MA615 Assignment 2 Part 2

Matthew Ciaramitaro

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){
  #x is number, n is strictly positive integer
  rng = 1:n
  return(1 + sum(x^rng / rng))

}
```

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

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

```
tmpFn <- function(xVec){
  n <- length(xVec)
  rng1 <- 1:(n-2)
  rng2 <- rng1 + 1
  rng3 <- rng2 + 1
  return((xVec[rng1] + xVec[rng2] + xVec[rng3] )/3)
}
c <- tmpFn(c(1:5,6:1))
c
```

```
## [1] 2.000000 3.000000 4.000000 5.000000 5.333333 5.000000 4.000000 3.000000
## [9] 2.000000
```

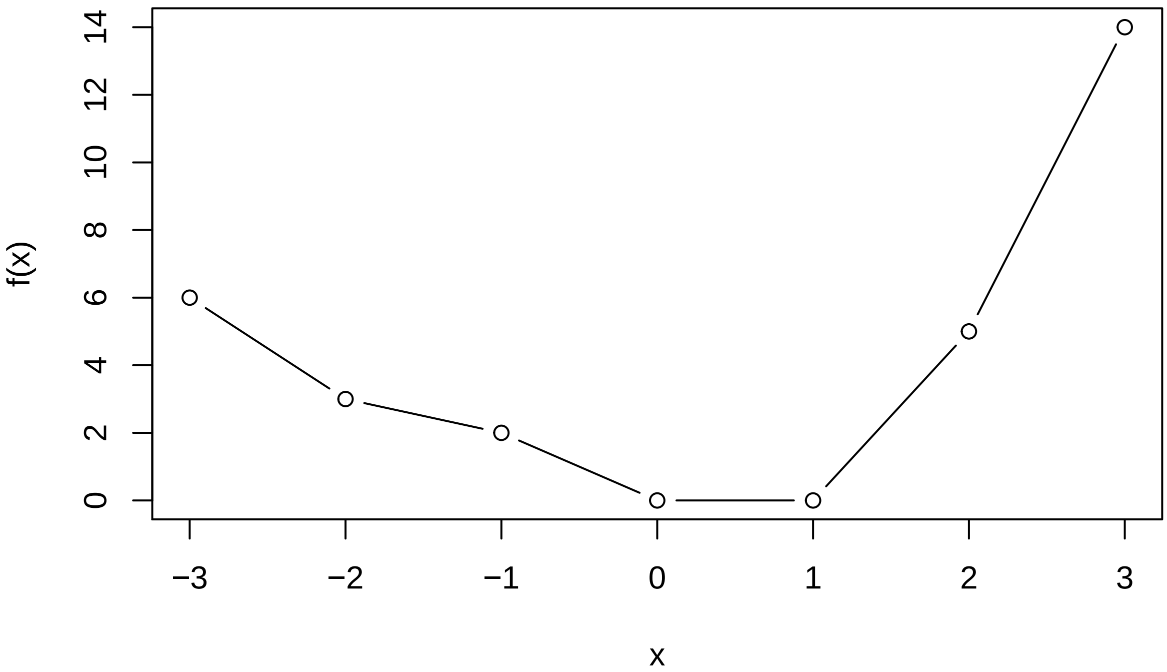
Try out your function. `tmpFn`(c(1:5,6:1))

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 $-3 < x < 3$.

```
tmpFn <- function(x){
  return(
    (x < 0) * (x^2 + 2*x + 3) +
    (0 <= x && x < 2) * (x + 3)     +
    (x >= 2) * (x^2 + 4*x -7)
  )
}
rng = (-3):3
plot(rng, tmpFn(rng), type="b", xlab="x", ylab="f(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:

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

```r
#let's start with indexing.
odddub <-  function(mat){
  #Note this function has the problem of updating the original matrix in place
  mat <- mat
  odds <- which(mat %% 2 == 1, arr.ind=T)
  mat[odds] <-  2 * mat[odds]
  return(mat)
}

test <- matrix(c(1,1,3,5,2,6,-2,-1,-3), nrow=3, byrow=T)
odddub(test) #testing R referencing
```

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

```r
test
```

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

```r
test <-  odddub(test)
test
```

```
##      [,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:

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

```r
fn <- function(n,k){
    mat <- matrix(rep(0,n*n), nrow = n, byrow = TRUE)
    mat[abs(row(mat)-col(mat))==1] <- 1 #produce 1s in columns next to diagonal
    diag(mat) <- k #line diagonal with ks
    return(mat)
}
#testing
fn(6,2)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    2    1    0    0    0    0
## [2,]    1    2    1    0    0    0
## [3,]    0    1    2    1    0    0
## [4,]    0    0    1    2    1    0
## [5,]    0    0    0    1    2    1
## [6,]    0    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 . . .

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

```r
quadrant <- function(alpha){
  ed <-  trunc(alpha / 90)
  return(ed + 1)
```

```
}
#testing
quadrant(180) #should be 3
```

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

```
quadrant(179) #should be 2
```

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

```
quadrant(89) #should be 1
```

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

```
quadrant(0) # should be 1
```

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

7.

(a) Zeller's congruence is the formula:

$$f = ([2.6m - 0.2] + k + y + [y/4] + [c/4] - 2c)mod7$$

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 $m = 12, k = 21, c = 19, andy = 62$.
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.

```
#Assume the months, day and year are given in normal date format (dd, mm, yyyy), such as (3, 2, 2018) f
weekday <- function(day,month,year){
  weekdays = c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")
  k <-  day
  m <- (month + 9) %% 12 + 1
  c <- trunc(year / 100)
  y <- year - 100*c
  index <- ((abs(2.6*m - .2) + k + y + trunc(y/4) + trunc(c/4) -2*c - 1)  %% 7) + 1 #indexing was faili


}
#testing
weekday(5,2, 2018) #should be Monday
```

```
## [1] "Monday"
```

```r
weekday(6,2, 2018) #should be Tuesday
```

```
## [1] "Tuesday"
```

```r
weekday(7,2, 2018) #should be Wednesday
```

```
## [1] "Wednesday"
```

```r
weekday(8,2, 2018) #should be Thursday
```

```
## [1] "Thursday"
```

```r
weekday(9,2, 2018) #should be Friday
```

```
## [1] "Friday"
```

```r
weekday(10,2, 2018) #should be Saturday
```

```
## [1] "Saturday"
```

```r
weekday(11,2, 2018) #should be Sunday
```

```
## [1] "Sunday"
```

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

```r
#testing with vectors
days <-  5:11
months <-  rep(2, times=length(days))
years <-  rep(2018, times=length(days))
#should list all days of the week starting from Monday
weekday(days,months,years)
```

```
## [1] "Monday"    "Tuesday"   "Wednesday" "Thursday"  "Friday"    "Saturday"
## [7] "Sunday"
```

```r
#It does work
```

8.

(a) Suppose $x_0 = 1$ and $x_1 = 2$ and

$$x_j = x_{j-1} + 2/x_{j-1}$$

for $j = 1, 2, ...$

Write a function testLoop(n) that returns the first n - 1 elements of the sequence

```r
testLoop <- function(n){ #by the name of this function I assume I should use a loop
  xs = rep(1, times=n-1)
  xs[2] = 2
  for(i in 3:length(xs)){
    xs[i] <- xs[i-1] + 2/xs[i-1]
  }
  return(xs)
}
#test
testLoop(10)
```

```
## [1] 1.000000 2.000000 3.000000 3.666667 4.212121 4.686941 5.113659 5.504768
## [9] 5.868090
```

(b) define testLoop2 that calculates

$$\sum_{j=1}^{n} e^j$$

for a vector y

```
testLoop2 <- function(yVec){ #much simpler without a loop
  tmp <- exp(yVec)
  return(sum(tmp))
}
#Test

testLoop2(c(1,0,1)) #should be about 6.44
```

```
## [1] 6.436564
```

9.

Solution of the difference equation

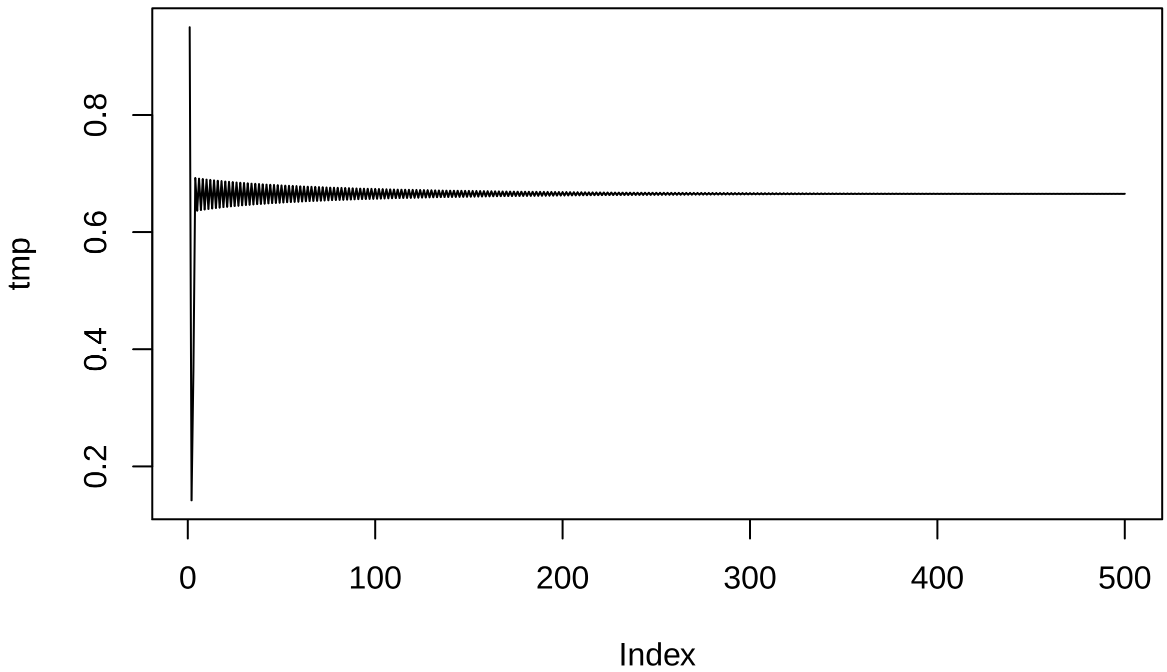
$$x_n = rx_{n1}(1 - x_{n-1})\text{with starting value } x_1$$

(a)

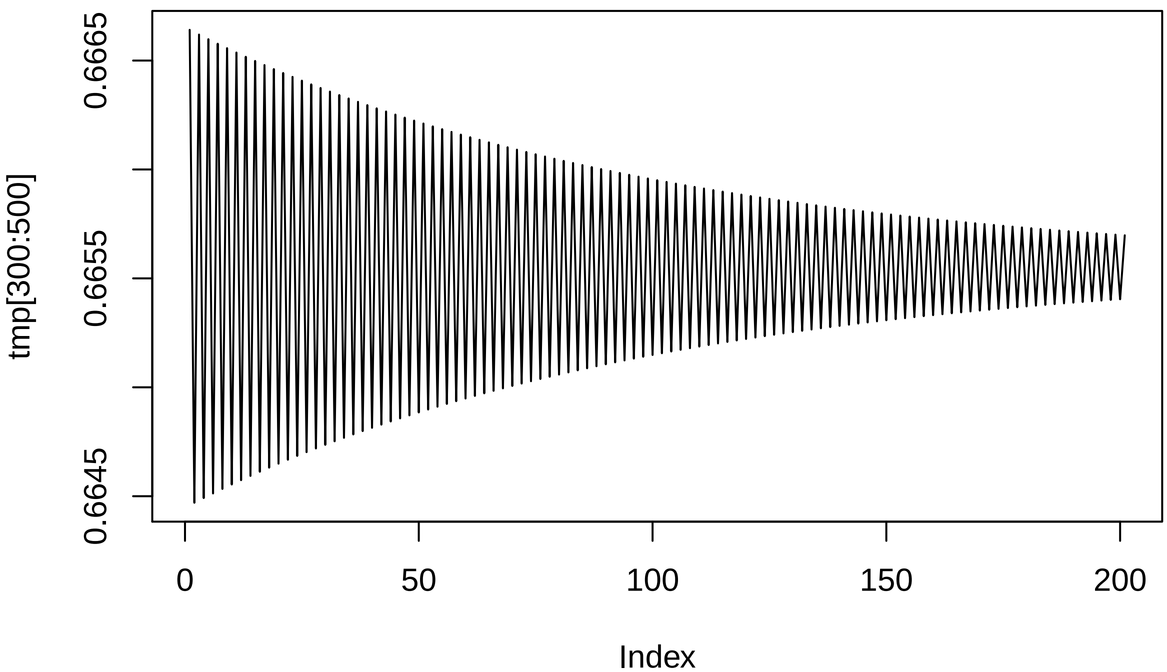
```
quadmap <- function(start, rho, niter){
  xVec <- rep(start,times=niter)
  for(i in 2:niter){
    xVec[i] = rho * xVec[i-1] * (1- xVec[i-1])
  }
  return(xVec)
}
#testing
quadmap(rho=2, start=0.1, niter=50) #should be about .5
```

```
##  [1] 0.1000000 0.1800000 0.2952000 0.4161139 0.4859263 0.4996039 0.4999997
##  [8] 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000
## [15] 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000
## [22] 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000
## [29] 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000
## [36] 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000
## [43] 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000
## [50] 0.5000000
```

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




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



There is a big difference in spread between these two plots

(b)

```
quadmap2 <- function(start, rho, epsilon=.02){
  #finds the number of iterations required to get the difference between consecutive values less than e
  iter <- 1
  x1 <- rho * start * (1- start)
  xVec = c(start, x1)
  while(abs(xVec[2] - xVec[1]) >= epsilon ){ #when it is less than epsilon the loop will stop
    xVec[1] <- xVec[2]
```

```r
    xVec[2] = rho * xVec[1] * (1- xVec[1])
    iter <- iter + 1
  }
  return(iter)
}
#testing
quadmap2(start=.95, rho=2.99) #should be 84
```

```
## [1] 84
```

###10. (a) Given a vector $(x_1, ..., x_n)$ the sample autocorrelation of lag k is defined to be

$$r_k = \frac{\sum_{i=k+1}^{n}(x_i - x_s)(x_{i-k} - x_s)}{\sum_{i=k+1}^{n}(x_i - x_s)^2}$$

Where x_s is the sample mean write a function to solve for r1 and r2

```r
funFn <- function(k,v,mu){
         rng <-  (k+1):length(v)
         rng2 <- rng - k
         bottom <- sum((v[rng] - mu)^2)
         top <- sum((v[rng] - mu) * (v[rng2] - mu) )
         return(top/bottom)
}

tmpFn <- function(xVec){ #This function solves for r_1 and r_2
  mu <-  mean(xVec)
  rs <-  c(1,2)
  sol <- sapply(rs, funFn, v=xVec, mu=mu)
  return(sol)
}
#testing function over given interval
testseq <- seq(from=2, to= 56, by=3)
tmpFn(testseq)
```

```
## [1] 0.9815951 0.9200000
```

since I started out with nearly the generalised code, It is simple to allow for my result to list $r_0, ..., r_k$ (b)

```r
funFn <- function(k,v,mu){
         rng <-  (k+1):length(v)
         rng2 <- rng - k
         bottom <- sum((v[rng] - mu)^2)
         top <- sum((v[rng] - mu) * (v[rng2] - mu) )
         return(top/bottom)
}

tmpFn <- function(xVec, k){
  #Returns vector of all r up to k, where k is between 1 and n-1, n=length(xVec)
  mu <-  mean(xVec)
  rs <-  1:k
  sol <- sapply(rs, funFn, v=xVec, mu=mu)
  return(c(1,sol)) #add 1 to solution for r0 before releasing
}
#testing function over given interval
testseq <- seq(from=2, to= 56, by=3)
tmpFn(testseq, 10)
```

```
## [1]  1.0000000  0.9815951  0.9200000  0.8085106  0.6470588  0.4444444
## [7]  0.2173913 -0.0137931 -0.2307692 -0.4210526 -0.5789474
```