STATS 782 Assignment 1

Douglas Callaway, ID 714671086

Due: 9 August 2016

1) Use :, seq(), rep() and possibly other commonly-used operators/functions, but definitely not c(), to create the following sequences:

(a) 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4

```
seq(from = 2.0, to = 4.4, by = 0.3)
```

```
## [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4
```

(b) "ax" "ay" "by" "bz" "az" "az"

```
paste(rep(c('a','b'), each = 2), rep(c('x', 'y', 'z'), 1:3), sep = '')
```

```
## [1] "ax" "ay" "by" "bz" "az" "az"
```

(c) TRUE TRUE FALSE FALSE FALSE FALSE

```
(1:6) <= 2
```

```
## [1]  TRUE  TRUE FALSE FALSE FALSE FALSE
```

(d) 1 22 333 4444 55555 666666

```
# implementation of "Smarandache" sequence #1 as depicted by Wolfram MathWorld at
# http://mathworld.wolfram.com/SmarandacheSequences.html; retrieved 4 August, 2016
(1:6) * (10^(1:6) - 1) / 9
```

```
## [1]      1     22    333   4444  55555 666666
```

(e) 0 1 2 3 0 2 4 6 0 3 6 9 0 4 8 12

```
rep(0:3, 4)*rep(1:4, each = 4)
```

```
##  [1]  0  1  2  3  0  2  4  6  0  3  6  9  0  4  8 12
```

2) Calculate Pn(x), for n = 5 and x = −1, −0.8, −0.6, . . . , 1, in the following three different ways of coding:

(a) Using a double loop, for x and k, respectively.

```
n = 5
s = seq(from = -1, to = 1, by = 0.2)

for(x in s){
    summation = 0
    for(k in 0:floor(n/2)){
        summation = summation + ((-1)^k*factorial(2*n-2*k)/
                                 (factorial(k)*factorial(n-k)*factorial(n-2*k))
                                 *x^(n-2*k)
                                )
    }
    cat(sprintf('P%s(%s) = %s \n', n, x, 1/2^n*summation))
}
```

```
## P5(-1) = -1
## P5(-0.8) = 0.399520000000001
```

```
## P5(-0.6) = 0.15264
## P5(-0.4) = -0.27064
## P5(-0.2) = -0.30752
## P5(0) = 0
## P5(0.2) = 0.30752
## P5(0.4) = 0.27064
## P5(0.6) = -0.15264
## P5(0.8) = -0.399520000000001
## P5(1) = 1
```

(b) Using a single loop, for x only.

```
legendre.polynomial = function(x, n) {
    k = 0:floor(n/2)
    1/2^n * sum(
        (-1)^k*factorial(2*n-2*k)/
        (factorial(k)*factorial(n-k)*factorial(n-2*k))
        *x^(n-2*k)
    )
}

for(x in s){
    cat(sprintf('P%s(%s) = %s \n', n, x, legendre.polynomial(x, n)))
}
```

```
## P5(-1) = -1
## P5(-0.8) = 0.399520000000001
## P5(-0.6) = 0.15264
## P5(-0.4) = -0.27064
## P5(-0.2) = -0.30752
## P5(0) = 0
## P5(0.2) = 0.30752
## P5(0.4) = 0.27064
## P5(0.6) = -0.15264
## P5(0.8) = -0.399520000000001
## P5(1) = 1
```

(c) Using no loop.

```
cat(sapply(s, function(x, n) {
    sprintf('P%s(%s) = %s \n', n, x, legendre.polynomial(x, n))
}, n = n))
```

```
## P5(-1) = -1
##  P5(-0.8) = 0.399520000000001
##  P5(-0.6) = 0.15264
##  P5(-0.4) = -0.27064
##  P5(-0.2) = -0.30752
##  P5(0) = 0
##  P5(0.2) = 0.30752
##  P5(0.4) = 0.27064
##  P5(0.6) = -0.15264
##  P5(0.8) = -0.399520000000001
##  P5(1) = 1
```

3) write the R function gap.freq() that, given x and t, returns the frequency table of gap lengths

```r
gap.freq = function(x, t=3) {

    onesIndex = which(x==1)
    l = length(onesIndex)

    # add placeholder to beginning, drop last element
    shiftIndex = c(0, onesIndex)[-(l + 1)]

    # gap length = index(n) - index(n-1) - 1
    # first element is only a placeholder, so it's dropped
    gapLengths = (onesIndex - shiftIndex - 1)[-1]

    # counts for 0 to t-1
    gapLengths.freq = table(factor(gapLengths, levels = 0:(t - 1)))

    # counts for t+
    gapLengths.freq[paste(t, '+', sep = '')] = sum(
        table(factor(gapLengths,
                     levels = {if (t < max(gapLengths)) t:max(gapLengths) else t})
              )
    )

    gapLengths.freq
}
```

```r
s1 = c(0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0)
gap.freq(s1)
```

```
##  0  1  2 3+
##  3  0  1  1
```

```r
set.seed(782)
u = runif(1e6)
```

```r
gap.freq(u <= 0.3, t = 10)
```

```
##     0     1     2     3     4     5     6     7     8     9   10+
## 89933 62900 43962 30758 21686 15032 10483  7609  4967  3676  8591
```

4) The dataset islands in R contains the areas in thousands of square miles of the 48 largest landmasses in the world. Use R expressions or functions to find:

(a) the area of the largest landmass.

```r
max(islands)
```

```
## [1] 16988
```

(b) the number of landmasses with areas between 100 and 1000 square miles.

```r
length(islands[islands >= 100 & islands <= 1000])
```

```
## [1] 6
```

(c) the ranking of the area of the North Island of New Zealand (New Zealand (N)) in the world.

```r
# reverse ordering rank (from highest to lowest)
length(islands) - rank(islands)['New Zealand (N)'] + 1
```

```
## New Zealand (N)
##              21
```

(d) the name of the landmass that has the most similar area to New Zealand (North and South Islands).

```r
totalNewZeland = islands['New Zealand (N)'] + islands['New Zealand (S)']
names(islands[abs(islands - totalNewZeland) == min(abs(islands - totalNewZeland))])
```

```
## [1] "Honshu"
```

(e) the names of the top 10 largest landmasses.

```r
names(head(sort(islands, decreasing = TRUE), n = 10))
```

```
##  [1] "Asia"          "Africa"        "North America" "South America"
##  [5] "Antarctica"    "Europe"        "Australia"     "Greenland"
##  [9] "New Guinea"    "Borneo"
```

5)

(a) Implement the midpoint rule

```r
integrate.midpoint = function(a, b, n, FUN) {

    h = (b - a) / n
    sequence = seq(from = a, to = b, by = h)
    summation = 0

    for (i in 0:(n - 1)) {
        summation = summation + FUN((sequence[i + 1] + sequence[i + 2]) / 2)
    }

    h * summation
}
```

```r
integrate.midpoint(a = 0, b = 1, n = 1e6, function(x) (x * (1 - x))^(-1 / 2))
```

```
## [1] 3.140383
```

(b) Implement the iterative version of the midpoint rule

```r
integrate.midpoint.dynamic = function(a, b, FUN) {

    j = 0
    n = 3^j
    h = (b - a) / n

    # initial estimate at midpoint of a, b
    result.old = h * FUN((a + b) / 2)

    # subdivide by multiples of three and estimate until desired precision
    repeat {

        result.new = {

            sequence.sub = seq(from = a, to = b, by = h / 3^(j + 1))
            summation = 0

            # measure new midpoints to left and right of each result.old midpoint
```

```
            for (i in 0:(n - 1)) {
                summation = summation +
                    FUN((sequence.sub[3 * i + 1] + sequence.sub[3 * i + 2]) / 2) +
                    FUN((sequence.sub[3 * i + 3] + sequence.sub[3 * i + 4]) / 2)
            }

            # combine with result.old; correct by n subdivisions
            1 / 3 * (result.old + h * summation / n)
        }

        if (abs(1 - result.old / result.new) <= 1e-3) {
            break
        }

        else {
            result.old = result.new
            j = j + 1
            n = 3^j

        }
    }

    result.new
}
```

```
integrate.midpoint.dynamic(a = 0, b = 1, function(x) (x * (1 - x))^(-1 / 2))
```

```
## [1] 3.138718
```