LAB100

Week 12: More with Functions.

Your aim for this workshop is to learn how to create new commands that perform exactly what you want it to do. Upon completing this workshop you should be able to:

- Understand how to produce error and warning messages
- Write functions which take functions as an argument
- Write and use recursive functions

Remember that you need to change the working directory and open a new R script at the start of each workshop.

1 Adapt an Existing Command

When creating new commands we need to be careful with what is requested as input and what we require it to return. Throughout this workshop we shall be creating a command that evaluates the function:

$$f(x) \quad = \quad \pm\sqrt{x}, \qquad \text{for } x \geq 0.$$

The `sqrt` command already exists but it only returns the positive solution. Forgetting the second negative solution is a constant source of frustration for many students. Therefore, let's create a new command that uses `sqrt` and returns both solutions to the above function. First, recall from the previous workshop how we construct a new command using `function`. It begins with the function name, say `mysqrt`, that is assigned a `function` that accepts the input argument `x`. To evaluate the above function we use the `sqrt` command to obtain the positive solution and multiply this with the vector `c(-1,1)` creates both answers. Finally, the `return` command is used to export only the answer from our new command. Type and execute the following into your editor and try it with a few examples:

```
mysqrt <- function(x){
   answer <- c(-1,1)*sqrt(x)
   return(answer)
}
sqrt(5)
mysqrt(5)
sqrt(0)
mysqrt(0)
```

2 Message Commands

Execute the example `sqrt(-1)`.

The command `sqrt` primarily operates with real numbers and therefore returns a not-a-number (`NaN`) value to this example. Furthermore, the command returned a message explaining why the example produced this unusual result. The table below defines three useful messaging commands:

Command	Meaning	Example
`print`	Simply prints a message in the RStudio console. This is primarily used to provide information to the user whilst a command is being performed, such as iteration number in a loop.	`print("4th Loop")`
`warning`	Prints a warning message in the RStudio console after executing the new command. This is useful for informing the user that the output may not be what they expect and may invalidate further calculations.	`warning("Output is NaN")`
`stop`	This command stops RStudio executing any further commands and prints a message to the console. This is useful for identifying errors due to command misuse or exiting from infinite loops.	`stop("Infinite loop!")`

Each of these commands accepts a single string of characters that will be printed as the message in the console. A string is a sequence of letters, numbers or punctuation characters that are enclosed in the quotation marks `" "`.

3 Conditions on the Input Arguments

Here is another example to try: `mysqrt(TRUE)`. The logical constants `TRUE` and `FALSE` have numerical values 1 and 0 respectively. However, the square root of a logical constant does not make any sense and should have never been evaluated. In fact, according to our function, only a single real number should be evaluated. We can adapt the `mysqrt` command such that any misspecified input values are identified and the command is immediately stopped. The command `is.real(x)` is a logical test that returns the constant `TRUE` if the object `x` contains a real number, otherwise it will return `FALSE`. In the example below, the negation command `!` reverses the output from this logical test such that the first `if` statement is evaluated to be `TRUE` when the `mysqrt` command is used with an invalid input argument. In this case, the `stop` command is called and an appropriate

message is printed in the console. We can invoke the same reaction whenever the inputted value is not a single value, i.e. when `length(x)!=1`.

A similar condition can be placed on the input argument whenever our command is used with a negative number. However, the square root of a negative number is a complex number and is therefore classed as not-a-number in the real range. In this case, we can let our command to return the constant `NaN` and use the `warning` command to notify the user that the answers do not lie in the real range.

The square root of every number on the positive real line has two solutions, but there is only one answer for $\sqrt{0}$; otherwise known as a repeated root. We can provide an extra condition in our command such that if the input value is zero, a single answer is returned with a courtesy message, using the `print` command, saying that the solution is a repeated root.

Collating these conditions on the input value creates the following alternative to the `mysqrt` command:

```
mysqrt <- function(x){
   if( !is.real(x) | length(x)!=1 ){
      stop("Input must be a single real number")
   }
   else if(x < 0){
      warning("Roots are complex numbers")
      answer <- NaN
   }
   else if(x == 0){
      print("Repeated root")
      answer <- 0
   }
   else{
      answer <- c(-1,1)*sqrt(x)
   }
   return(answer)
}
```

Type this into your editor and copy it to your console. Make sure that your command works correctly by trying the following examples:

```
mysqrt(5)
```

```
mysqrt(-5)
mysqrt(0)
mysqrt(FALSE)
mysqrt("text")
```

4 Functions taking a function as an argument

We have already seen that functions can take scalars and vectors are arguments and can take multiple arguments. It is also to write a function in which one or more of the arguments is also a function.

For instance, we may want a function that calculates the approximate derivative of a given scalar function. A numerical approximation to the derivative of a function at a value x_0 can be obtained from

$$f^{'}(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

for an appropriately chosen ϵ. We can write a function in R to perform this operation

```
numerical_derivative <- function(f, x, ep = 1e-5) {
    g <- (f(x + ep) - f(x))/ep
    return(g)
}
```

Note that by specifying `ep = 1e-5` within the definition of the function, we have defined a *default* value for ϵ. This means that we do not need to specify `ep` each time the function is called, and if we do not, it will be assumed that $\epsilon = 1 \times 10^{-5}$.

```
numerical_derivative(sin,1,ep=1e-5)
cos(1)
numerical_derivative(cos,1)
-sin(1)
```

Theoretically, we would expect the approximation to improve as we let $\epsilon \to 0$. However, R can only store numbers up to a certain level of accuracy and hence if `ep` is made too small the resulting derivative will represent rounding error.

For instance try the following, which estimates the derivative of $sin(x)$ at $x = 1$ for decreasing values of ϵ.

```
numerical_derivative(sin,1,ep=c(1e-6,1e-7,1e-8,1e-9,1e-10,1e-11,1e-12))
```

We see that the approximation starts to get worse after `1e-8`.

Note that we can provide our own functions to be differentiated and we can supply a vector of values of x. Hence we can use the function to help us plot the derivative of a function.

```
f <- function(x) sin(exp(x) + 0.5*sqrt((x + 1)^2))
g <- numerical_derivative(x,seq(0,1,by=0.01),ep=1e-5)
plot(seq(0,1,by=0.01),g,type='l')
```

5 Recursive functions

A finite simple continued fraction is an expression of the form:

$$x = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{\ddots}}}}$$

for some sequence $a_1, \ldots, a_n$. For instance, we can represent the number $2\frac{12}{17}$ by the continued fraction with

$$\mathbf{a} = (2, 1, 2, 2, 2)$$

meaning

$$2\frac{12}{17} = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2}}}}.$$

Suppose we want to write a function in R to evaluate a simple finite continued fraction with a particular sequence vector $\mathbf{a}$. One way of doing this might be via a `for` loop starting from the inner most term. However, a more efficient representation is possible. Let

$$CF((a_1, \ldots, a_n))$$

denote the continued fraction generated from the sequence $a_1, \ldots, a_n$, then we can note that

$$CF((a_1, \ldots, a_n)) = a_1 + \frac{1}{CF((a_2, \ldots, a_n))}$$

Hence we can write a continued fraction in terms of a simpler continued fraction. To represent this in R, we need to write a function that calls itself

```
CF <- function(a) {
    n <- length(a)
    if (n == 1) {
        out <- a[1]
    }else {
        out <- a[1] + 1/CF(a[2:n])
    }
    return(out)
}
```

Quiz 1: Error checking

Suppose you are writing a function `f(x,y)` that takes inputs `x`,`y` of a certain type. Match the following if statements with the appropriate error message:

(**A**) `if (x < 0)`

(**B**) `if (length(x)>1)`

(**C**) `if (!is.numeric(x) | !is.numeric(y))`

(**D**) `if (any(is.na(y)))`

(i) `stop("x must be a scalar")`
(ii) `stop("x must be non-negative")`
(iii) `stop("y must not contain missing values")`
(iv) `stop("All inputs must be numerical")`

Quiz 2: Continued fractions I

Use the continued fractions function, `CF`, defined in Section 5 to determine which of the following finite continued fractions defined by a sequence $\mathbf{a}$, best approximates $\sqrt[3]{\pi}$

(**A**) $\mathbf{a} = (1, 2, 6, 1, 4)$

(**B**) $\mathbf{a} = (1, 2, 6, 1, 1, 3, 1)$

(**C**) $\mathbf{a} = (1, 2, 7, 1, 2, 12)$

(**D**) $\mathbf{a} = (1, 2, 7, 2, 1, 1, 1)$

(**E**) $\mathbf{a} = (1, 2, 6, 1, 1, 1, 4)$

Quiz 3: Continued fractions II

$\tanh(1)$ can be approximated by a continued fraction defined by the sequence of length n with kth entry

$$a_k = \begin{cases} 0 & \text{if } k = 1 \\ 2k - 3 & \text{if } k > 1 \end{cases}$$

What is the shortest length of sequence $\mathbf{a}$ required to ensure $|\tanh(1) - CF(\mathbf{a})| < 1 \times 10^{-7}$?

Quiz 4: Second derivatives

An approximation to the second derivative of a scalar function can be obtained via

$$f''(x) \approx \frac{f'(x + \epsilon) - f'(x)}{\epsilon}$$

where, in turn, the first derivatives can be approximated via

$$f'(x) \approx \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

Use or adapt the `numerical_derivative` function, taking `ep = 1e-4` and using `x` defined by

```
x <- seq(0,1,by=0.01)
```

to find the x for which $f''(x) = 0$ to 2 d.p. where

$$f(x) = \sin(\exp(x) + \sqrt{(1 + x^2)}).$$

Quiz 5: Recursive function

A recursive function is defined as follows,

$$R(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ R(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ R(m-1, R(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

By writing a function in R that calls itself, find $R(3,5)$
[Note that the number of recursions needed to evaluate the function increases extremely quickly as m and n increase. You may therefore see the error:

```
Error: evaluation nested too deeply: infinite recursion
```

if larger values are considered.]