STATS782 Assignment1

Yujie Zhang 130011770 yzhb915

August 3, 2019

Question 1 [10 marks] Identify the patterns in the following sequences, and use :, seq(), rep() and/or other commonlyused operators/functions, but definitely not c() or any explicit loop, to create these sequences:

(a) 0 3 6 9 12 15 18 21 24 27

```
(seq(1:10)-1)*3
```

```
##  [1]  0  3  6  9 12 15 18 21 24 27
```

(b) 1 20 300 4000 50000

```
seq(1:5)*10^(0:4)
```

```
## [1]     1    20   300  4000 50000
```

(c) 2 4 7 11 16 22 29 37 46 56

```
append((2+cumsum(2:10)),2,after=0)
```

```
##  [1]  2  4  7 11 16 22 29 37 46 56
```

(d) 10 13 16 20 23 26 30 33 36 40 43 46

```
rep((seq(0:2)-1)*3) + 10*rep(seq(1:4),each=3)
```

```
##  [1] 10 13 16 20 23 26 30 33 36 40 43 46
```

(e) "a.1" "a.2" "a.3" "b.4" "b.5" "c.6"

```
gsub("\\s", "", paste(rep(letters[seq(1:3)],3:1),".",seq(1:6)))
```

```
## [1] "a.1" "a.2" "a.3" "b.4" "b.5" "c.6"
```

Question 2 [10 marks] The dataset mammals in the package MASS contains average brain weights (g) and body weights (kg) of 62 species of land mammals. Run

```
library(MASS); data(mammals)
species = rownames(mammals) # species names
bodywt = mammals$body # body weights
brainwt = mammals$brain # brain weights
```

Based on the three created vectors species, bodywt and brainwt, use R expressions (not your or my eyes) to find the values below (and show the values only, nothing else). Don't use any (explicit) loop.

(a) The largest brain weight among all species.

```
max(brainwt)
```

```
## [1] 5712
```

(b) The name of the species with the heaviest body.

```
rownames(mammals[mammals$body == max(bodywt),])
```

```
## [1] "African elephant"
```

(c) The names of the species with a brain heavier than 500g and body weight less than 1000kg.

```r
rownames(mammals[brainwt > 500 & bodywt < 1000,])
```

```
## [1] "Horse"   "Giraffe" "Human"
```

(d) The name(s) of the species with their average body weights closest to the mean.

```r
nearest = function(x, y) x[apply(abs(outer(x,y,"-")), 2, which.min)]
rownames(mammals[bodywt == nearest(bodywt,mean(bodywt)),])
```

```
## [1] "Pig"
```

(e) The names of the top five species with the largest ratios between brain and body weights (i.e., brain weight/body weight).

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
top_n(tbl_df(sort(brainwt/bodywt, decreasing = TRUE)),5)
```

```
## Selecting by value
```

```
## # A tibble: 6 x 1
##   value
##   <dbl>
## 1  39.6
## 2  32.3
## 3  28.
## 4  26.3
## 5  25
## 6  25
```

Question 3 [10 marks] (a) Write an R function, with arguments x for x0, x1, . . . , xk and p for p1, . . . , pk, which uses a for loop to compute the above probability. Use it to find the probability values for the following two cases: (i) x = 4:1, p = 1:3 * 0.1; (ii) x = 10:1, p = 9:1 * 0.02. [5 marks]

```r
S1 = function(x, p, k) {
  n = sum(x)
  p0 = 1-sum(p)
  a = factorial(n) - 1
  b = (p0^x[1]) / (factorial(x[1]) - 1)
  prod = a * b
  for(i in 2:k)
    prod = prod * (p[i-1]^x[i]) / factorial(x[i])

  prod
}
```

```
x = 4:1
p = 1:3 * 0.1
S1(x, p, 4)
```

```
## [1] 0.004039011
```

```
x = 10:1
p = 9:1 * 0.02
S1(x, p, 10)
```

```
## [1] 1.447629e-07
```

(b) Rewrite your R function in Part (a) to remove the explicit for loop, and use your new function to recompute the two probabilities in Part (a). [5 marks]

```
S1 = function(x, p, k) {
  n = sum(x)
  p0 = 1-sum(p)
  a = factorial(n) - 1
  b = (p0^x[1]) / (factorial(x[1]) - 1)
  s = a * b
  d = c(2:k)
  result = s * prod((p[d-1]^x[d]) / factorial(x[d]))
  result
}

x = 4:1
p = 1:3 * 0.1
S1(x, p, 4)
```

```
## [1] 0.004039011
```

```
x = 10:1
p = 9:1 * 0.02
S1(x, p, 10)
```

```
## [1] 1.447629e-07
```

Question 4 [10 marks] Consider a vector of numbers that are already sorted in ascending order. We are interested in finding the index of the nearest neighbour for each value in the vector itself. (This problem is associated with a data resampling method known as delete-1 cross-validation.) We can certainly use the method described in Chapter 3. However, the method there is general in the sense that it can deal with two different vectors and thus is inefficient for just one vector here. The nearest neighbour of a number in the same vector must be its left or right neighbour. An efficient implementation should utilise this fact. Write two versions of an R function as follows: (a) Using a for loop. [5 marks] (b) Using no loop. [5 marks] Given a vector of increasing values, either function returns the index of the nearest neighbour for each of the values in the vector. For example, running your function for vector x1 created as follows,

```
set.seed(782)
(x1 = sort(round(rnorm(10), 2)))
```

```
##  [1] -2.43 -1.98 -0.92 -0.66 -0.31 -0.22  0.01  0.38  0.82  1.68
```

-2.43 -1.98 -0.92 -0.66 -0.31 -0.22 0.01 0.38 0.82 1.68

it should return nn1(x1) [1] 2 1 4 3 6 5 6 7 8 9

Demonstrate that your functions work for x1, and also give the results for x2 and x3 created as follows:

```
set.seed(123); x2 = sort(rnorm(20))
set.seed(456); x3 = sort(rnorm(50))
```

(a) Using a for loop. [5 marks]

```
nn1 = function(x) {
  length_x = length(x)
  result = c()
  for(i in 1:length_x)
    if (i == 1){
      result[1] = 2
    }
  else if (i == length_x){
    result[length_x] = length_x - 1
  }else {
    a = abs(x[i] - x[i-1])
    b = abs(x[i+1] - x[i])
    if (a > b){
      result[i] = i+1
    }else{
      result[i] = i-1
    }
  }
  result
}


set.seed(782); x1 = sort(round(rnorm(10), 2))
nn1(x1)
```

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

```
set.seed(123); x2 = sort(rnorm(20))
nn1(x2)
```

```
##  [1]  2  3  4  5  4  7  6  7 10 11 10 13 12 15 14 15 18 19 20 19
```

```
set.seed(456); x3 = sort(rnorm(50))
nn1(x3)
```

```
##  [1]  2  3  4  3  4  5  8  7  8 11 10 13 14 15 14 15 18 17 18 21 22 21 22
## [24] 25 26 25 28 27 28 29 32 31 32 35 34 35 38 37 40 39 40 43 44 43 46 47
## [47] 46 49 48 49
```

(b) Using no loop. [5 marks]

```
nn1 = function(x){
  o = abs(outer(x,x,"-"))
  diag(o)=NA
  apply(o,2,which.min)
}

set.seed(782); x1 = sort(round(rnorm(10), 2))
nn1(x1)
```

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

```r
set.seed(123); x2 = sort(rnorm(20))
nn1(x2)
```

```
##  [1]  2  3  4  5  4  7  6  7 10 11 10 13 12 15 14 15 18 19 20 19
```

```r
set.seed(456); x3 = sort(rnorm(50))
nn1(x3)
```

```
##  [1]  2  3  4  3  4  5  8  7  8 11 10 13 14 15 14 15 18 17 18 21 22 21 22
## [24] 25 26 25 28 27 28 29 32 31 32 35 34 35 38 37 40 39 40 43 44 43 46 47
## [47] 46 49 48 49
```

Question 5 [10 marks] As we know, a prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. Write an R function prime(n) that finds all prime numbers less than or equal to n. You may use any algorithm for finding prime numbers. Running your function should produce the following results:

prime(1) integer(0) prime(2) [1] 2

prime(3) [1] 2 3

prime(100) [1] 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 [23] 83 89 97

Answer:

```r
get_prime = function(x,y){
  return(x%%y==0)
}
prime = function(x){
  i = 1:x
  a = outer(i,i,get_prime)
  which(apply(a,1,sum)==2)
}

prime(1)
```

```
## integer(0)
```

```r
prime(2)
```

```
## [1] 2
```

```r
prime(3)
```

```
## [1] 2 3
```

```r
prime(100)
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
## [24] 89 97
```