R Function Exercises

Brianna Kincaid

February 2, 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)))
}
```

Here is a simple example of **tmpFn1**:

```
a <- c(2, 5, 3, 8, 2, 4)

b <- tmpFn1(a)
b
```

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

Here is **tmpFn2**:

```
tmpFn2 <- function(xVec2){
  n = length(xVec2)
  return(xVec2^(1:n)/(1:n))
}
```

Here is a simple example of **tmpFn2**:

```
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

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

Here is the function **tmpFn3**:

```
tmpFn3 <- function(x,n){
  exp <- 1:n
  return(1+sum((x^exp)/exp))
}
```

Here is a simple example of **tmpFn3** when $x = 5$ and $n = 2$. The expected result is 18.5:

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

```
## [1] 18.5
```

---

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:

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

```
tmpFn <- function(xVec){
  n = 3:length(xVec)
  return((xVec[n-2]+xVec[n-1]+xVec[n])/3)
}
```

Here is the function in use with the following input: `tmpFn`(c(1:5,6:1))

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

```
## [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

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

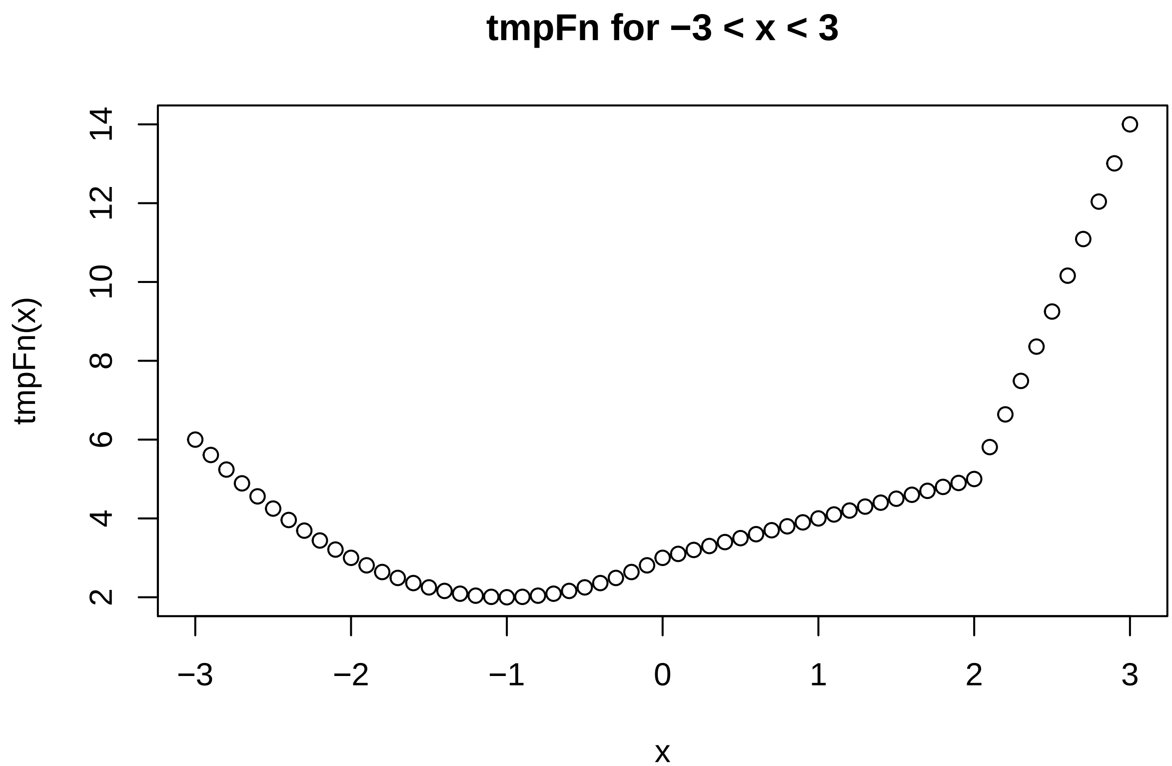
Write a function `tmpFn` which takes a single argument `xVec`. the function should return the vector the values of the function $f(x)$ evaluated at the values in `xVec`. Hence plot the function $f(x)$ for $-3 < x < 3$.

Here is the function:

```
tmpFn <- function(xVec){
  result <- rep(0,length(xVec))
  for(i in 1:length(xVec)){
  if (xVec[i] < 0) {result[i]<-(xVec[i]^2 + 2*xVec[i] + 3)}
  if (xVec[i] >= 0 && xVec[i] <2) {result[i] <- (xVec[i]+3)}
  if (xVec[i] >= 2) {result[i] <- (xVec[i]^2 + 4*xVec[i] - 7)}
  }
  return(result)
}
```

Here is the plot for $-3 < x < 3$.

```
x <- seq(-3,3,0.1)
y <- tmpFn(x)

plot(x,y,ylab='tmpFn(x)',main='tmpFn for -3 < x < 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.

Here is the function:

```
matFn <- function(A) {
  B <- (A%%2==1)*2
  B[B==0]<-1
  return(A*B)
}
```

Here is the result of using the function on the matrix

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

The expected result is:

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

```
A <- matrix(c(1,1,3,5,2,6,-2,-1,-3),nrow=3,ncol=3,byrow=TRUE)
B <- matFn(A)
B
```

```
##      [,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:

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

Here is the function:

```
matFn2 <- function(n,k) {
  A <- diag(n)*k
  B <- abs(row(A)-col(A))
  B[B!=1]<-0
  return(B+A)
}
```

Here is a special case, when $n = 5$ and $k = 2$:

```
C <- matFn2(5,2)
C
```

```
##      [,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 . . .

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

```
quadrant <- function(alpha) {

  if(alpha > 360){alpha <- alpha - 360*(as.integer(alpha/360))}

  if(alpha >= 0 & alpha < 90) {quad <- 1}
  if(alpha >= 90 & alpha < 180) {quad <- 2}
  if(alpha >= 180 & alpha < 270) {quad <- 3}
  if(alpha >= 270 & alpha < 360) {quad <- 4}

  return(quad)
}
```

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 21/07/1963 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.

```
weekday <- function(day,month,year){
  days = c('Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')
  #setting the values
  if(month==2 | month==1) {m <- (month-2)+12} else {m <- (month-2)}
  k <- day
  y <- year
  while(y>100){y <- y-100}
  c <- (year-y)/100
  #Zeller's congruence
  f <- (as.integer((2.6*m) -0.2) + k + y + as.integer(y/4) + as.integer(c/4) - (2*c))%%7
  return(days[f])
}
```

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

```
weekday <- function(day,month,year){
  days = c('Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')
  m <-0
  cent<-0
  k<-0
  y<-0
  f <- 0

  for(i in 1:length(day)){
  #setting the values
  if(month[i]==2 | month[i]==1) {m[i] <- (month[i]-2)+12} else {m[i] <- (month[i]-2)}
  k[i] <- day[i]
  y[i] <- year[i]
  while(y[i]>100){y[i] <- y[i]-100}
  cent[i] <- (year[i]-y[i])/100

  #Zeller's congruence
  f[i] <- (as.integer((2.6*m[i]) -0.2) + k[i] + y[i] + as.integer(y[i]/4) + as.integer(cent[i]/4) - (2*
  }

  return(days[f])
}
```

Let's test the function for a input parameters that are vectors.

```
day <- c(4,5,1,23,15,8)
month <- c(12,11,4,5,7,2)
year <- c(1996,1987,1972,2018,2014,2017)

vector_result <- weekday(day,month,year)
vector_result
```

```
## [1] "Tuesday"   "Wednesday" "Friday"    "Tuesday"   "Monday"    "Wednesday"
```

8.

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

$$x_j = x_{j-1} + \frac{2}{x_{j-1}}$$

for

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

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 $x_0, x_1, x_2, ..., x_{n-2}$.

```
testloop <- function(n) {
  x <- rep(0,n-1)
  x[1] <- 1 #x0
```

```
  x[2] <- 2 #x1
  for (i in 3:(n-1)) {x[i] = x[i-1] + 2/(x[i-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 <- lenght(yVec)
  j <- 1:n
  return(sum(e^j))

}
```

9.*Solution of the difference equation $x_n = rx_{n-1}(1 - x_{n-1})$, with starting value x_1.*

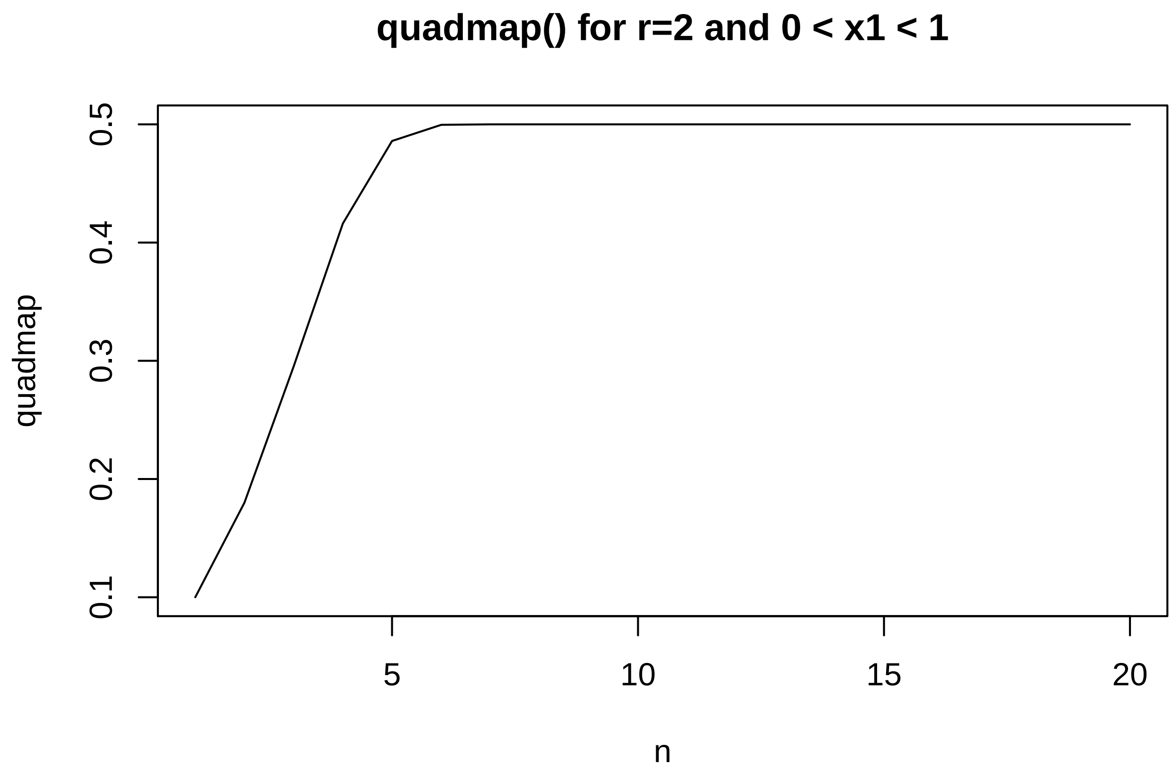
(a) Write a function quadmap(start,rho,niter) which returns the vector $(x_1, ..., x_n)$ where $x_k = rx_{k-1}(1-x_{k-1})$ and ***niter** denotes n,* start denotes x_1, and *rho denotes r.

Here is the function:

```
quadmap <- function(start, rho, niter) {
  x <- rep(0,niter)
  x[1] <- start

  for(i in 2:niter) {
  x[i] <- rho*(x[i-1])*(1-x[i-1])
  }

  #for some reason this worked when I put it in a for loop but it did not work when I
  #pulled it out of the for loop -- can't really figure out why

  return(x)
}
```

Now let's try out the function for $r = 2$ and $0 < x1 < 1$. x_n should go to 0.5 as $n \to \infty$.

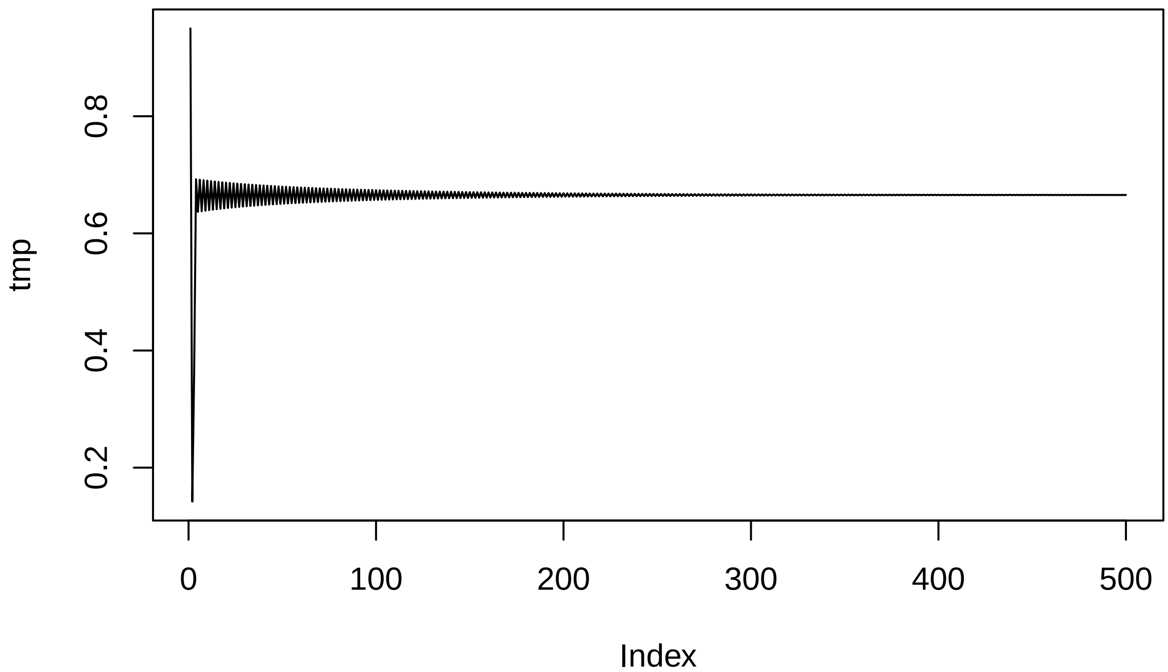
```
y <- quadmap(.1,2,20)
n <- 1:20
plot(n,y,type="l",ylab='quadmap',main='quadmap() for r=2 and 0 < x1 < 1')
```



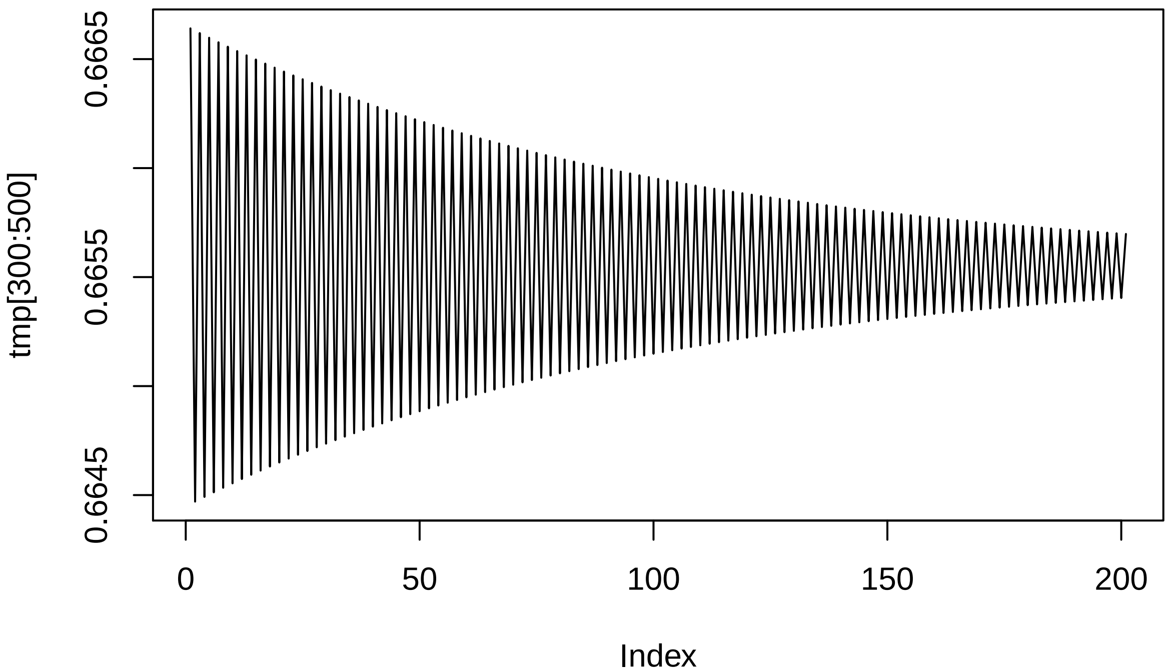
As you can see from the plot, the values of x go to 0.5 as n gets large.

Now let's try out the function with $start = 0.95$, $rho = 2.99$, and $niter = 500$.

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




```
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.)

```
iterations <- function(start,rho) {
  x <- start
  x[2] <- rho*(start)*(1-start)
  n <- 2 #minimum value that n must be

  while(abs(x[n]-x[n-1])>= 0.02) {
    n <- n+1
    x[n] <- rho*(x[n-1])*(1-x[n-1])
  }

  return(n)
}
```

Here is the function called with the inputs `start=0.95` and `rho=2.99`.

```
iterations(0.95,2.99)
```

```
## [1] 85
```

10.

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

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

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).

Here is the function.

```
tmpFn <- function(xVec){
  xb <- mean(xVec) #x-bar

  ## r1
  it <- 2:length(xVec)
  id <- 1:length(xVec)
  r1 <- sum((xVec[it]-xb)*(xVec[it-1]-xb))/sum(((xVec[id]-xb)^2))

  ## r2
  it <- 3:length(xVec)
  id <- 1:length(xVec)
  r2 <- sum((xVec[it]-xb)*(xVec[it-2]-xb))/sum(((xVec[id]-xb)^2))

  return(c(r1,r2))
}
```

Now let's test the function for the vector $(2, 5, 8, ..., 53, 56)$.

```
input <- seq(2,56,3)
output <- tmpFn(input)

r1 <- output[1]
r1
```

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

```
r2 <- output[2]
r2
```

```
## [1] 0.6859649
```

(b) Generalize 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 $(r_0 = 1, r_1, ..., r_k)$.

```
tmpFn2 <- function(xVec,k) {
  r <- 1

  #r[2]=r1, r[3]=r2,... r[k+1]=rk

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

```
  return(r)

}
```