Problem set #7

Ravi Patel

4/11/2017

1) Download and install RStudio. Open it up. Play around. Try some syntax you know from Python. Does it work? If not, see if you can figure out how to do it in R. In particular:

a) Figure out how to get the length of a vector or list

```
exampleVector = c(1,2,3,4,5)
length(exampleVector)
```

```
## [1] 5
```

b) Figure out the equivalent of range(a,b) from Python

Lets create a range from 2 to 11 in two different ways

```
## Using the colon operator
2:11
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

```
## Using 'seq' function
seq(from=2, to=11, by=1)
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

c) Figure out how to initialize a vector of all 0s

```
vectorLength = 15
zeroVector = numeric(vectorLength)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

2) Get used to working with vectorized objects

a) Make two vectors, v1 = c(3,6,7,3,1) and v2 = c(6,3,0,6,1)

```
v1 = c(3,6,7,3,1)
v2 = c(6,3,0,6,1)
```

b) What do you expect the outcome of v1 + v2, v1 * v2, v1 − v2, and v1/v2 will be? Does something go wrong with one of them?

I expect that each operation will be performed element-wise:
v1 + v2 = 9 9 7 9 2

```
v1 + v2
```

```
## [1] 9 9 7 9 2
```

v1 * v2 = 18 18 0 18 1

```
v1 * v2
```

```
## [1] 18 18  0 18  1
```

v1 - v2 = -3 3 7 -3 0

```
v1 - v2
```

```
## [1] -3  3  7 -3  0
```

v1 / v2 = 0.5 2 Inf 0.5 1

```
v1 / v2
```

```
## [1] 0.5 2.0 Inf 0.5 1.0
```

It doesn't seem that anything went wrong. Interestingly, a divide-by-zero returns Inf.

c) Try some other mathematical operations, like exponentiation, square root, log, etc. Do you get the kind of result you expect?

```
# Exponentiation
v1^2
```

```
## [1]  9 36 49  9  1
```

```
# Square root
sqrt(v1)
```

```
## [1] 1.732051 2.449490 2.645751 1.732051 1.000000
```

```
# Log
log(v1)
```

```
## [1] 1.098612 1.791759 1.945910 1.098612 0.000000
```

Results are as expected: operations are performed element-wise on a vector.

3) R is very good at doing matrix operations, like matrix multiplication. The operator for matrix multiplication is %*% (yes, including the percent signs).

a) Reach way back, and try to remember matrix multiplication. Maybe look it up on Wikipedia. Predict what you should get if you do v1 %*% v2. Now try it. Do you get it?

Since v1 and v2 are vectors, the matrix multiplication is the sum of element-wise multiplication. The result should be 55.

```
v1 %*% v2
```

```
##      [,1]
## [1,]   55
```

b) Figure out how to make a matrix in R. Create a matrix m with 2 rows and 2 columns, first row is 3, 6; second row is 7, 1. Also create a vector v = c(3,1).

```
m = matrix(c(3,6, 7,1), nrow=2, ncol=2, byrow=TRUE)
```

```
##      [,1] [,2]
## [1,]    3    6
## [2,]    7    1
```

```
v = c(3,1)
```

```
## [1] 3 1
```

c) What should you get for m%*%v? How about v%*%m? Do it and seeif you're right!

For m%*%v, we should get:

```
##      [,1]
## [1,]   15
## [2,]   22
```

```
m%*%v
```

```
##      [,1]
## [1,]   15
## [2,]   22
```

For v%*%m, we should get:

```
##      [,1] [,2]
## [1,]   16   19
```

```
v%*%m
```

```
##      [,1] [,2]
## [1,]   16   19
```

4) Let's learn how to data.table.

c) Read the resulting table in R using fread.

```
yeast = fread("yeast_results.txt")
```

d) Using data.table syntax, compute the log fold change of every gene in ONE LINE. NB: log fold-change means log2(BY_expression/RM_expression) i.e. how many factors of 2 you need to get from the RM expression level to the BY expression level.

```
logFoldChange = log2(yeast[,by_count]/yeast[,rm_count])
head(logFoldChange, n=20)
```

```
##  [1] -1.029146346         -Inf -0.443606651         -Inf -1.925999419
##  [6] -1.134649527 -0.327361981 -0.007304801          NaN -0.074000581
## [11] -0.361739273 -0.990218979          Inf          Inf  0.105709441
## [16]  0.410088283          NaN -1.287281952  0.060120992 -2.565597176
```

e) In part d, you probably got a ton of NA values. Why? Using data.table syntax, filter out the "bad" rows, and redo part d. This should take ONE LINE

```
cleanLogFoldChange = log2(yeast[rm_count != 0][,by_count]/yeast[rm_count != 0][,rm_count])
head(cleanLogFoldChange, n=20)
```

```
##  [1] -1.029146346         -Inf -0.443606651         -Inf -1.925999419
##  [6] -1.134649527 -0.327361981 -0.007304801 -0.074000581 -0.361739273
## [11] -0.990218979  0.105709441  0.410088283 -1.287281952  0.060120992
## [16] -2.565597176  0.955605881  0.029146346 -0.575312331 -1.031026896
```

f) Another option to solve the NAs in part d is to have pseudo-counts. A pseudo-count is simply a phony observation of exactly one extra read per gene. Using data.table syntax, add pseudo-counts to every gene. This should take ONE line

```
pseudoYeast = yeast[, .(gene_name=gene_name, gene_length=gene_length, rm_pseudo=rm_count+1,by_pseudo=by
```

g) Re-compute log-fold change using the pseudo-counted data. ONE line.

```
pseudoLogFoldChange = log2(pseudoYeast[,by_pseudo]/pseudoYeast[,rm_pseudo])
head(pseudoLogFoldChange, n=20)
```

```
##  [1] -1.01435529 -5.67242534 -0.43609911 -6.64385619 -1.90500349
##  [6] -1.11783649 -0.32598631 -0.00726800  0.00000000 -0.07324898
## [11] -0.36008909 -0.98540262  5.58496250  5.61470984  0.10541224
## [16]  0.40911267  0.00000000 -1.27301849  0.05889369 -2.54189378
```

h) So far, we've been working with raw counts, not FPKM. Using ONE line, compute the FPKM values for every gene.

i) Finally, compute the log fold change of the FPKM values. How does it compare to the FPKM of the counts?