Simple R Functions

Natalya Shelchkova

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){
  xVec <- rep(x,n)
  return (1 + sum((xVec^(1:n))/(1:n)))
}

tmpFn3(2, 3)
```

```
## [1] 7.666667
```

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))

```
tmpFn4 <- function(xVec){
  n <- length(xVec)
  return((xVec[1:(n-2)]+xVec[2:(n-1)] + xVec[3:n])/3)
}
tmpFn4(c(1:5,6:1))
```

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

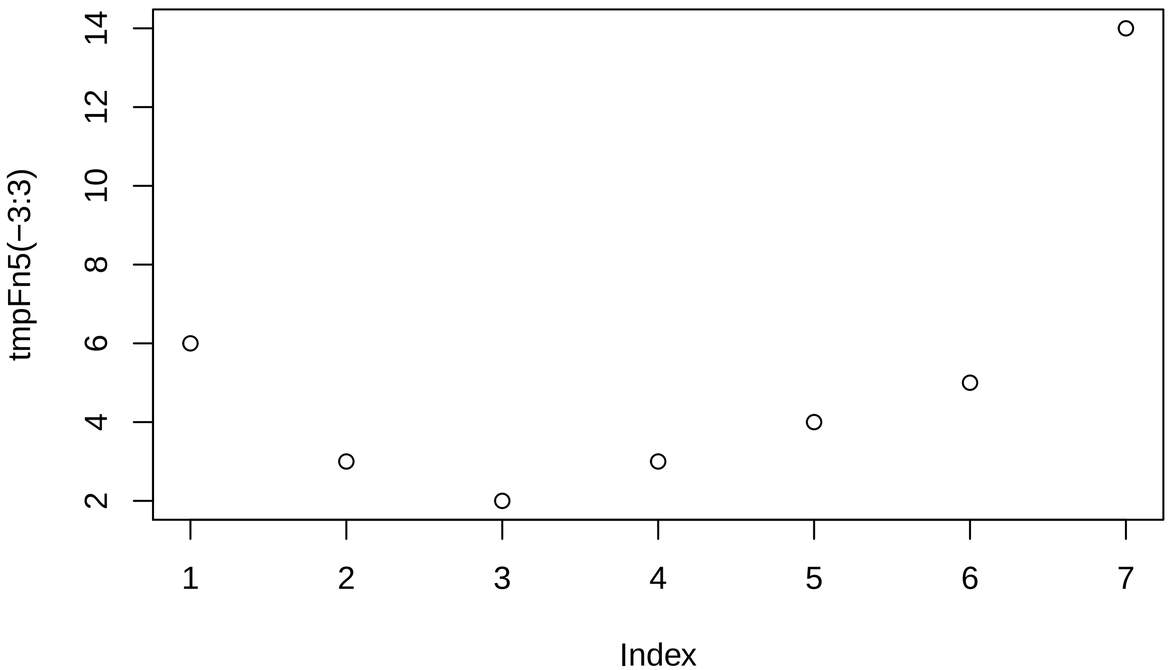
3. Consider the continuous function

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

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 $-3 < x < 3$.

```
tmpFn5 <- function(xVec){
  return(ifelse(xVec < 0, xVec^2+2*xVec+3, ifelse(xVec < 2, xVec+3, xVec^2+4*xVec-7)))
}

plot(tmpFn5(-3:3))
```



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:

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

```
tmpFn6 <- function(A){
  return(ifelse(A%%2 == 1, A*2, A))
}

tmpFn6(matrix(c(1,1,3,5,2,6,-2,-1,-3), byrow = TRUE, nr = 3))
```

```
##      [,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:

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

```
tmpFn7 <- function(n, k){
  return(diag(k, nr = n))
}
```

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.

```
quadrant <- function(alpha){
  if (alpha > 360){
    alpha <- alpha%%360
  }
  if (alpha >= 0 & alpha < 90){
    print("Quadrant 1")
  }
  else if (alpha >= 90 & alpha < 180){
    print("Quadrant 2")
  }
  else if (alpha >= 180 & alpha < 270){
    print("Quadrant 3")
  }
  else {
    print("Quadrant 4")
  }
}

quadrant(90)
```

```
## [1] "Quadrant 2"
```

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 $m = 12, k = 21, c = 19, and y = 62$.
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){
  month_vector <- c(11:12,1:10)
  m <- month_vector[month[1]]
  k <- day[1]
  tmp_year <- as.numeric(strsplit(as.character(year[1]), "")[[1]])
  mod_year <- c(as.numeric(paste0(tmp_year[1],tmp_year[2])), as.numeric(paste0(tmp_year[3],tmp_year[4]))
  if (m == 11 || m == 12){
    y <- (mod_year[2] - 1)
    if (y == 99){
      c <- (mod_year[1] - 1)
    }
    else{
      c <- mod_year[1]
    }
  }
  else {
    c <- mod_year[1]
    y <- mod_year[2]
  }
  f <- (floor((2.6*m) - 0.2) + k + y + floor(y/4) + floor(c/4))%%7
  return (f)

}
```

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

Yes, depending on whether day, month, and year are individual vectors or if its one vector that has [day, month, year].

8.

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

$$x_j = x_{j-1} + \frac{2}{x_{j-1}} \; for \; j = 1, 2, \ldots$$

Write a function `testLoop` which takes the single argument n and returns the first n-1 values of the sequence $x_{j_{j \geq 0}}$: that means the values of $x_0, x_1, x_2, \ldots, x_{n-2}$.

```
testLoop <- function(n){
  x <- 1:(n-1)
  for (ii in 3:(n-1)){
    x[ii] <- x[ii-1] + (2/x[ii-1])
  }
```

```
  return(x)
}
```

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

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

where n is the length of `yVec`.

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

9.

(a) Write a function `quadmap(start, rho, niter)` which returns the vector $(x_1, \ldots, x_n)$ where $x_k = rx_{k-1}(1 - x_{k-1})$ and

`niter` denotes n

`start` denotes x_1, and

`rho` denotes r.

Try out the function you have written:

- for $r = 2$ and $0 < x_1 < 1$ you should get $x_n \to 0.5$ as $n \to \infty$.
- try `tmp <- quadmap(start=0.95, rho=2.99, niter = 500)`

```
quadmap <- function(start, rho, niter){
  x <- 1:niter
  x[1] <- start
  for (ii in 2:niter){
    x[ii] <- rho*x[ii-1]*(1-x[ii-1])
  }
  return(x)
}

tmp <- quadmap(0.95, 2.99, 500)
```

Now switch back to the Commands window and type:

`plot(tmp, type ="l")`

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

(b) Now write a function which determines the number of iterations needed to get $|x_n - x_{n-1}| < 0.02$. So this function has only 2 arguments: `start` and `rho`. (For `start = 0.95` and `rho=2.99`, the answer is 84.)

```
number_of_iterations <- function(start, rho){
  count <- 0
  diff <- 100000
  x_n <- start
  while (diff >= 0.02){
```

```
    x_n1 <- rho*x_n*(1-x_n)
    diff <- abs(x_n1 - x_n)
    count <- count + 1
    x_n <- x_n1
  }
  return(count)
}
```

10.

(a) Given a vector $(x-1,\ldots,x_n)$ the sample autocorrelation of lag k is defined to be

$$r_k = \frac{\Sigma_{i=k+1}^{n}(x_i - \overline{x})(x_{i-k} - \overline{x})}{\Sigma_{i=1}^{n}(x_i - \overline{x})^2}$$

Thus

$$r_1 = \frac{\Sigma_{i=2}^{n}(x_i - \overline{x})(x_{i-k} - \overline{x})}{\Sigma_{i=1}^{n}(x_i - \overline{x})^2} = \frac{(x_2 - \overline{x})(x_1 - \overline{x}) + \cdots + (n_n - \overline{x})(x_{n-1} - \overline{x})}{\Sigma_{i=1}^{n}(x_i - \overline{x})^2}$$

Write a function `tmpFn(xVec)` which takes a single argument `xVec` which is a vector and returns a `list` of two values: r_1 and r_2. In particular, find r_1 and r_2 for the vector (2,5,8,. . . ,53,56).

```
tmpFn8 <- function(xVec){
  x_mean <- mean(xVec)
  n <- length(xVec)
  r <- 1:length(xVec)
  for (k in 1:(length(xVec) - 1)){
    i <- k+1
    r[k] <- sum((xVec[i:n] - x_mean) *(xVec[k:(n-1)] - x_mean))/sum((xVec[k:n] - x_mean)^2)
  }
  return(r)
}

tmpFn8(seq(2,56,3))[1:2]
```

```
## [1] 0.8421053 0.8343558
```

(b) (Harder.) Generalise the function so that it takes two arguments: the vector `xVec` and an integer `k` which lies between 1 and $n-1$ where n is the length of `xVec`

The function should return a vector of the values $(r_0 = 1, r_1, \ldots, r_k)$. If you used a loop to answer part (b), then you need to be aware that much, much better solutions are possible _ see exercises 4. (Hint: `sapply`.)

```
tmpFn9 <- function(xVec, k){
  x_mean <- mean(xVec)
  n <- length(xVec)
  r <- 1:(k)
  for (j in 1:(k)){
    i <- j+1
    r[j] <- sum((xVec[i:n] - x_mean) *(xVec[j:(n-1)] - x_mean))/sum((xVec[j:n] - x_mean)^2)
  }
  return(append(1,r))
}

tmpFn9(seq(2,56,3), 5)
```

```
## [1] 1.0000000 0.8421053 0.8343558 0.8282353 0.8244681 0.8235294
```