {
        if(all(x[i:(i+k-1)]==1)) runs <- c(runs,i)
    }
    return(runs)
}
# Test the function:
findruns(c(1,0,0,1,1,0,1,1,1),2)
```

Debugging

1. Function error

First thing to debug: parentheses, brackets, braces.

2. If the function had no obvious errors, but does it do what we want? CONFIRM CONFIRM CONFIRM.

This should find the position of all runs of 1 of length 2:

findruns(x=c(1,0,0,1,1,0,1,1,1),k=2)

We have confirmed something is wrong with a simple test case.

Debugging

Three basic ways of debugging:

1) Use print() statement to confirm each section. This gets messy, but is a quick and dirty way to debug code.

2) Single-stepping through your code is an important trick, and R provides nice functions to help you with this. We use debug() to debug code.

3) Use a "breakpoint“ to debug certain parts of the code. In R, we could use browser() to insert a breakpoint.

Debugging

Some commands for the Browser prompt:
n ("next") or ENTER will execute the next line and then pause.

c ("continue") basically execute until a close brace "}" is found. What this amounts to is if you enter a loop, the entire loop will execute before pausing again. If you are outside of a loop, the rest of the function will execute down to the brace.

Any R command: you can type any R command at the browser prompt to query/modify the execution. One exception: if you have a variable named after a browser command, wrap it in a print() statement, e.g. print(n) or print(c).

Q : quits the browser and exits back to interactive level.

Tools for composing function code

If we've written some code that have functions in it, and written it to a file "xyx.R". We can quickly load them into the environment via:

source("xyz.R")

Tools for composing function code

If you want to quickly edit a function form within the R console, you can use the fix() and edit() function:

```
f1 <- function(x) return(x+1)

fix(f1)

f2 <- edit(f1)
```

Math and Simulation

```
### Math Functions
# R has any basic math function can you think of:
?exp()          # Exponential function, base e
?log()          # Natural logarithm
?log10()        # Logarithm base 10
?sqrt()         # Square root
?abs()          # Absolute value
?sin()          # Trig functions, also includes cos, tan, asin, atan, and
atan2
?min()          # Minimum and maximum value of a vector
?max()
?which.min()    # Index of the min or max value of a vector
?which.max()
```

Math and Simulation

```
### Math Functions
# R has any basic math function can you think of:
?pmin()             # Element-wise min or max of several vectors
?pmax()
?sum()          # Sum of the elements of a vector
?prod()         # Product of the elements of a vector
?cumsum()           # Cumulative sum of the elements of a vector
?cumprod()          # Cumulative product of the elements of a vector
?round()        # Round to the closest integer
?floor()        # Round to the closest integer lower
?ceiling()      # Round to the closest integer higher
?factorial()    # Factorial function
```

Assignment 2

Your goal is to write a local-window smoothing function for use on matrices.

The inputs are as follows:

myMatrix: an arbitrarily large numeric matrix that is at least 3 x 3

smoothingMatrix: an arbitrary 3 x 3 numeric matrix

The output should be a matrix of the same size as myMatrix. For each location in myMatrix, the value should be equal to the average value of the 3x3 window around that location multiplied element-wise by the smoothingMatrix.

Assignment 2

For example:

```
myMatrix <- matrix(1:25,nrow=5)
myMatrix
smoothingMatrix <- matrix(0.5,nrow=3,ncol=3)
smoothingMatrix
# The value of the output matrix at position 2,3 will be
# based on the local window around that point:
localWindow23 <- myMatrix[1:3,2:4]
localWindow23
# Multiplying this local window by the example smoothing matrix and
#    determining the mean value results in a single value of 6.
```

Assignment 2

The requirements are as follows:

1) The function name should be "localSmoother".

2) smoothingMatrix, by default, should be a 3x3 matrix of all 1s.

3) The class of myMatrix should be checked to make sure it is a matrix of mode numeric. If it fails these checks it should stop with a warning.

4) The dimensions of smoothingMatrix should be checked to make sure it is a numeric 3x3 matrix. If it fails these checks it should stop with a warning.

5) If a local window would run off the matrix (e.g. is an edge cell), the output value at that location should be NA.

6) Comment your code in at least 3 places.

7) You may NOT use any R packages except the default set.

8) The code should be submitted on Compass 2g as a single function with the filename:

lastname-firstname-geog489-s20-assignment-02.R

and should have at the top:

[Your name]

Assignment #2

Assignment 2

1 point of extra credit if you add a third parameter to the function "na.rm" which should default to FALSE that, if TRUE, functions as follows:

Instead of returning NA for locations where the local window runs off the edge of myMatrix, it uses all the values it can (e.g. the values that are NOT off the edge) to calculate the mean.

Assignment 2

```
# Small hint:
#   If we have nested for() loops, we get the following behavior:

for(y in 1:3)
{
    for(x in 1:3)
    {
        print(paste("y,x=",y,",",x,sep=""))
    }
}
```

Assignment 2 is due next Tuesday, Feb. 25 2019 at midnight.