Exercise: R and bioinformatics

This is a collection of exercise for the presentation on R programming, before you start the assignment, make sure you have:

(1) Completed the pre-reading if you haven't done so yet.

(2) Read the slides of the presentation, it is an updated version which contains more information than the the you see during the presentation.

Part 1. R Basics

(1) Some questions to orientate yourself.

(a) What is the meaning of the following abbreviations: *rm* , *sum* , *prod* , *seq* , *sd* , *nrow*.

(b) For what purpose are the following functions useful : *grep* , *apply* , *gl* , *library* , *source* , *setwd* , *history* , *str*.

Suggested answer:

(a) rm = remove ; sum = summation ; prod = product ; seq = sequence ; sd =standard deviation ; nrow = number of rows.

(b) You can use R help to find out about these functions, grep is searching regular expressions, apply return a vector from a function on the rows or columns of a matrix , gl generate a factor by specifying the pattern of levels , library load add-on packages , source make R reading input from a file or URL , setwd set the working directory to a certain map , history print the command history , str give the structure of an object.

(2) In the presentation we discussed about how to construct factors, here shown how you can construct these factors using function gl() *look up gl in R help if you are not familiar with this function

(a) An experiment with two conditions each with four measurements.

(b) Five conditions each with three measurements.

(c) Three conditions each with five measurements.

Suggested answer:

(a) gl(2,4)

(b) gl(5,3)

(c) gl(3,5)

(3) Consider the following matrix m that's already been constructed,

```
> m = matrix (1:25 , ncol = 5 , dimnames = list (letters[1:5],LETTERS[1:5])) ; m
   A  B  C  D  E
a  1  6 11 16 21
b  2  7 12 17 22
c  3  8 13 18 23
d  4  9 14 19 24
e  5 10 15 20 25
```

(a) Show how to extract element "3".

(b) Show how to subset the first 3 row & the first three column (row a,b,c and column A,B,C).

(c) Show how to obtain the 4th row and all columns.

(d) Create a new matrix from m by removing the second row and the 4th column (Hint: try negative index).

(e) Subset matrix m such that you keep only rows where the value in the "D" column is greater than 17.

(f) Find element-wise product of rows "b"&"d".

Suggested answer:

(a) >m[3,1]

(b) >m[1:3,1:3]

(c) >m[4,]

(d) >m[-2,-4]

(e) >d_gt17 = m[, "D"] > 17
>m [d_gt17 ,]

(f) >m["b" ,] * m["d" ,]

Part 2. Defining functions and programs in R

(1) Defining your own function is an essential part of creating efficient and reproducible analyses as it allows you to organize a sequence of computation into a unit that can be applied to any set of appropriate inputs.

Here is a basic function definition , but look carefully to how it behaves.:

```
> say = function(name,greeting = "hello") +
+ {
+           paste (greeting , name)
+ }
```

Now in your computer, define this function exactly as shown above and test the results of the following function call (i) say ("world") (ii) say ("world" , "goodbye") (iii) say (greeting = "g'day" , name = "Toronto")

Notice how this function say() behaves given different argument and comment on how arguments to a function works in R.

Suggested answer :

Here is output of the function call:

(i) > say ("world")
[1] "hello world"

(ii) say ("world" , "goodbye")
[1] "goodbye world"

(iii) say (greeting = "g'day" , name = "Toronto")
[1] g'day Toronto"

The say() function has one formal argument: *name*. The body of the function is between the curly braces. The return value of a function in R is the value of the last evaluated expression in the body. Arguments to a function can be specify default values, as is the case with *greeting* above. The default value is used if a value is not provided when the function is called.

When calling functions in R, you can provide arguments in the same order as in the definition of the function, or you can name the arguments as shown in the last call to say above. Naming arguments is a good practice because it makes code more self-explanatory and robust (a change in the function's argument order won't impact your call, for example).

(2) Remember the use of special argument name "..." discussed in the presentation,?If not here is a example to refresh your knowledge:

```
> mean.of.all =
      Function (...)
      Mean (c (...))
>mean.of.all (1:3 , 1:5)
[1] 2.625
```

Now design a function that computes the (trimmed) mean of the mean of several vector(Lets call it mean.of.means , it takes several vectors as argument and also has an optional trim argument you can set).

Here is an example of how it will behave:

(i) > mean.of.means (c(1, 2, 6) , c(1, 2, 3))
[1] 2.5
#because the mean of vector [1, 2, 6] is 3, the mean of vector [1, 2, 3] is 2,
#therefor the mean of 2 and 3 becomes 2.5

(ii) > mean.of.means (c(1, 2, 6) , c(1, 2, 3) , trim = 1)
[1] 2
#because the trimmed mean of vector [1, 2, 6] is 2 (just 1 element left after trimming)
#likewise the trimmed mean of vector [1, 2, 3] is 2, therefore the mean of 2 and 2 is 2

Suggested solution:

Here is a function definition that can accomplish this job(you might found other ways to define this function to achieve the same functionality)

```
> mean.of.means =
      function(... , trim = 0) {                    #notice trim is set to default value 0 here
            x = list (...)
            n = length (x)
            means = numeric (n)
            for (i in 1:n)
                  means [i] = mean (x[[i]]) , trim = trim
            mean(means)
            }
```

(3) A world is a palindrome if it does not change when its letters are reversed (e.g. "kayak") , now write a R function is.palindrome() that checks whether an input string is a palindrome or not, for example, function call is.palindrome("foobar") should return FALSE ; is.palindrome("racecar") should return TRUE.

To get you started, read and understand the following script since you can built your function using this script as a template, although this not not necessary since you may be able to come up with better algorithms.

```
>    x = "foobar"
> substring (x , 1:nchar (x) , 1:nchar (x))
[1] "f" "o" "o" "b" "a" "r"
> substring(x , nchar (x):1 , nchar (x):1)
[1] "r" "a" "b" "o" "o" "f"
> paste ( c ("f", "o", "o", "b" ,"a" ,"r" ) , collapse = "" )
[1] "foobar"
> paste ( substring(x , nchar (x):1 , nchar (x):1) , collapse = "")
[1] "raboof"
```

Suggested solution:
Here is one possible solution

```
#let's start with a helper function
> strrev =
      function(x)
      paste( substring (x , nchar(x) : 1 , nchar(x) : 1) , collapse = "" )

> is.palindrome =
      function(x)
      x[1] == strrev( x[1] )
```

Part 3. R and bioinformatics

(1) Read the "R and bioinformatics" section of the presentation slides, get an idea of what's bioconductor if you haven't used this useful bioinformatics tool that's based on the R programming languages before. If you would like to know more about bioconductor, follow the links provided on the last slides and explore these webpages.

(2) Follows the instruction on the slides or alternatively, go to (http://www.bioconductor.org/install/) and install a selection of core bioconductor package. (Tips : use *biocLite()*)

(3) Briefly discuss the use of bioconductor.

Suggested solution:

Bioconductor is a powerful tool for the analysis and comprehension of high-throughput genomic data, it contains 516 software package now (and the number is growing).

Some example of the use of Bioconductor are :

(1) Microarrays
(2) High Throughput Assays
(3) Sequence Data
(4) Annotation

And many more...

(4) (*Optional) Think about how R can be used in the field of bioinformatics in general, list a few uses if you can.

Suggested solution:

This is an open question, if you want to learn more about the use of R programming in bioinforamtics, here is a great book you should read (freely available online) : Applied Statistics for Bioinformatics in R (http://cran.r-project.org/doc/contrib/Krijnen-IntroBioInfStatistics.pdf)