CSC 120 (R Section)— Lab Exercise 8

This is a non-credit exercise, which you do not hand in.
You may work on your own or together with another student, as you please.

In this lab, you will do exercises in passing a function as an argument to another function, and in using logical vectors as subscripts. You will also get more practice in debugging functions.

As was demonstrated in an earlier lecture, you can ask R to stop when it starts a function by using `debug(f)`, where `f` is the name of the function. When this function is called, R will stop, and display a message. You can then go through the function one step at a time, by typing "n" (or just Enter/Return) for each step. You can also examine variables or see the values of expressions computed from them by just typing the variable name or expression (but you will need to put the name in parentheses if it is the same as a debugger command, such as "n"). You can exit the debugger and the function you're running with "Q".

RStudio displays the position in the function you are debugging in a separate pane of the window. It also displays the variables in the current function.

You can set more than one function to be debugged. To undo debugging, use `undebug(f)`. Note that redefining a function (eg, by clicking the "Source" button for the script that defines it) will also have the effect of undoing debugging.

You should try using the debugger when doing the rest of this lab, when you need to find out why something isn't working. If you happen to get your functions to work first time, you can still try out the debugger just to see how it lets you step through a function.

To help get you started on passing functions as arguments, and using such an argument, here is a simple function that takes a function as an argument, and returns the sum of applying it to the arguments 1 and 2:

```
> apply_to_1_and_2 <- function (f) f(1) + f(2)
> apply_to_1_and_2(sqrt)
[1] 2.414214
```

Creating a matrix with elements computed by some function. For this exercise, you should write a function called `make_matrix` that takes a first argument called `element`, which is a function, and second and third arguments called `nrow` and `ncol`. The second and third arguments specify the number of rows and number of columns in the matrix. The first argument specifies what the elements of the matrix will be. It should be a function that takes as arguments the row and column indexes of an element in the matrix, and returns a single number that should be the value of that element.

Here are examples of how one might use `make_matrix`:

```
> elem <- function (i,j) (i+j)^2
> make_matrix (elem, 3, 4)
     [,1] [,2] [,3] [,4]
[1,]    4    9   16   25
[2,]    9   16   25   36
[3,]   16   25   36   49
```

```
> make_matrix (function (i,j) i-j, 4, 4)
     [,1] [,2] [,3] [,4]
[1,]    0   -1   -2   -3
[2,]    1    0   -1   -2
[3,]    2    1    0   -1
[4,]    3    2    1    0
```

Notice that you don't have to give a name to a functon in order to pass it as an argument.

Selecting vector elements with a logical subscript vector. Recall from the lecture slides that you can use a logical vector as a subscript for a vector — this returns a vector with only those elements where the subscript vector is `TRUE`.

Write a function called `away_from_mean` that takes as arguments a numeric vector, `vec`, and a single positive number, `d`, and returns a vector of all elements of `vec` that are at least `d` away (either above or below) from the mean (average) of the elements in `vec`.

Selecting rows of a data frame with a logical subscript vector. You can also use a logical vector to select rows of a data frame. Write R commands (which needn't be in a functon) to find the subset of rows of the built-in `iris` demonstration data set for which the ratio of `Petal.Length` to `Petal.Width` is at least four. Do you notice anything about the species of the plants that you selected?

Selecting rows or columns of a matrix with a logical vector. You can also use a logical subscript vector to select rows or columns of a matrix. Write a function called `sum_both_pos_rows` that takes two matrices, `A` and `B`, that have the same number of rows and the same number of columns, and returns a matrix that is the sum of the matrices `A` and `B`, except with only those rows for which the elements in that row in *both* `A` and `B` have a positive sum. The `rowSums` function will return a vector that contains the sums of the elements of the rows of a matrix. (Try it out to see how it works!)

Here is an example:

```
> U <- matrix (c(-3,2,0,-1,2,2,4,2),4,2); U
     [,1] [,2]
[1,]   -3    2
[2,]    2    2
[3,]    0    4
[4,]   -1    2
> V <- matrix (c(0,1,-8,-7,4,4,6,8),4,2); V
     [,1] [,2]
[1,]    0    4
[2,]    1    4
[3,]   -8    6
[4,]   -7    8
>
> sum_both_pos_rows(U,V)
     [,1] [,2]
[1,]    3    6
[2,]   -8   10
```