Simple R Functions

Kayla Ippongi

January 26, 2018

1.

(a) Write functions `tmpFn1` and `tmpFn2` such that if `xVec` is the vector $(x_1, x_2, ..., x_n)$, then `tmpFn1(xVec)` returns vector $(x_1, x_2^2, ..., x_n^n)$ and `tmpFn2(xVec)` returns the vector $(x_1, \frac{x_2^2}{2}, ..., \frac{x_n^n}{n})$.

Here is `tmpFn1`

```
tmpFn1 <- function(xVec){
  return(xVec^(1:length(xVec)))
}

## simple example
a <- c(2, 5, 3, 8, 2, 4)

b <- tmpFn1(a)
b
```

```
## [1]    2   25   27 4096   32 4096
```

and now `tmpFn2`

```
tmpFn2 <- function(xVec2){

  n = length(xVec2)

  return(xVec2^(1:n)/(1:n))
}



c <- tmpFn2(a)
c
```

```
## [1]    2.0000   12.5000    9.0000 1024.0000    6.4000  682.6667
```

(b) Now write a fuction `tmpFn3` which takes 2 arguments x and n where x is a single number and n is a strictly positive integer. The function should return the value of

$$1 + \frac{x}{1} + \frac{x^2}{2} + \frac{x^3}{3} + ... + \frac{x^n}{n}$$

```
tmpFn3 <- function(x,n){

  nVect = 1:n
  return(1 + sum(x^(nVect)/(nVect)))
}
```

```
c <- tmpFn3(2,5)
c
```

```
## [1] 18.06667
```

2. Write a funciton `tmpFn(xVec)` such that if `xVec` is the vector $x = (x_1, ..., x_n)$ then `tmpFn(xVec)` returns the vector of moving averages:

$$\frac{x_1 + x_2 + x_3}{3}, \frac{x_2 + x_3 + x_4}{3}, ..., \frac{x_{n-2} + x_{n-1} + x_n}{3}$$

Try out your function. `tmpFn`(c(1:5,6:1))

```
tmpFn <- function(x){

  num = colSums(matrix(x,nrow = 3))
  denom = 3
  return(num/denom)
}

c <- tmpFn((c(1:5,6:1)))
```

```
## Warning in matrix(x, nrow = 3): data length [11] is not a sub-multiple or
## multiple of the number of rows [3]
```

```
c
```

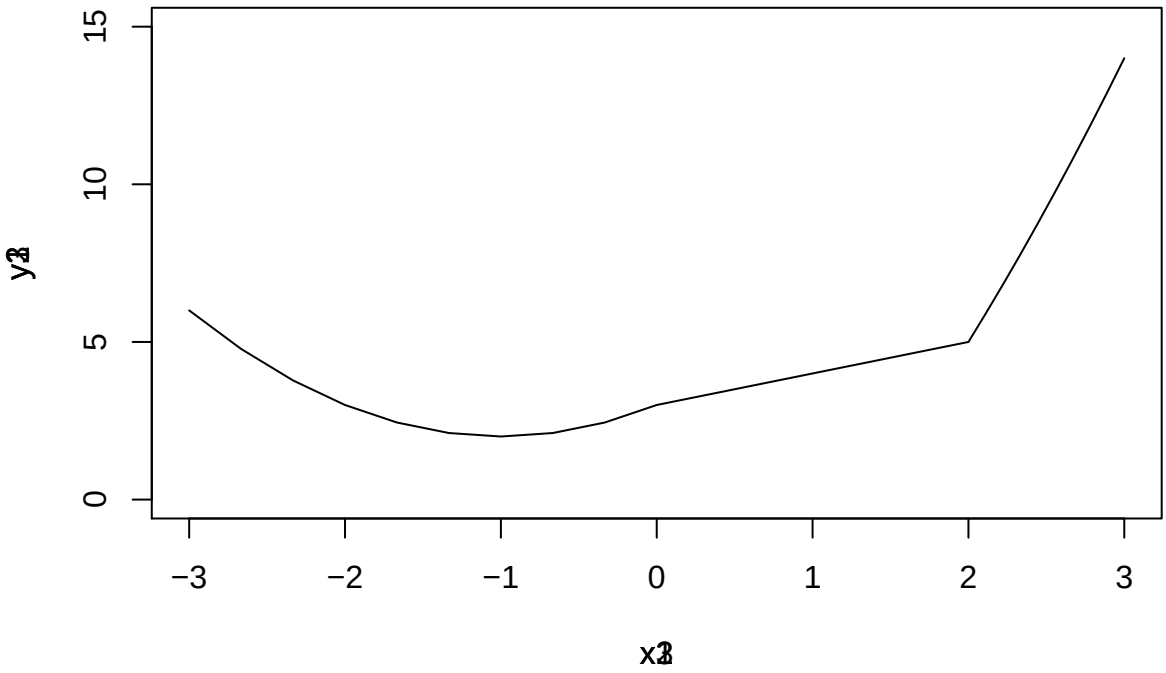
```
## [1] 2.000000 5.000000 4.000000 1.333333
```

3. Consider the continuous function

$$f(x) = \begin{cases} x^2 + 2x + 3 & if \quad x < 0 \\ x + 3 & if \quad 0 \leq x < 2 \\ x^2 + 4x - 7 & if \quad 2 \leq x \end{cases}$$

Write a function `tmpFn` which takes a single argument `xVec`. the function should return the vector the values of the function $f(x)$ evaluated at the values in `xVec`.
Hence plot the function $f(x)$ for $-3 < x < 3$.

```
x1 <- seq(from = -3, to = 0, length.out = 10)
x2 <- seq(from = 0, to = 2, length.out = 10)
x3 <- seq(from = 2, to = 3, length.out = 10)

y1 <- x1^2 + 2*x1 + 3
y2 <- x2 + 3
y3 <- x3^2 + 4*x3 -7

tmpFn <- function(x){
#function 1
plot(x1,y1,type = 'l',xlim=c(-3, 3), ylim=c(0, 15))
par(new = TRUE)

#function 2
plot(x2,y2,type = 'l', axes = FALSE ,xlim=c(-3, 3), ylim=c(0, 15))
par(new = TRUE)

#function 3
plot(x3,y3,type = 'l', axes = FALSE ,xlim=c(-3, 3), ylim=c(0, 15))

}
tmpFn(x)
```



4. Write a function which takes a single argument which is a matrix. The function should return a matrix which is the same as the function argument but every odd number is doubled.

Hence the result of using the function on the matrix

$$\begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$$

should be:

$$\begin{bmatrix} 2 & 2 & 6 \\ 10 & 2 & 6 \\ -2 & -2 & -6 \end{bmatrix}$$

```
A = matrix(c(1,5,2,1,2,-1,3,6,-3),nrow = 3,ncol=3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    3
## [2,]    5    2    6
## [3,]    2   -1   -3
```

```
 A <- ifelse(A %% 2 !=0,A*2,A)
 A
```

```
##      [,1] [,2] [,3]
## [1,]    2    2    6
## [2,]   10    2    6
## [3,]    2   -2   -6
```

5. Write a function which takes 2 arguements n and k which are positive integers. It should return the nxn matrix:

$$\begin{bmatrix} k & 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & k & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & k & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & k & \cdots & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \cdots & k & 1 \\ 0 & 0 & 0 & 0 & \cdots & 1 & k \end{bmatrix}$$

```
tmpFn <- function(n,k){
matE <- matrix(rep(0,n*n), nrow = n, byrow = TRUE)
matE[abs(row(matE)-col(matE))==1] <- 1
matE
diag(matE) <- k
matE
}
tmpFn(5,2)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    1    0    0    0
## [2,]    1    2    1    0    0
## [3,]    0    1    2    1    0
## [4,]    0    0    1    2    1
## [5,]    0    0    0    1    2
```

6. Suppose an angle α is given as a positive real number of degrees.

If $0 \leq \alpha < 90$ then it is quadrant 1. If $90 \leq \alpha < 180$ then it is quadrant 2.
if $180 \leq \alpha < 270$ then it is quadrant3. if $270 \leq \alpha < 360$ then it is quadrant 4.
if $360 \leq \alpha < 450$ then it is quadrant 1.
And so on ...

```
quadrant <- function(alpha){
if(alpha < 90){
  print('quadrant 1')
}
if(alpha > 90 && alpha <= 180){
  print('quadrant 2')
}
if(alpha > 180 && alpha <= 270){
  print('quadrant 3')
}
if(alpha > 270 && alpha <= 360){
  print('quadrant 4')
}
}
quadrant(87)
```

```
## [1] "quadrant 1"
```

Write a function `quadrant(alpha)` which returns the quadrant of the angle α.

7.

(a) Zeller's congruence is the formula:

$$f = ([2.6m - 0.2] + k + y + [y/4] + [c/4] - 2c) mod 7$$

where $[x]$ denotes the integer part of x; for example $[7.5] = 7$.

Zeller's congruence returns the day of the week f given:

k = the day of the month
y = the year in the century
c = the first 2 digits of the year (the century number)
m = the month number (where January is month 11 of the preceding year, February is month 12 of the preceding year, March is month 1, etc.)
For example, the date 21/07/1‘963 has $m = 5, k = 21, c = 19, y = 63$;
the date 21/2/63 has $m = 12, k = 21, c = 19, and y = 62$.
Write a function `weekday(day,month,year)` which returns the day of the week when given the numerical inputs of the day, month and year.
Note that the value of 1 for f denotes Sunday, 2 denotes Monday, etc.

```
weekday <- function(day,month,year){
y <- year%%100
c <- substr(year,1, nchar(year)-2)
y <- as.numeric(y)

c <- as.numeric(c)

result <- ((2.6*month-0.2) + day + year + (y/4) + (c/4)-2*c )%%7

if(result == 1){
  print('Sunday')
}
if(result == 2){
  print('Monday')
}
if(result == 3){
  print('Tuesday')
}
if(result == 4){
  print('Wednesday')
}
if(result == 5){
  print('Thursday')
}
if(result == 6){
  print('Friday')
}
if(result == 7){
  print('Saturday')
}
}
weekday(1,2,2018)
```

(b) Does your function work if the input parameters day, month, and year are vectors with the same length and valid entries?

No it doesn't seem to work: i tested it with these vectors as parameters and got warnings because if statements were implemented to work with 1 number not a vector of numbers.

```
x <- c(1:5)
y <- c(2,2,2,2,2)
z <- c(2018,2018,2018,2018,2018)
weekday(x,y,z)
```

```
## Warning in if (result == 1) {: the condition has length > 1 and only the
## first element will be used

## Warning in if (result == 2) {: the condition has length > 1 and only the
## first element will be used

## Warning in if (result == 3) {: the condition has length > 1 and only the
## first element will be used

## Warning in if (result == 4) {: the condition has length > 1 and only the
## first element will be used

## Warning in if (result == 5) {: the condition has length > 1 and only the
## first element will be used

## Warning in if (result == 6) {: the condition has length > 1 and only the
## first element will be used

## Warning in if (result == 7) {: the condition has length > 1 and only the
## first element will be used
```

HOWEVER, To overide these warnings i could simply put my if statments in a for loop and iterate through each number in the result vector

8.

(a) Write a function testLoop which takes the single argument n and returns the first n 1 values of the sequence

```
testLoop <- function(n){
  v <- c(1,2)
  for (i in n){
    if (i == 0){
      #v <- seq(v, 1)
      print(v)
      next
    }
    if(i == 1){
      #v <-seq(v,2)
      print(v)
      next
    }
    else{
      x_j = v[[i-1]] + 2/(v[[i-1]])
      v <-c(v,x_j)
    }
  }
  return(v)
}
```

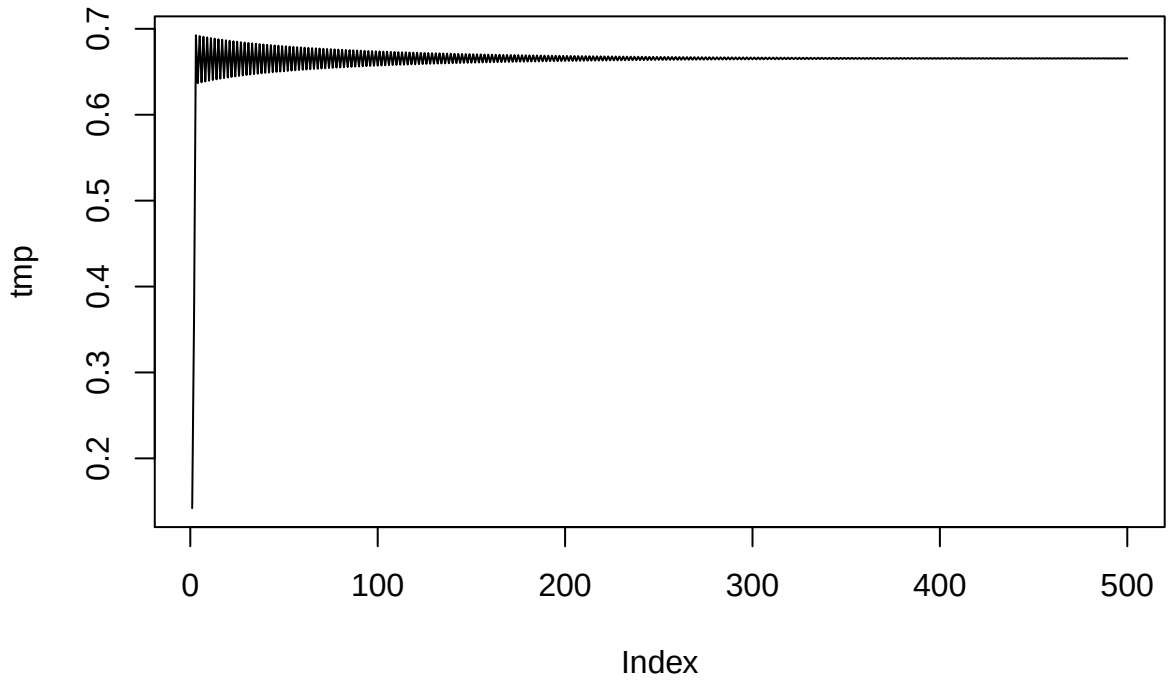
(b) Now write a function testLoop2 which takes a single argument yVec which is a vector. The function should return

```
testLoop2 <- function(yVec){
  n <- length(yVec)
  result <- sum(exp(1:n))
  print(result)
  return(result)
}
```

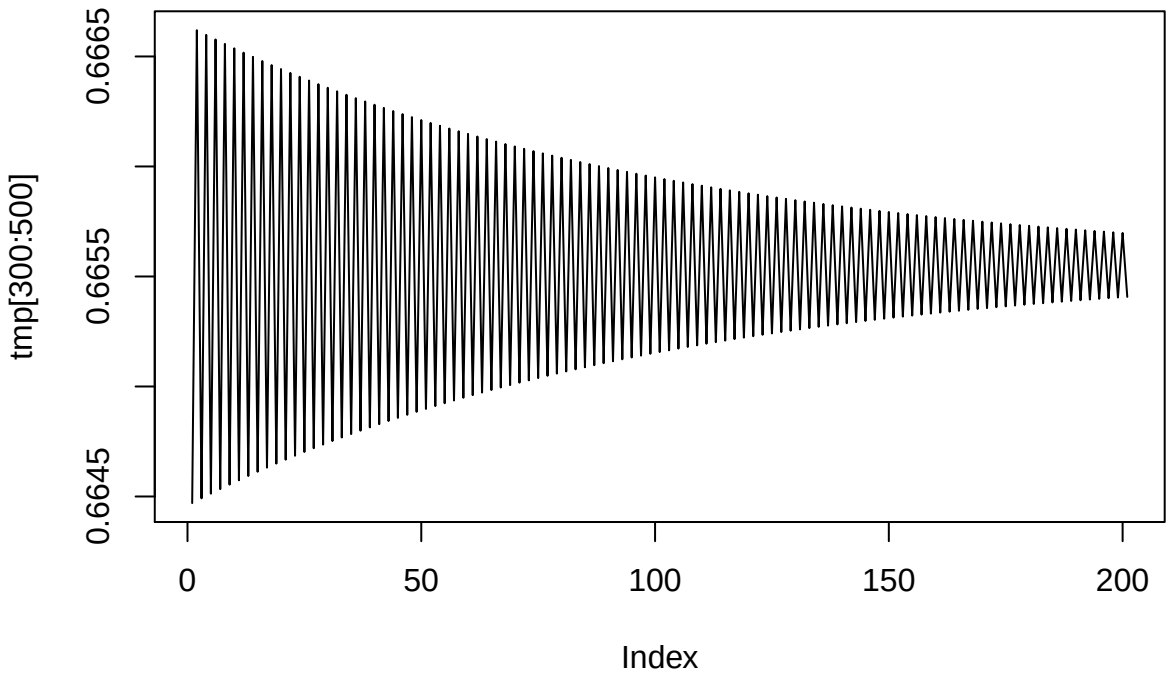
9

(a) Write a function quadmap(start, rho, niter)

```
quadmap <- function(start,rho,niter){
  result <-c(rho*start*(1-start))
  for (i in 1:niter){
    if (i==1){
      next
    }
    else{
      #print(i)
      #print(result)
      result <-c(result,rho*result[i-1]*(1-result[i-1]))
    }
  }
  return(result)
}

tmp <- quadmap(start=0.95, rho=2.99, niter=500)
plot(tmp, type="l")
```



```
plot(tmp[300:500], type="l")
```



(b)

```
fun1 <- function(start,rho){
  check = start
  count = 0
  distance = abs(start-rho)
  print(distance)
  for (i in 1:300){
    distance = distance - 0.02
    print(distance)
    count = count + 1
    print(count)
    if (distance <= 0.02){
      #print(check)
      print(count)
      return(count)
    }
  }
}
```

10

(a)

```
xVect <- seq(from = 2, to = 56, by = 3)
tempFn<- function(xVect){
  m  = mean(xVect)
  n = length(xVect)
  r1 <- (sum(((2:n)-m) * (xVect[1])-m))/(sum((1:n)-m)^2)
  r2 <- (sum(((3:n)-m) * (xVect[1])-m))/(sum((2:n)-m)^2)
  print(r1,r2)
}
```

(b)

```
xVect <- seq(from = 2, to = 56, by = 3)
tempFn<- function(xVect,k){
  m  = mean(xVect)
  n = length(xVect)
  r1 <- (sum(((2:n)-m) * (xVect[1])-m))/(sum((1:n)-m)^2)
  result <- sapply(xVect,(sum(((2:n)-m) * (xVect[1])-m))/(sum((1:n)-m)^2))

  print(result)
}
```