Exercise_3_Xiru Lyu

Xiru Lyu

1/26/2018

Problem 1

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

```
tmpFn1 <- function(xVec) {
  for (i in 1:length(xVec)) {
    xVec[i] <- xVec[i]^2
  }
  xVec
}
```

(b) Now write a function `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

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

```
tmpFn2 <- function(xVec) {
  for (i in 1:length(xVec)) {
    xVec[i] <- xVec[i]^2/i
  }
  xVec
}
```

Problem 2

Write a function `tmpFn(xVec)` such that if `xVec` is the vector $x = (x_1, x_2, ..., x_n)$ 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) {
  # create a new vector
  new <- numeric()

  for (i in 3:length(xVec)) {
    new <- c(new,(xVec[i-2]+xVec[i-1]+xVec[i])/3)
  }
  new
}
```

Problem 3

Consider the continuous function

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

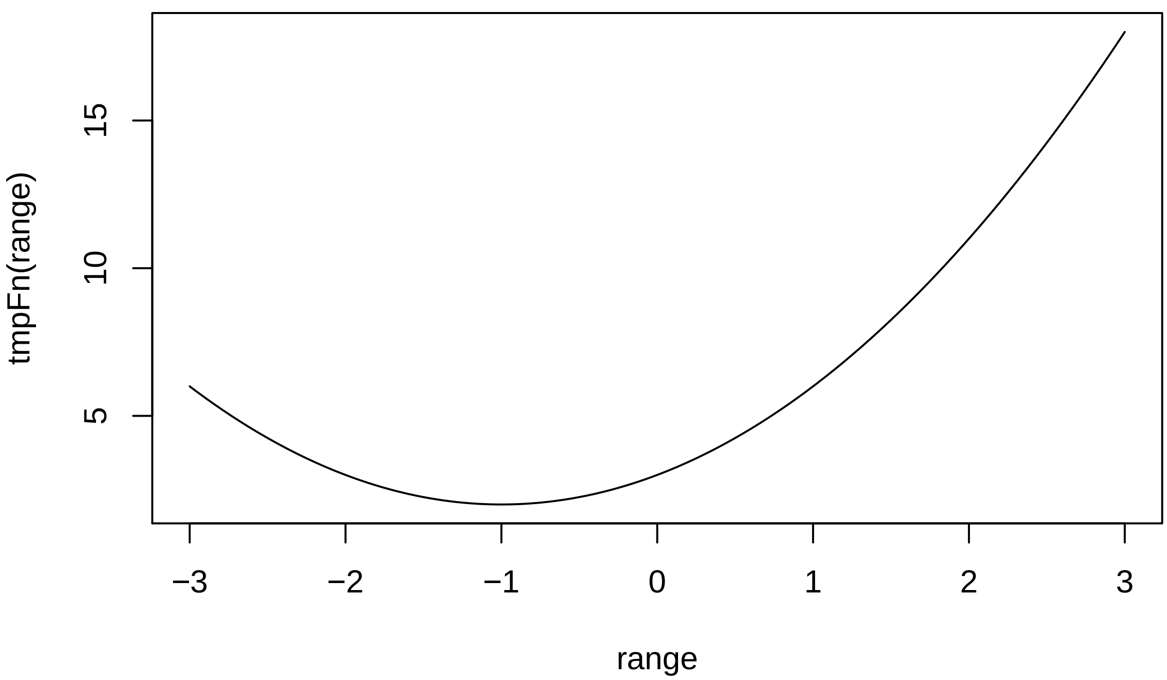
Write a function `tmpFn` which takes a single argument `xVec`. The function should return the vector of values of the function *f(x)* evaluated at the values in `xVec`.

```
tmpFn <- function(xVec){
  if (xVec < 0){
    xVec^2+2*xVec+3
  } else if (xVec < 2) {
    xVec+3
  } else {
    xVec^2+xVec-7
  }
}
```

Hence plot the function *f(x)* for $-3 < x < 3$.

```
# set up the range of the plot
range <- seq(-3,3,length=1000)

# make the plot
plot(range,tmpFn(range),type='l')
```



Problem 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

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

```
fun1 <- function(matrix) {
  # create a new matrix that is a copy of the input matrix
  new_matrix <- matrix

  for (i in 1:nrow(matrix)) {
    for (j in 1:ncol(matrix)) {
      if (matrix[i,j] %% 2 == 1) {
        new_matrix[i,j] <- 2*new_matrix[i,j]
      }
    }
  }
  new_matrix
}
```

Problem 5

Write a function which takes 2 arguments `n` and `k` which are positive integers. It should return the $n \times n$ matrix:

$$\begin{bmatrix} k & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & k & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & k & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & k & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & k & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & k \end{bmatrix}$$

```
fun2 <- function(n,k) {
    new <- diag(k,ncol=n,nrow=n)
    new[abs(col(new)-row(new))==1] <- 1
    new
}
```

Problem 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 quadrant 3. 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) {
    if (alpha >= 0 & alpha < 90) {
      print('quadrant 1')
    } else if (alpha < 180) {
      print('quadrant 2')
    } else if (alpha < 270) {
      print('quadrant 3')
    } else {
      print('qudrant 4')
    }
  }
  else {
    alpha_new <- alpha - 360
    if (alpha_new >= 0 & alpha_new < 90){
      print('quadrant 1')
    } else if (alpha_new < 180) {
      print('quadrant 2')
    } else if (alpha_new < 270) {
      print('qudrant 3')
    } else {
      print('quadrant 4')
    }
  }
}
```

Problem 7

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

Write a function `weekday(day,month,year)` which returns the day of the week when given the nu- merical inputs of the day, month and year.

```
weekday <- function(day,month,year) {
  k <- day
  y <- year %% 100
  c <- trunc(year / 100)
```

```
  if (month < 3) {
    m <- month + 10
    y <- y - 1
  } else {
    m <- month - 2
  }
  f <- (trunc(2.6*m-0.2) + k + y + trunc(y/4) + trunc(c/4) - 2*c) %% 7 + 1
  c('Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')[f]
}
```

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

```
# the function needs to have a part to check the validity of each entry

weekday2 <- function(day,month,year) {
  if (month < 0 | month > 12) {
    break
  } else if (day < 0 | day > 31) {
    break
  } else {
    k <- day
    y <- year %% 100
    c <- trunc(year / 100)
    if (month < 3) {
      m <- month + 10
      y <- y - 1
    } else {
    m <- month - 2
    }
    f <- (trunc(2.6*m-0.2) + k + y + trunc(y/4) + trunc(c/4) - 2*c) %% 7 + 1
    c('Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')[f]
  }
}
```

Problem 8

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

$$x_j = x_{j-1} + \frac{2}{x_{j-1}} \quad \text{and } j = 1, 2, \ldots.$$

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

```
testLoop <- function(n) {
  x <- rep(NA,n-1)
  x[1] <- 1
  x[2] <- 2
  ifelse(x <= 2,x,for (i in 3:n-1) {x[i] <- x[i-1] + 2/x[i-1]})
```

```
  x
  }
```

(b) Now write a funtoin `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)
  seq <- 1:n
  sum(exp(seq))
}
```

Problem 9

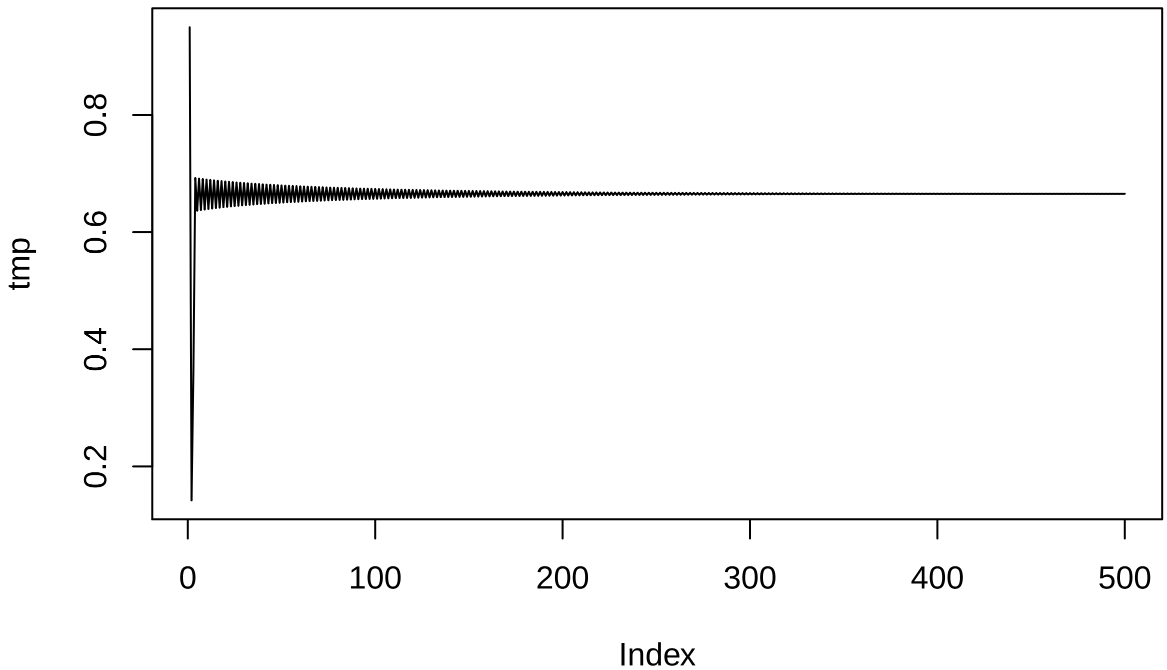
(a) Write a function `quadmap(start,rho,niter)` which returns the vector $(x_1, ..., x_n)$ where $x_k = rx_{k-1}(1 - x_{k-1})$

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

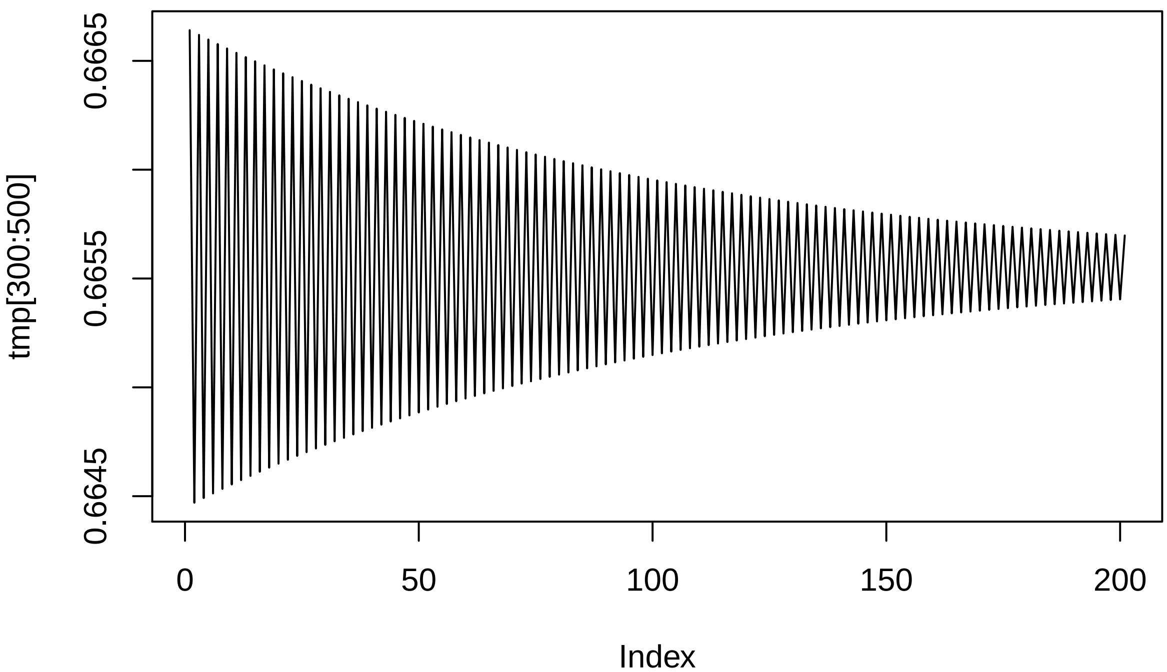
Try out the function you have written

```
tmp <- quadmap(start=0.95, rho=2.99, niter=500)

# try the plot
plot(tmp,type='l')
```




```
# try another 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 only has two arguments.

```
quadmap2 <- function(start,rho){
  x_1 <- start
  niter <- 1
  x_2 <- rho*x_1*(1-x_1)
```

```r
  while (abs(x_2-x_1) >= 0.02) {
    x_1 <- x_2
    x_2 <- rho*x_1*(1-x_1)
    niter <- niter + 1
  }
  niter
}

# test the function
# the function should return 84 for start=0.95, rho=2.99
quadmap2(start=0.95,rho=2.99)
```

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

Problem 10

(a) Given a vector $(x_1, ..., x_n)$, the sample autocorrelation `k` is defined to be

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

Thus

$$r_1 = \frac{\sum_{i=2}^{n}(x_i - \bar{x})(x_{i-1} - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{(x_2 - \bar{x})(x_1 - \bar{x}) + ... + (x_n - \bar{x})(x_{n-1} - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Write a function `tmpFn(xVec)` which takes a single argument `xVec` 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).

```r
tmpFn <- function(xVec) {
  xbar <- mean(xVec)

  # Calculate the value of the denominator
  length_d <- seq(from=1, to=length(xVec))
  denom <- sum((xVec[length_d]-xbar)^2)

  # Calculate the value of the numerator for r1
  length_n_1 <- seq(from=2, to=length(xVec))
  num_1 <- sum((xVec[length_n_1]-xbar)*(xVec[length_n_1-1]-xbar))

  # Calcalate the value of the nuerator for r2
  length_n_2 <- seq(from=3, to=length(xVec))
  num_2 <- sum((xVec[length_n_2]-xbar)*(xVec[length_n_2-2]-xbar))

  # Calculate the value of r1
  r1 <- num_1/denom

  # Calculate the value of r2
  r2 <- num_2/denom

  # Return the list with r1 and r2
  r <- list(r1 = r1,r2 = r2)
```

```
  r
}

# find r1 and r2 for the vector
vec <- seq(from=2,to=56,by=3)
tmpFn(vec)
```

```
## $r1
## [1] 0.8421053
##
## $r2
## [1] 0.6859649
```

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

The function should return a vector of the values $(r_0 = 1, r_1, ..., r_k)$.

```
tmpFn_2 <- function(xVec,k) {
  # Write a function that generates the value for each k
  tmp <- function(j) {
    xbar <- mean(xVec)

    # Calculate the value of the denominator
    length_d <- seq(from=1, to=length(xVec))
    denom <- sum((xVec[length_d]-xbar)^2)

    # Calculate the value of the numerator
    length_n <- seq(from=j+1, to=length(xVec))
    num_1 <- sum((xVec[length_n]-xbar)*(xVec[length_n-j]-xbar))

    r <- num_1/denom }

  list <- c(1,sapply(1:k,tmp))
  list
}
```