**Name:Khushali Begde**

**Roll no: 07(A)**

**Assignment 3,4,5**

**Code:**

#Exercises 1. Vectors

#1. Create the vectors:

#(a) (1, 2, 3, . . . , 19, 20)

vec1<-c(1:20)

vec1

#(b) (20, 19, . . . , 2, 1)

vec2<-c(20:1)

vec2

#(c) (1, 2, 3, . . . , 19, 20, 19, 18, . . . , 2, 1)

x<-1:20

y<-19:1

vec3<-c(x,y)

vec3

#(d) (4, 6, 3) and assign it to the name tmp.

tmp<-c(4,6,3)

tmp

#(e) (4, 6, 3, 4, 6, 3, . . . , 4, 6, 3) where there are 10 occurrences of 4.

vec4<-rep(c(4,6,3),times=10)

vec4

#(f) (4, 6, 3, 4, 6, 3, . . . , 4, 6, 3, 4) where there are 11 occurrences of 4, 10 occurrences of 6 and

#10 occurrences of 3.

vec5 <- c(rep(c(4,6,3),10),4)

vec5

#(g) (4, 4, . . . , 4, 6, 6, . . . , 6, 3, 3, . . . , 3) where there are 10 occurrences of 4, 20 occurrences of

# 6 and 30 occurrences of 3.

vec6<-rep(c(4,6,3),times=c(10,20,30))

vec6

#2. Create a vector of the values of e x cos(x) at x = 3, 3.1, 3.2, . . . , 6.

x <- seq(3,6,by = 0.1)

my\_vec <- exp(x) \* cos(x)

my\_vec

#3. Create the following vectors:

#(a)

x <- seq(3,36,by = 3)

y <- seq(1,34,by = 3)

v <- 0.1 ^ (x) \* 0.2 ^ (y)

v

#(b)

vev7<-(2^(1:25))/(1:25)

#5. Use the function paste to create the following character vectors of length 30:

#(a) (label 1,label 2, .....,label 30).

paste("label",1:30)

#(fn1,fn2, ..., fn30).

paste("fn", 1:30,sep="")

#6. Execute the following lines which create two vectors of random integers which are chosen with

#replace- ment from the integers 0, 1, . . . , 999. Both vectors have length 250.

#set.seed(50)

#xVec &lt;- sample(0:999, 250, replace=T) yVec &lt;-

# sample(0:999, 250, replace=T)

#Suppose x = (x 1 , x 2 , . . . , x n ) denotes the vector xVec and y = (y 1 , y 2 , . . . , y n ) denotes the vector yVec.

set.seed(50)

xVec <- sample(0:999, 250, replace = TRUE)

yVec <- sample(0:999, 250, replace = TRUE)

# (a) Create the vector (y2 - x1, ..., yn - xn-1)

yVec[-1] - xVec[-length(xVec)]

# (b) Create the vector sin(y1), sin(y2), ..., sin(yn-1)

sin(yVec[-length(yVec)]) / cos(xVec[-1])

# (c) Create the vector (x1 + 2x2 - x3, x2 + 2x3 - x4, ..., xn-2 + 2xn-1 - xn)

xVecLen <- length(xVec)

xVec[-c(xVecLen-1,xVecLen)] + 2\*xVec[-c(1,xVecLen)] - xVec[-c(1,2)]

# (d) Calculate the sum of n-1 elements in xVec

sum(exp(-xVec[-1])/(xVec[-length(xVec)]+10))

#7. This question uses the vectors xVec and yVec created in the previous question and the

#functions sort, order, mean, sqrt, sum and abs.

#(a) Pick out the values in yVec which are >600.

yVec[yVec>600]

#(b) What are the index positions in yVec of the values which are > 600?

(1:length(yVec))[yVec>600]

#(c) What are the values in xVec which correspond to the values in yVec which are > 600?

xVec[yVec>600]

#(d) Create the vector ( x 1 x¯ 1/2 , x 2 x¯ 1/2 , . . . , x n x¯ 1/2 ) where x¯ denotes the mean of the vector

#x = (x 1 , x 2 , . . . , x n ).

sqrt(abs(xVec-mean(xVec)))

#(e) How many values in yVec are within 200 of the maximum value of the terms in yVec?

sum( yVec>max(yVec)-200 )

#(f) How many numbers in xVec are divisible by 2?

sum(xVec%%2==0)

#(g) Sort the numbers in the vector xVec in the order of increasing values in yVec.

xVec[order(yVec)]

#(h) Pick out the elements in yVec at index positions 1, 4, 7, 10, 13, . . . .

yVec[c(T,F,F)]

#8. By using the function cumprod or otherwise, calculate 2 ( 2 4 ) ( 2 4 6 ) ( 2 4 38 )

1+sum(cumprod(seq(2,38,b=2)/seq(3,39,b=2)))

#Exercise 2

#1.Suppose A =[1 1 3 5 2 6 −2 −1 −3]

#(a) Check that A3 = 0 where 0 is a 3 × 3 matrix with every entry equal to 0.

tmp <- matrix( c(1,5,-2,1,2,-1,3,6,-3),nr=3)

tmp

tmp%\*%tmp%\*%tmp

#(b) Replace the third column of A by the sum of the second and third columns.

(tmp[,3]<-tmp[,2]+tmp[,3])

#2. 2. Create the following matrix B with 15 rows:

#B =10 −10 10

#10 −10 10

#· · · · · · · · ·

#10 −10 10

#Calculate the 3 × 3 matrix BTB. (Look at the help for crossprod.)

crossprod(tmp)

#3. Create a 6 × 6 matrix matE with every entry equal to 0. Check what the functions row and col return when applied to matE. Hence create the 6 × 6 matrix:

matE <- matrix(0,nr=6,nc=6)

matE

matE[ abs(col(matE)-row(matE))==1 ] <- 1

#4.Look at the help for the function outer. Hence create the following patterned matrix:

outer(0:4,0:4,"+")

#5.Create the following patterned matrices. In each case, your solution should make use of the special form of the matrix—this means that the solution should easily generalise to creating a larger matrix with the same structure and should not involve typing in all the entries in the matrix.

#(a)

outer(0:4,0:4,"+")%%5

#(b)

outer(0:9,0:9,"+")%%10

#(c)

outer(0:8,0:8,"-")%%9

#6.Solve the following system of linear equations in five unknowns

# x1 + 2x2 + 3x3 + 4x4 + 5x5 = 7

# 2x1 + x2 + 2x3 + 3x4 + 4x5 = −1

# 3x1 + 2x2 + x3 + 2x4 + 3x5 = −3

# 4x1 + 3x2 + 2x3 + x4 + 2x5 = 5

# 5x1 + 4x2 + 3x3 + 2x4 + x5 = 17

# by considering an appropriate matrix equation Ax = y.

AMat <- matrix(0,nr=5, nc=5)

AMat <- abs(col(AMat)-row(AMat))+1

AMat

yVec <- c(7,-1,-3,5,17)

(AMat)%\*%yVec

#7.Create a 6 × 10 matrix of random integers chosen from 1, 2,. . . , 10 by executing the following two lines of code:

set.seed(75)

aMat <- matrix( sample(10, size=60, replace=T), nr=6)

aMat

apply(aMat, 1, function(x){sum(x>4)})

which( apply(aMat,1,function(x){sum(x==7)==2}) )

aMatColSums <- colSums(aMat)

which( outer(aMatColSums,aMatColSums,"+")>75, arr.ind=T )

#8.

#(a)

sum( (1:20)^4 ) \* sum( 1/(4:8) )

#(b)

sum( (1:20)^4 / (3 + outer(1:20,1:5,"\*")))

#(c)

sum( outer(1:10,1:10,function(i,j){ (i>=j)\*i^4/(3+i\*j) }) )

#Exercises 3. Simple Functions

#1. (a) Write functions tmpFn1 and tmpFn2 such that if xVec is the vector (x1, x2, . . . , xn), then tmpFn1(xVec) returns the vector (x1, x22, . . . , xn n)and tmpFn2(xVec) returns the vector

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

tmpFn2 <- function(xVec){ n <- length(xVec) (xVec^(1:n))/(1:n) }

tmpFn1(1:9)

#(b)

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

#2.

tmpFn <- function(xVec)

{

n <- length(xVec)

( x[1:(n-2)] + x[2:(n-1)] + x[3:n] )/3

}

tmpFn( c(1:5,6:1) )

#3.

tmpFn <- function(x)

{

ifelse(x < 0, x^2 + 2\*x + 3, ifelse(x < 2, x+3, x^2 + 4\*x - 7))

}

tmp <- seq(-3, 3, len=100)

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

#4

tmpFn <- function(mat)

{

mat[mat%%2 == 1] <- 2 \* mat[mat%%2 == 1]

mat

}

#5

tmp <- diag(2, nr = 5)

tmp[abs(row(tmp) - col(tmp)) == 1] <- 1

tmp

#6

quadrant <- function(alpha)

{

1 + (alpha%%360)%/%90

}

#7

weekday <- function(day, month, year)

{

flag <- month <= 2

month <- month - 2 + 12\*flag

year <- year - flag

cc <- year %/% 100

year <- year %% 100

tmp <- floor(2.6\*month - 0.2) + day + year + year %/% 4 + cc %/% 4 - 2 \* cc

c("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")[1+tmp%%7]

}

weekday( c(27,18,21), c(2,2,1), c(1997,1940,1963) )

#8 (a)

testLoop <- function(n)

{

xVec <- rep(NA, n-1)

xVec[1] <- 1

xVec[2] <- 2

for( j in 3:(n-1) )

xVec[j] <- xVec[j-1] + 2/xVec[j-1]

xVec

}

#(b)

testLoop2 <- function(yVec)

{

n <- length(yVec)

sum( exp(seq(along=yVec)) )

}

#9 (a)

quadmap <- function(start, rho, niter)

{

xVec <- rep(NA,niter)

xVec[1] <- start

for(i in 1:(niter-1)) {

xVec[i + 1] <- rho \* xVec[i] \* (1 - xVec[i])

}

x

}

#(b)

quad2 <- function(start, rho, eps = 0.02)

{

x1 <- start

x2 <- rho\*x1\*(1 - x1)

niter <- 1

while(abs(x1 - x2) >= eps) {

x1 <- x2

x2 <- rho\*x1\*(1 - x1)

niter <- niter + 1

}

niter

}

#10 (a)

tmpAcf <- function(xVec)

{

xc <- xVec - mean(xVec)

denom <- sum(xc^2)

n <- length(x)

r1 <- sum( xc[2:n] \* xc[1:(n-1)] )/denom

r2 <- sum( xc[3:n] \* xc[1:(n-2)] )/denom

list(r1 = r1, r2 = r2)

}

#(b)

tmpAcf <- function(x, k)

{

xc <- x - mean(x)

denom <- sum(xc^2)

n <- length(x)

tmpFn <- function(j){ sum( xc[(j+1):n] \* xc[1:(n-j)] )/denom }

c(1, sapply(1:k, tmpFn))

}

#Exercises 4. Harder functions

#1(a)

fun4q1a <- function(xVec, yVec){

colSums( outer(yVec, xVec, "<") )

}

#(b)

fun4q1b <- function(xVec, yVec){

rowSums( sapply(yVec, FUN=function(y){y < xVec}) )

}

#(c)

fun4q1c <- function(xVec, yVec){

rowSums( vapply(yVec, FUN=function(y){y<xVec}, FUN.VALUE=seq(along=xVec)) )

}

#(b)

rjr1 <- rnorm(10000)

rjr2 <- rnorm(12000)

system.time(fun4q1a(rjr1,rjr2))

system.time(fun4q1b(rjr1,rjr2))

system.time(fun4q1c(rjr1,rjr2))

#system.time(fun4q1d(rjr1,rjr2))

#2.(a)

tmpFn <- function(mat){

mat[, !apply(is.na(mat), 2, any), drop = F]

}

#(b)

tmpFn2 <- function(mat){

mat[!apply(is.na(mat), 1, any), !apply(is.na(mat), 2, any), drop = F]

}

#3(a)

empCopula <- function( u, v, xVec, yVec )

{

n <- length(xVec)

rVecN <- rank(xVec)/(n+1)

sVecN <- rank(yVec)/(n+1)

sum( (rVecN <= u) & (sVecN <= v) ) /n

}

#(b)

empCopula2 <- function( u, v, xVec, yVec )

{

n <- length(xVec)

rVecN <- rank(xVec)/(n+1)

sVecN <- rank(yVec)/(n+1)

valuesN <- colSums( outer(rVecN, u, "<=")&outer(sVecN, v, "<=") )

cbind( uCoord = u, vCoord = v, empCop=valuesN/n )

}

#4(a)

funA <- function (n)

{

su <- 0

for(r in 1:n)

{

for(s in 1:r)

su <- su+s^2/(10+4\*r^3)

}

su

}

#(b)

funB <- function (n)

{

mat <- matrix(0, ncol=n, nrow=n)

sum( (col(mat)^2)/(10+4\*row(mat)^3)\*(col(mat)<=row(mat)) )

}

#(c)

funC <- function (n)

{

sum( outer(1:n,1:n,FUN=function(r,s){ (s<=r)\*(s^2)/(10+4\*r^3) }) )

}

#(d)

funD <- function (n)

{

tmpfn <- function(r){sum(((1:r)^2)/(10+4\*r^3))}

sum(sapply(1:n, FUN=tmpfn))

}

funE <- function (n)

{

tmpfn <- function(r){sum(((1:r)^2)/(10+4\*r^3))}

sum(unlist(lapply(1:n, FUN=tmpfn)))

}

#(e)

funF <- function (n)

{

tmpf <- function(s,r){(s^2)/(10+4\*r^3)\*(s<=r)}

sum(mapply(tmpf, rep(1:n, times=rep(n,n)), 1:n))

}

#5.(a)

queue1 <- function(n, aRate, sRate)

{

w <- 0

for(i in 1:n){

w <- max(0, w+rexp(1,sRate)-rexp(1,aRate))

}

w

}

#(b)

queueRep1 <- function (nReps, n, aRate, sRate)

{

wVec <- rep(NA, nReps)

for(j in 1:nReps)

wVec[j] <- queue2(n, aRate, sRate)

wVec

}

queueRep2 <- function (nReps, n, aRate, sRate)

{

sapply( rep(n,nReps), queue2, aRate, sRate )

}

#(c)

queueRep3 <- function (nReps, n, aRate, sRate)

{

w <- rep(0, nReps)

s <- matrix(rexp(n\*nReps, sRate), ncol=nReps)

a <- matrix(rexp(n\*nReps, aRate), ncol=nReps)

for(i in 1:n){

w <- pmax(0, w+s[i,]-a[i,])

}

w

}

#6. (a)

rwalk <- function(n)

{

c( 0, cumsum(sample( c(-1,1), n, replace=TRUE, prob=c(0.5,0.5))) )

}

#(b)

rwalkPos <- function(n)

{

rw <- cumsum(c(0, sample( c(-1,1), n, replace=TRUE, prob=c(0.5,0.5))))

sum( (rw[-(n+1)] + rw[-1]) > 0 )

}

#(c)

rwalkPos1 <- function(nReps, n)

{

results <- rep(NA, nReps)

for(i in 1:nReps)

results[i]<-rwalkPos(n)

results

}

rwalkPos2 <- function(nReps, n)

{

replicate( nReps, rwalkPos(n) )

}

#(d)

rwalkPos3 <- function(nReps, n)

{

stepWalks <- matrix( sample( c(-1,1), n, replace=TRUE, prob=c(0.5,0.5)), nr=nReps )

for(j in 2:n)

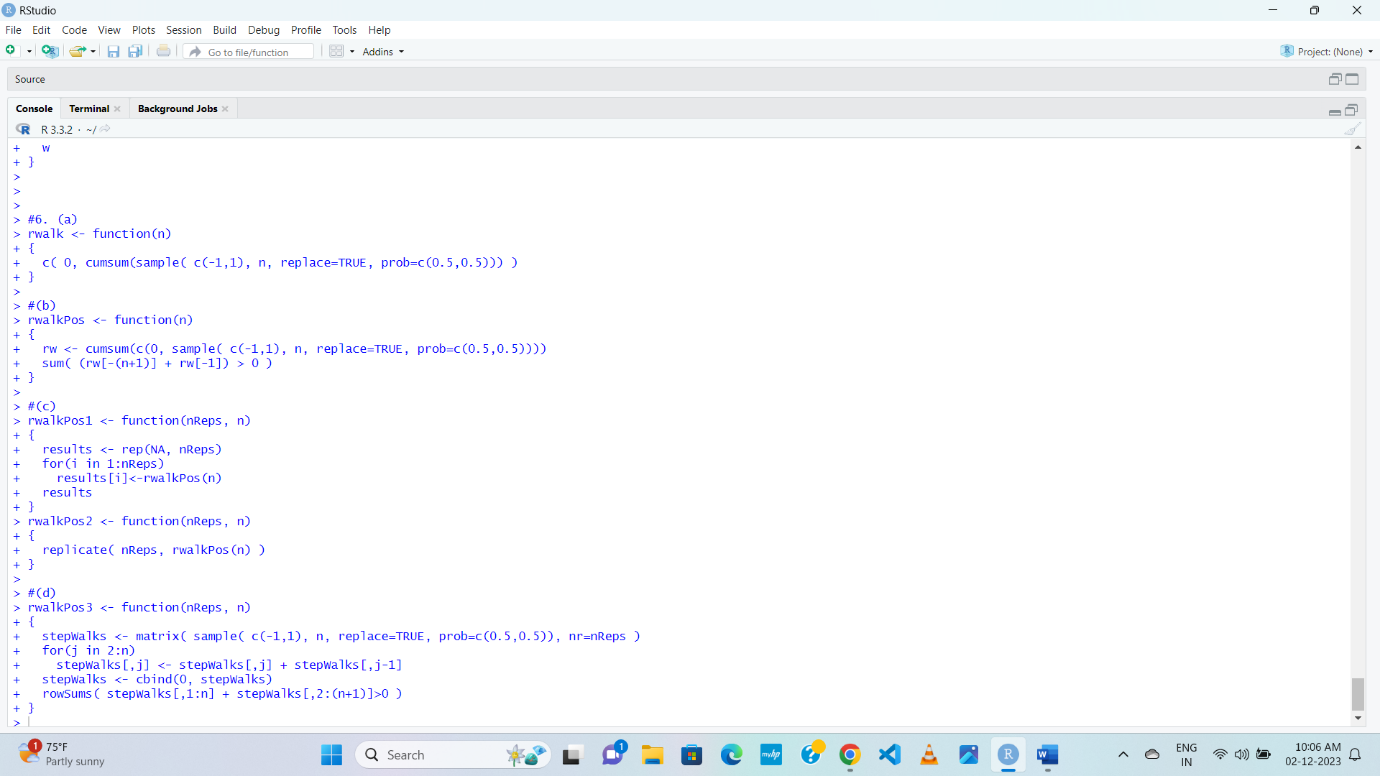
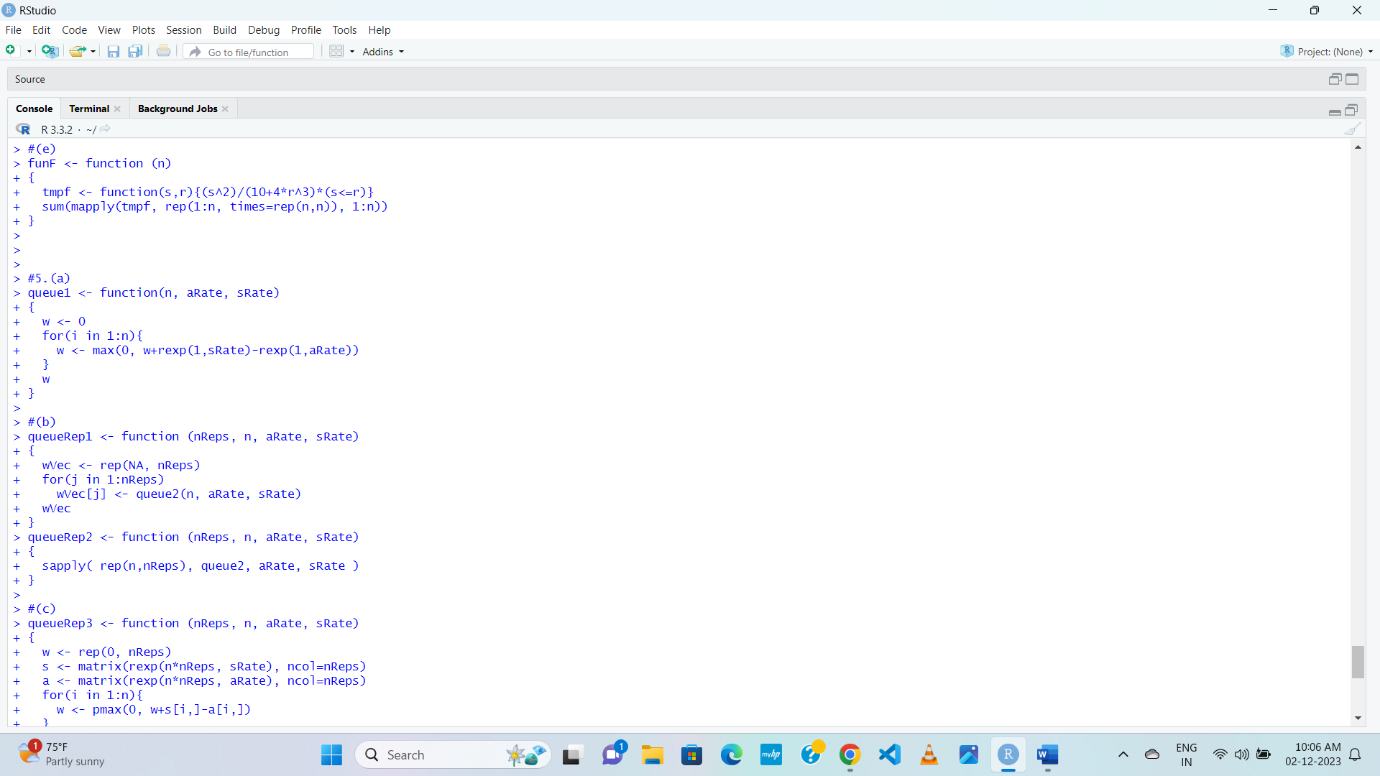
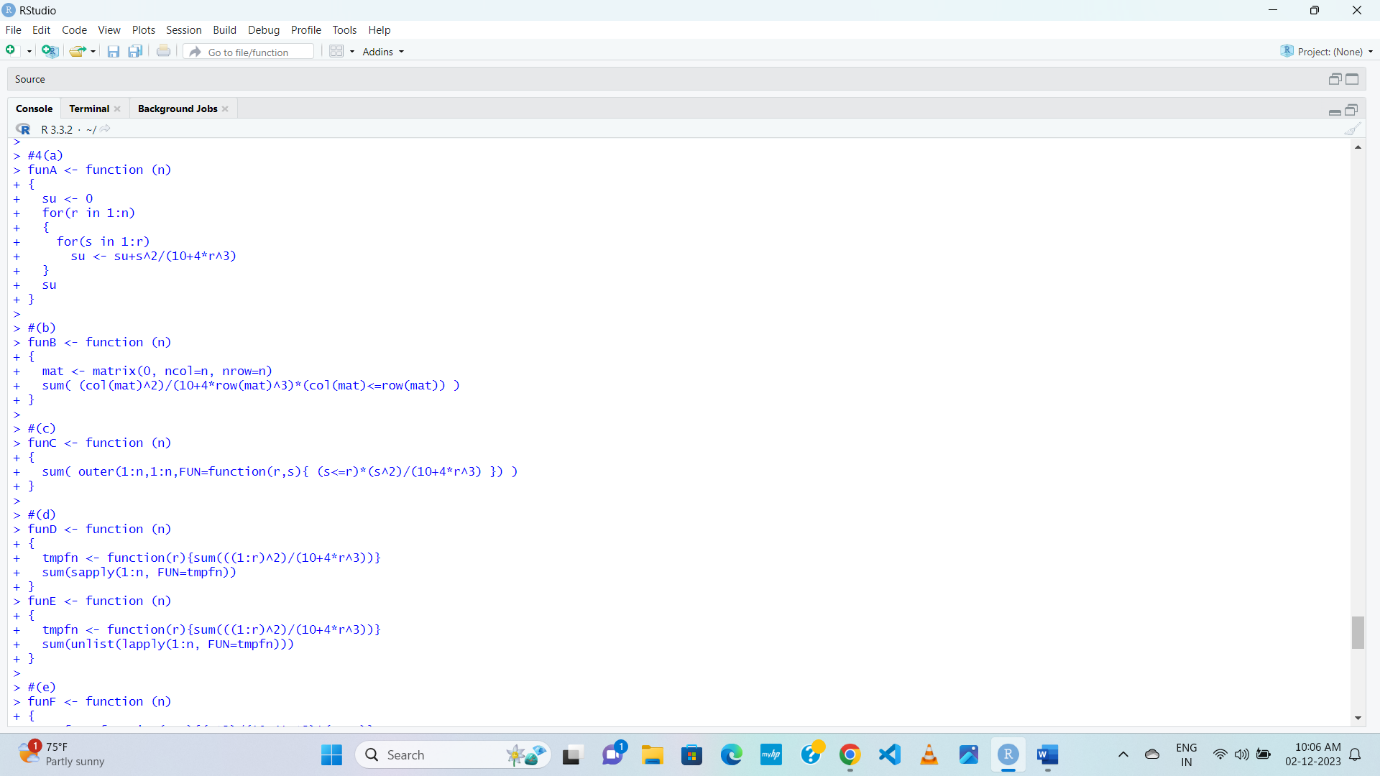
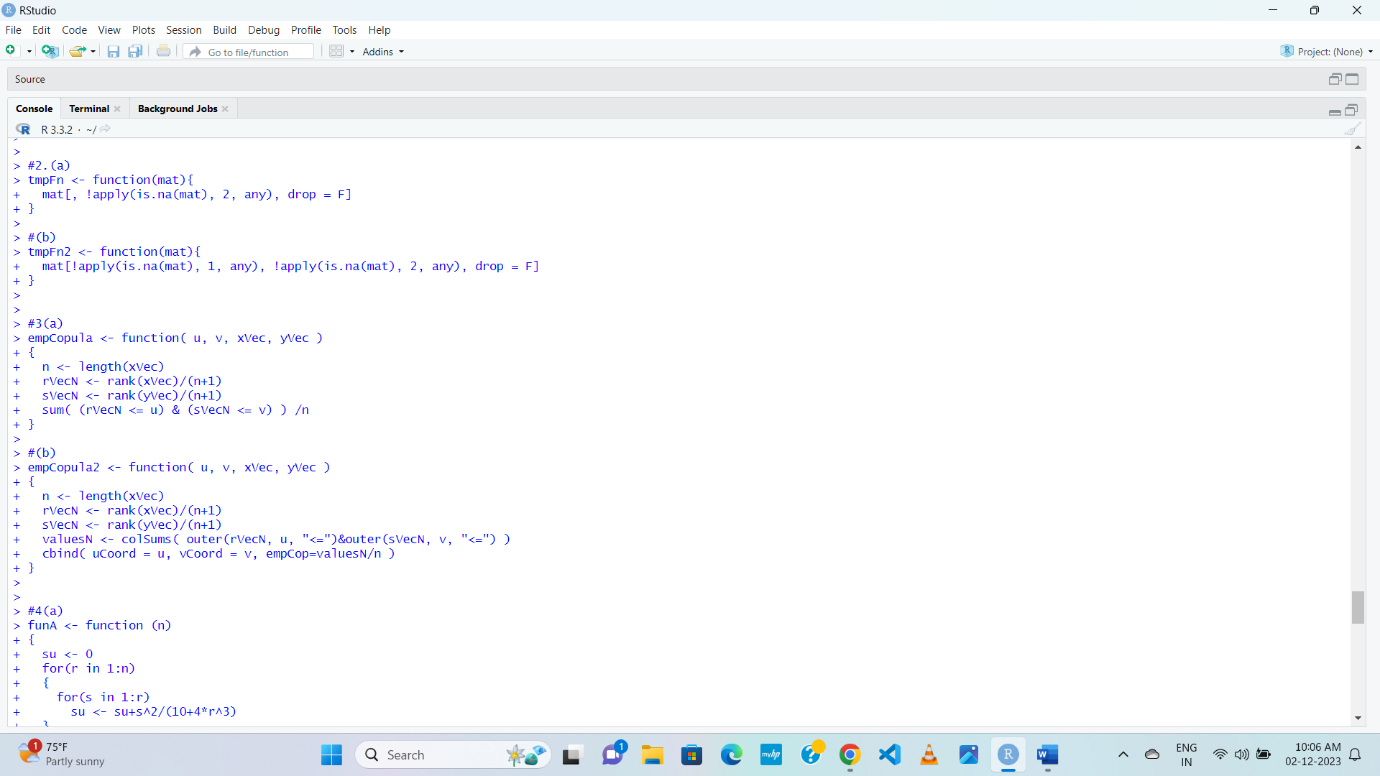
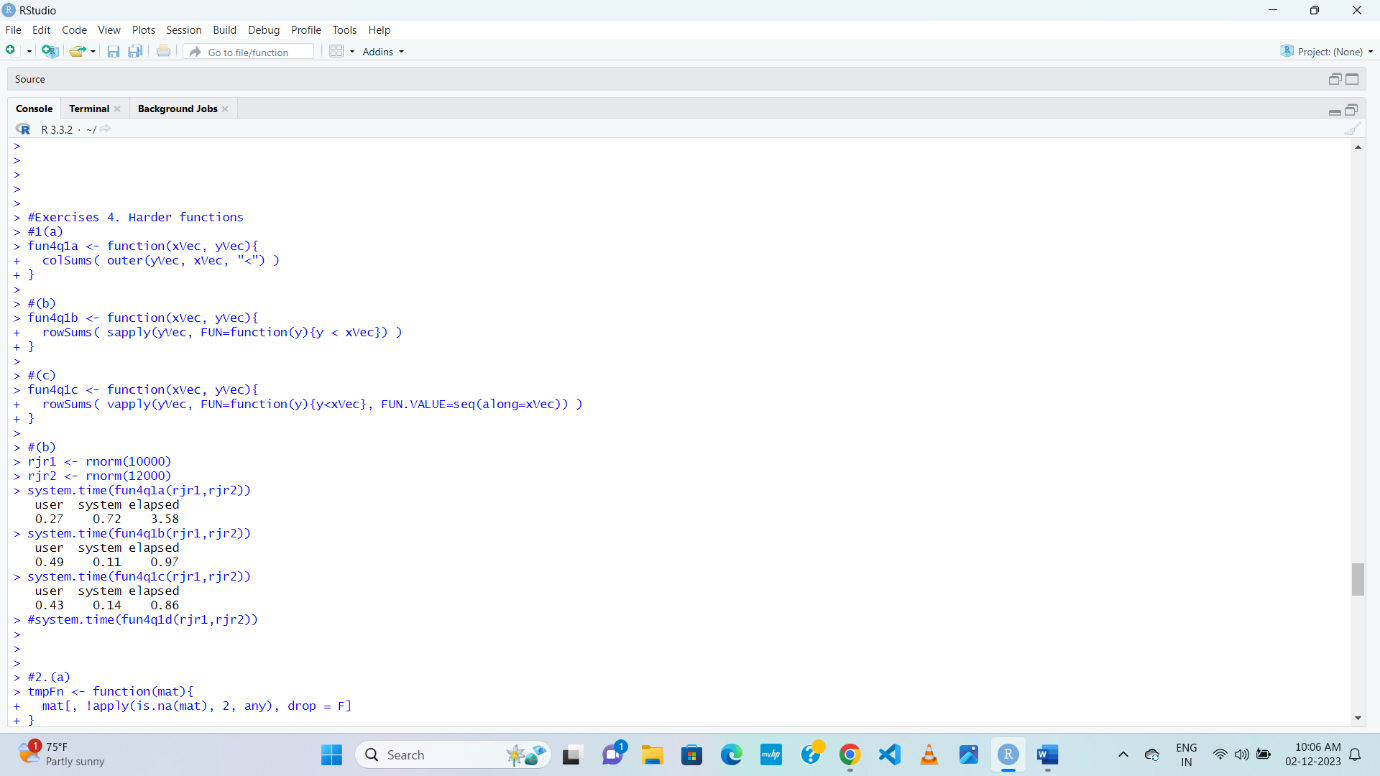
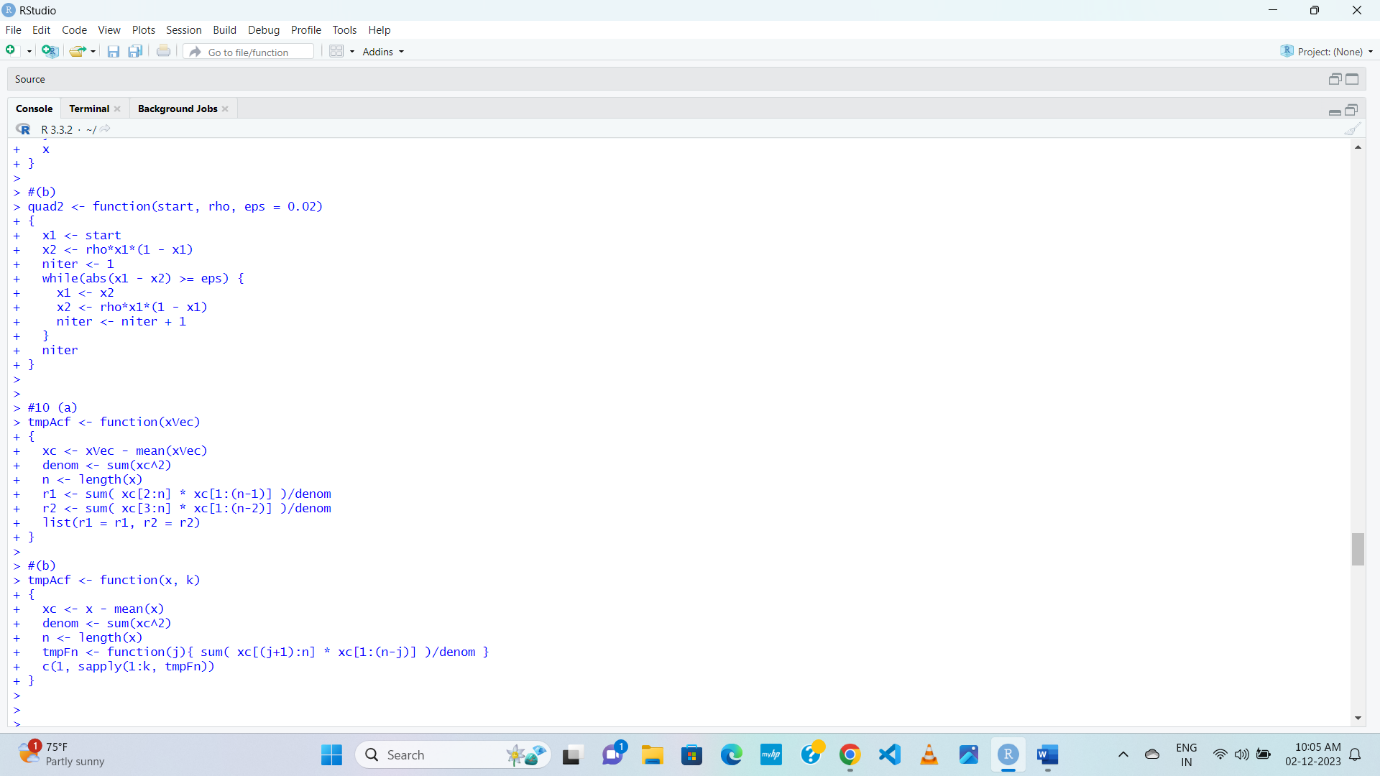
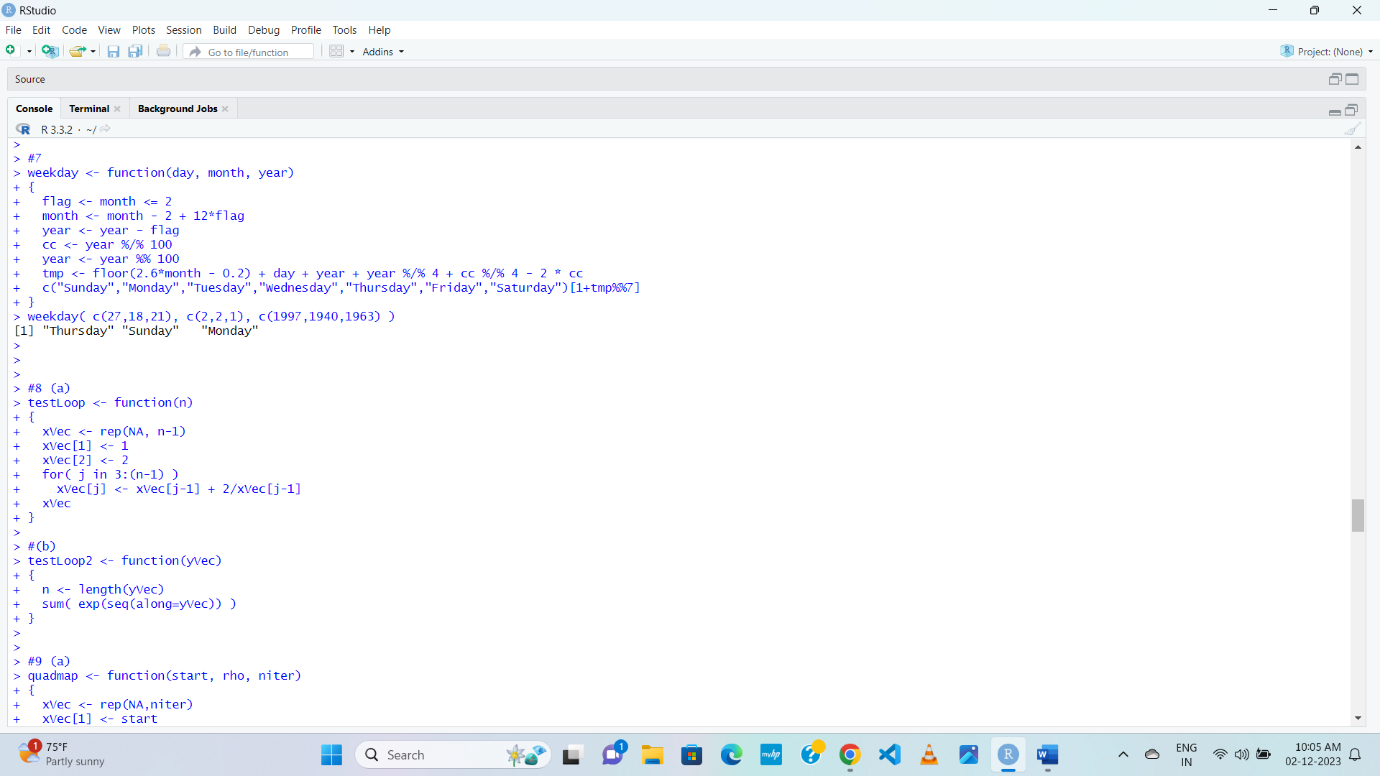
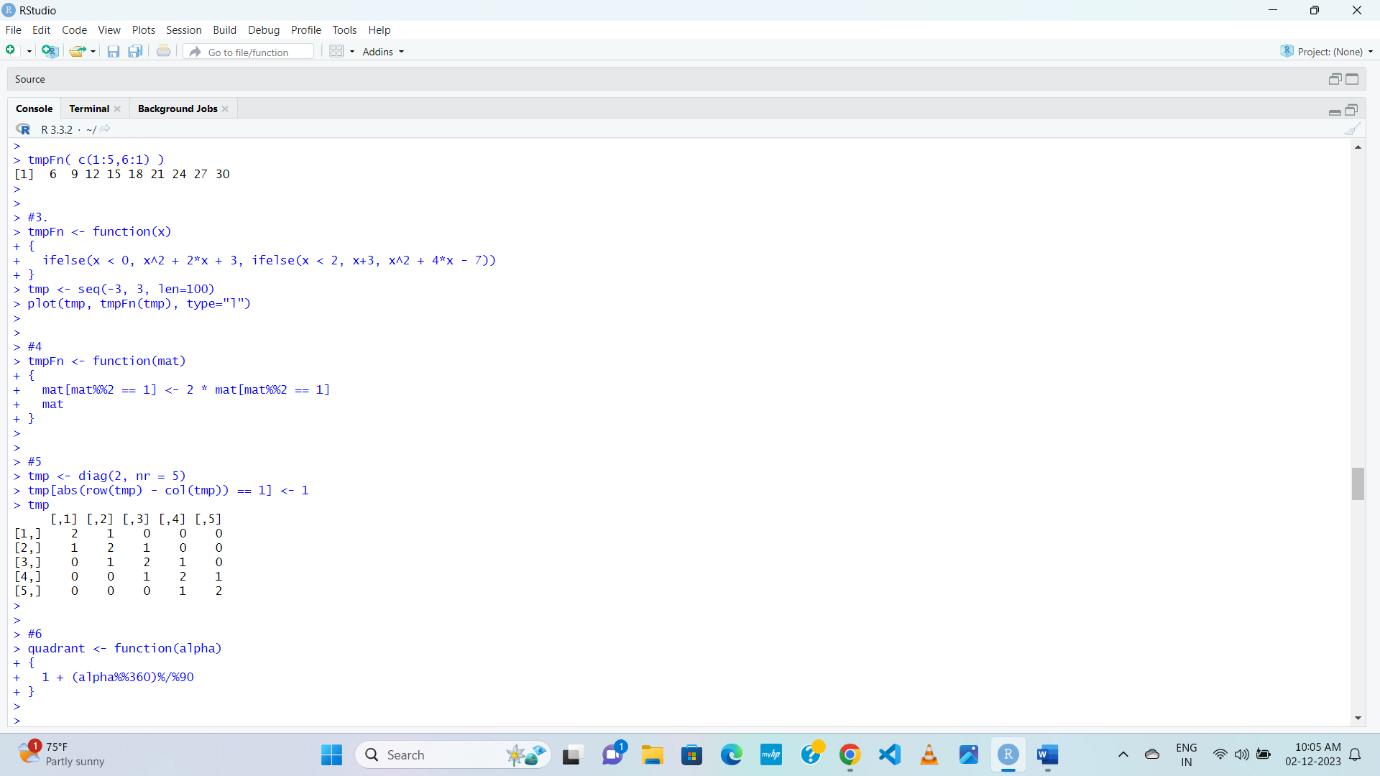
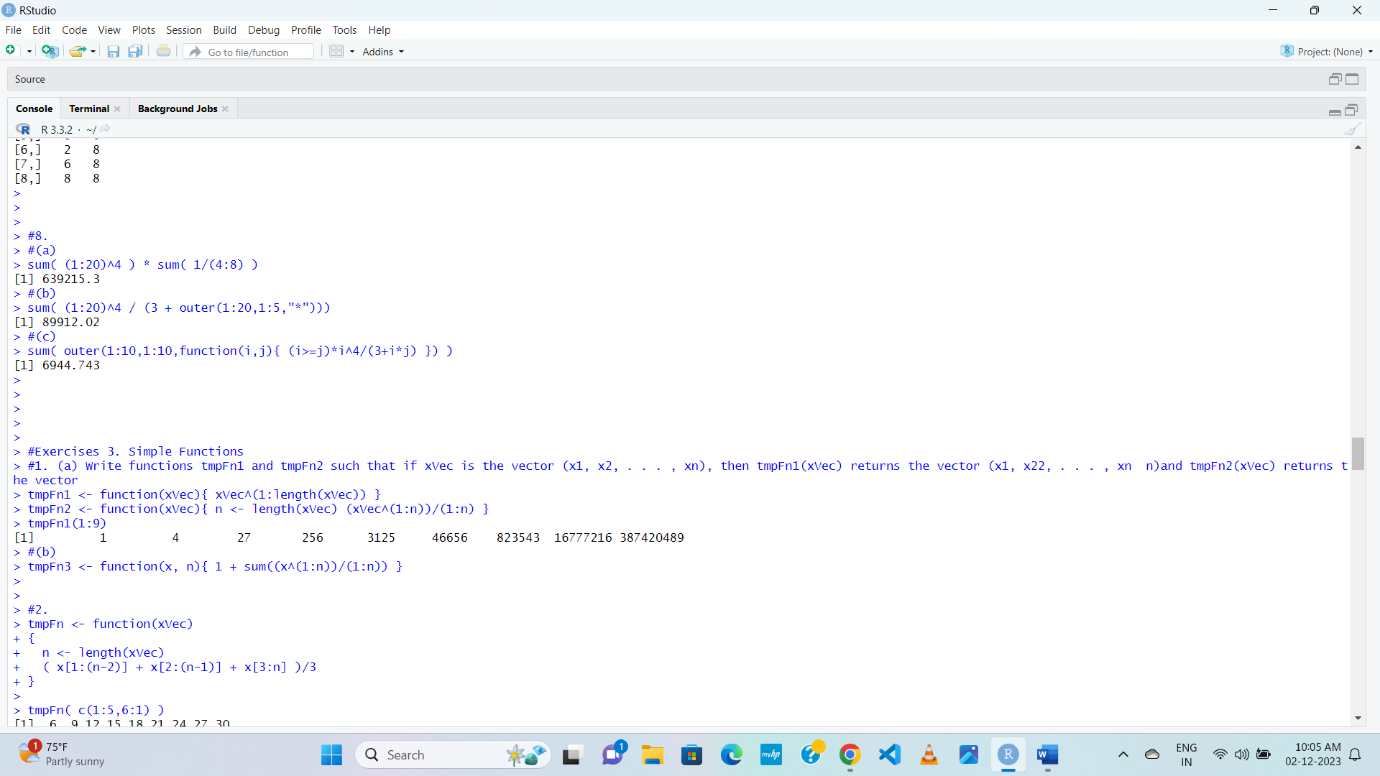
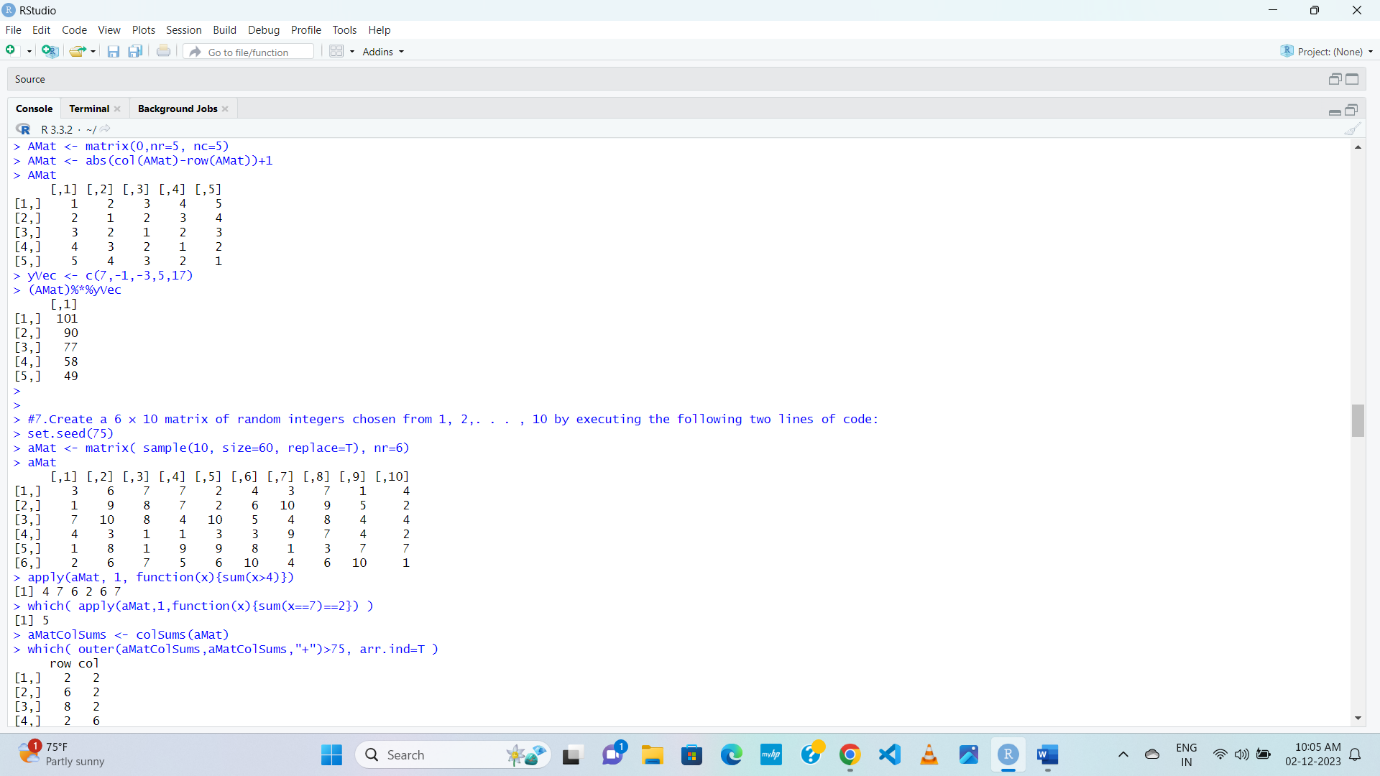
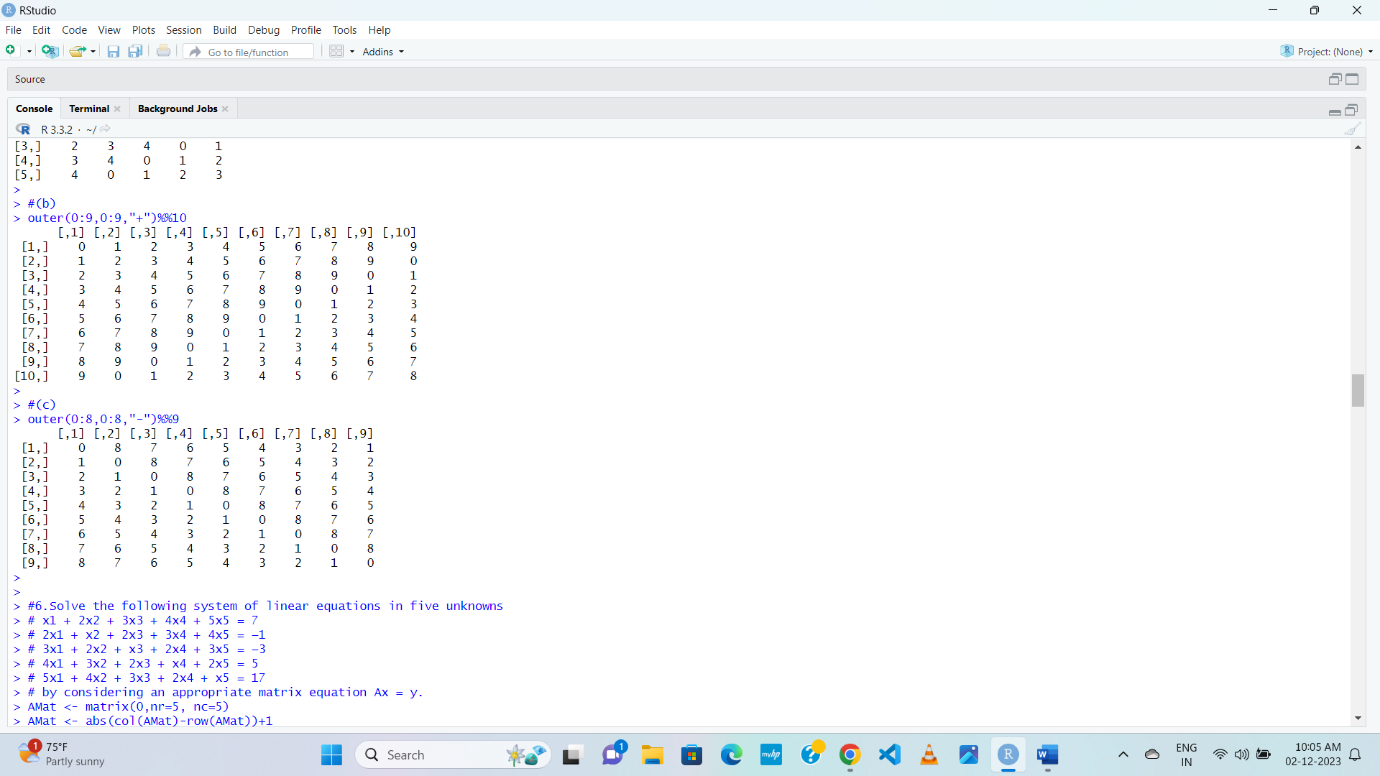
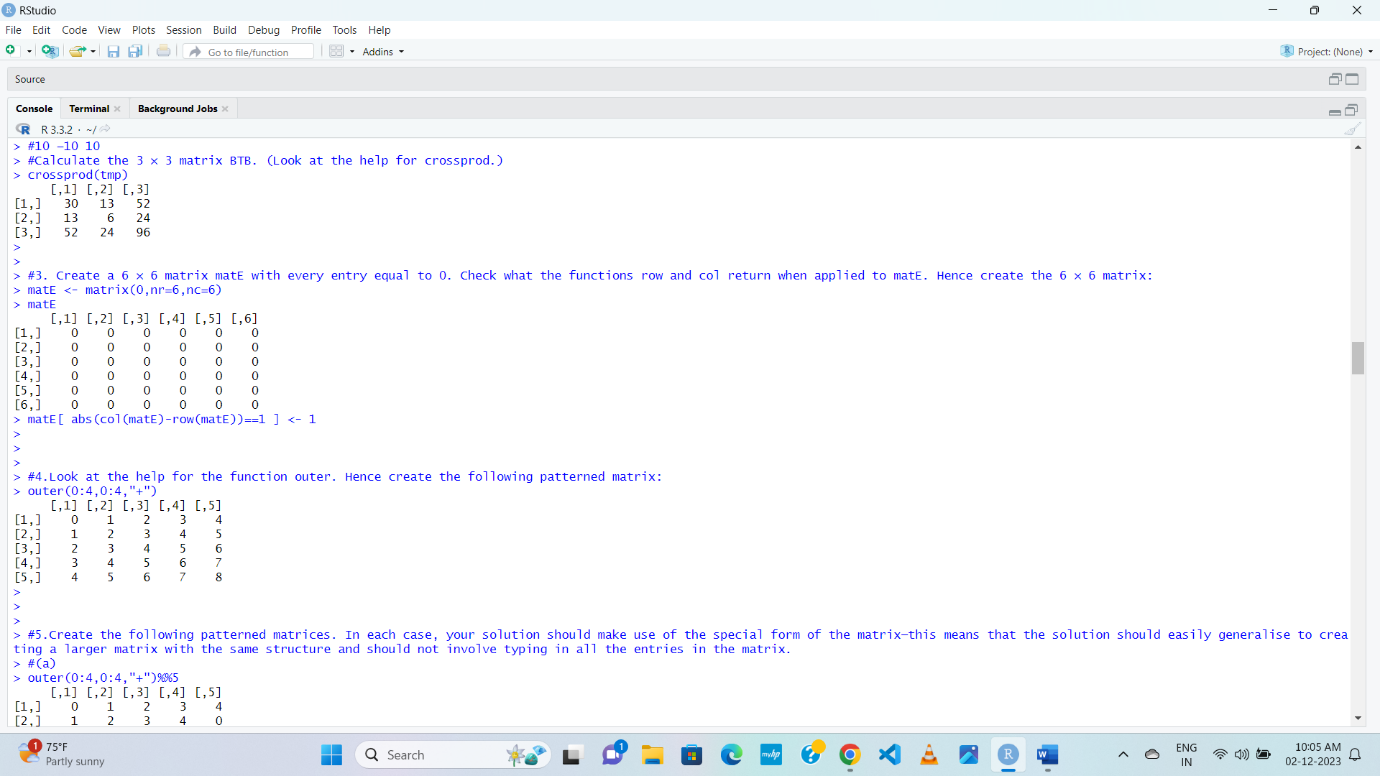
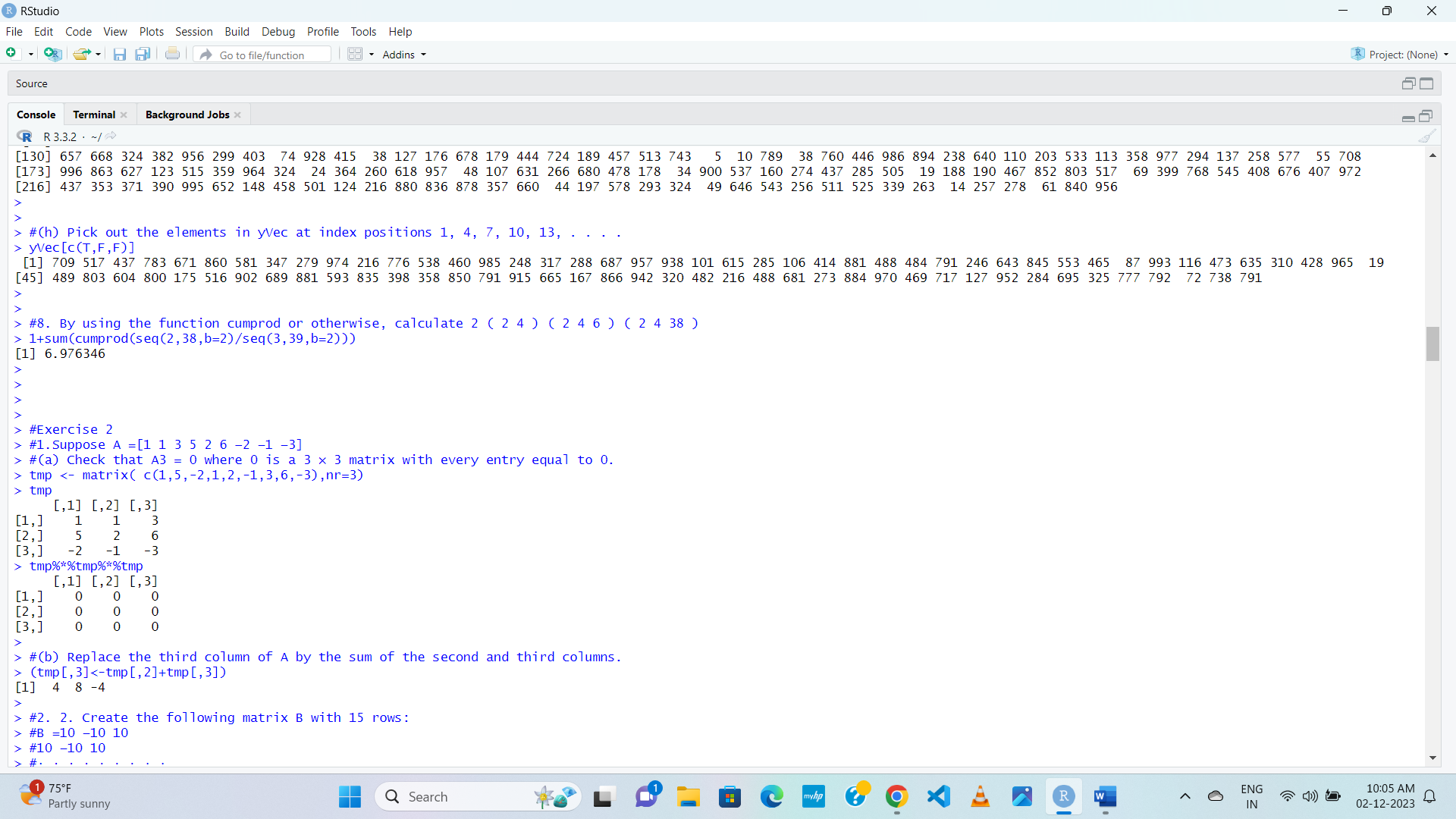
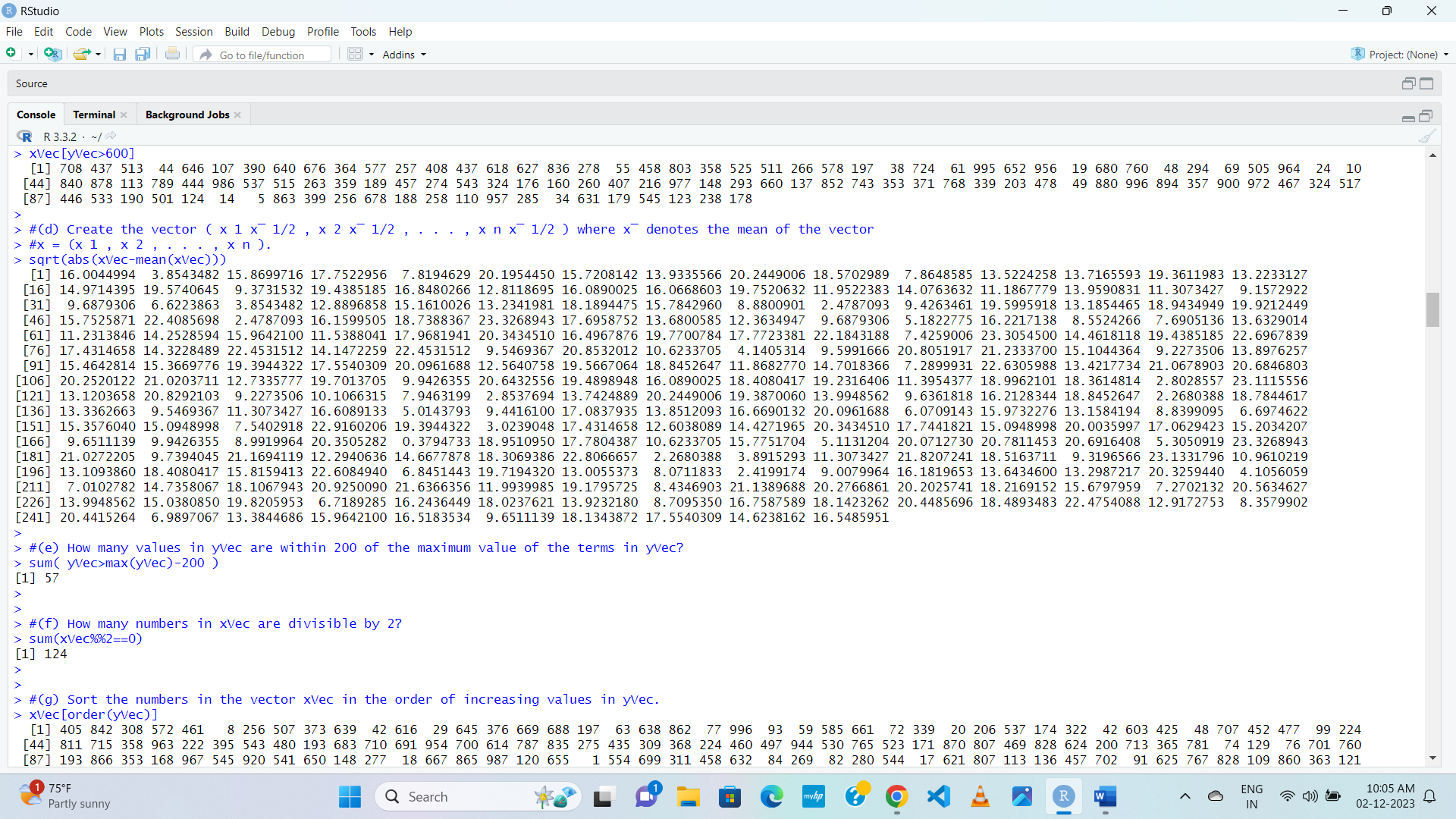
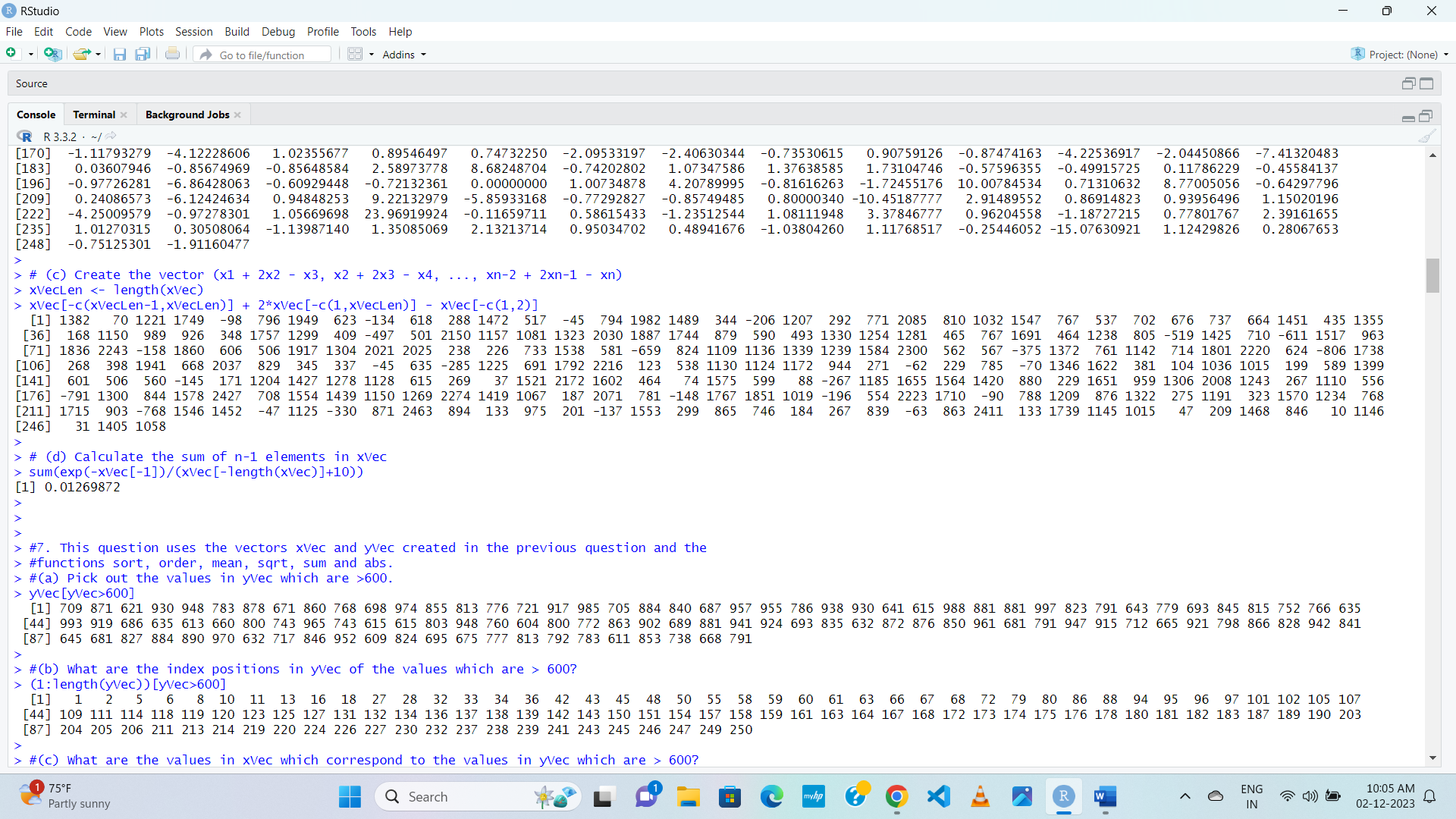
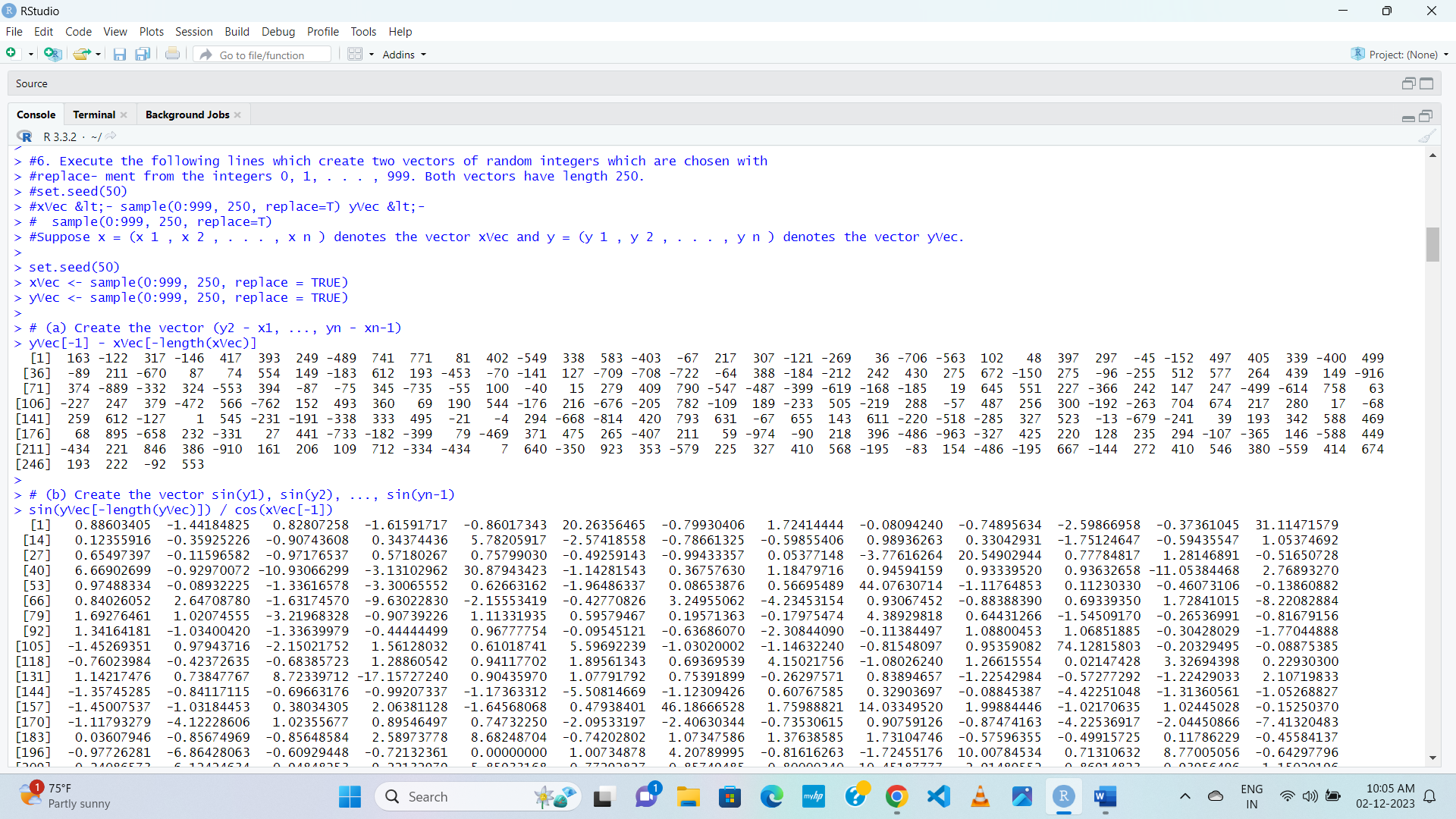
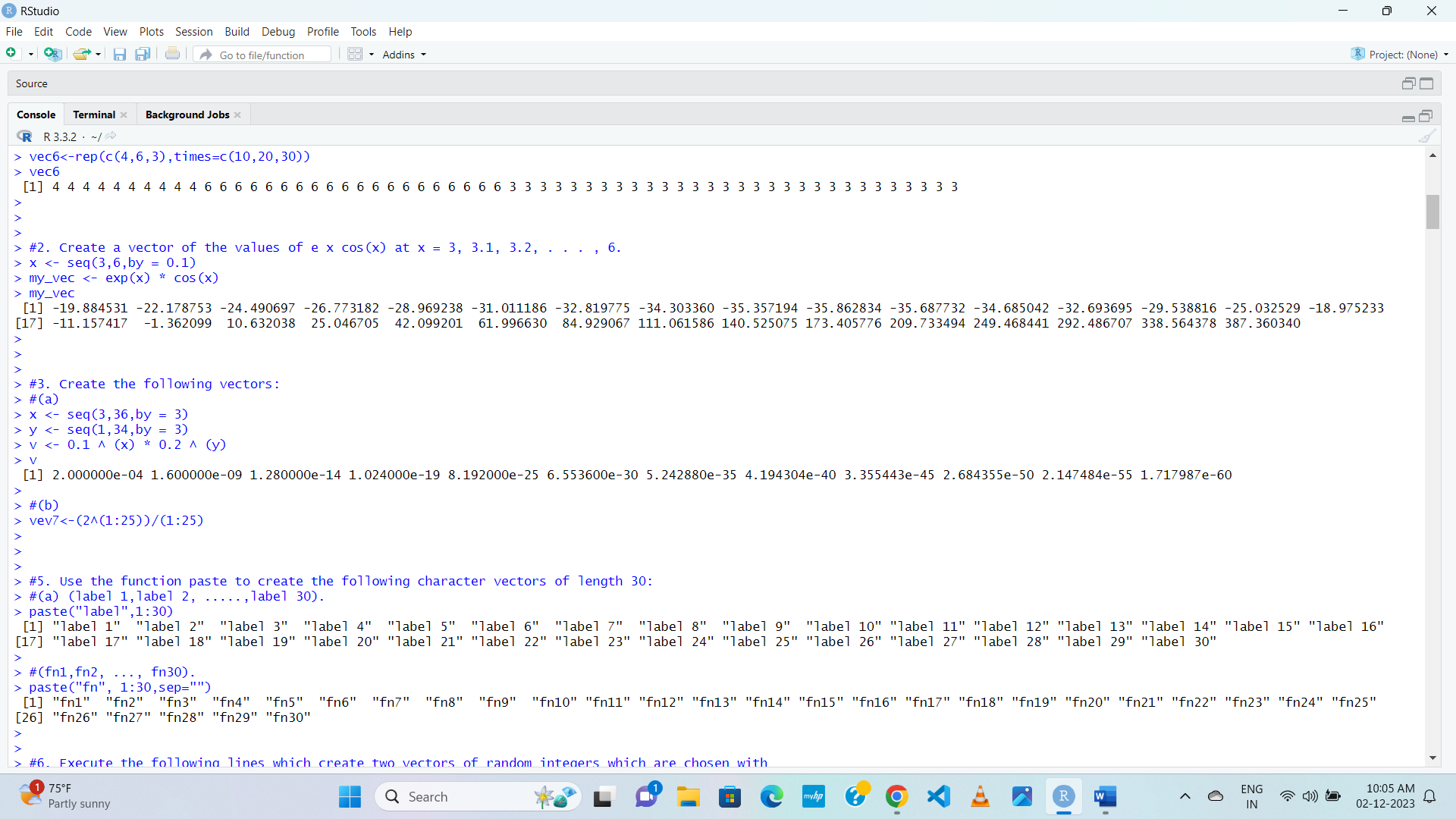
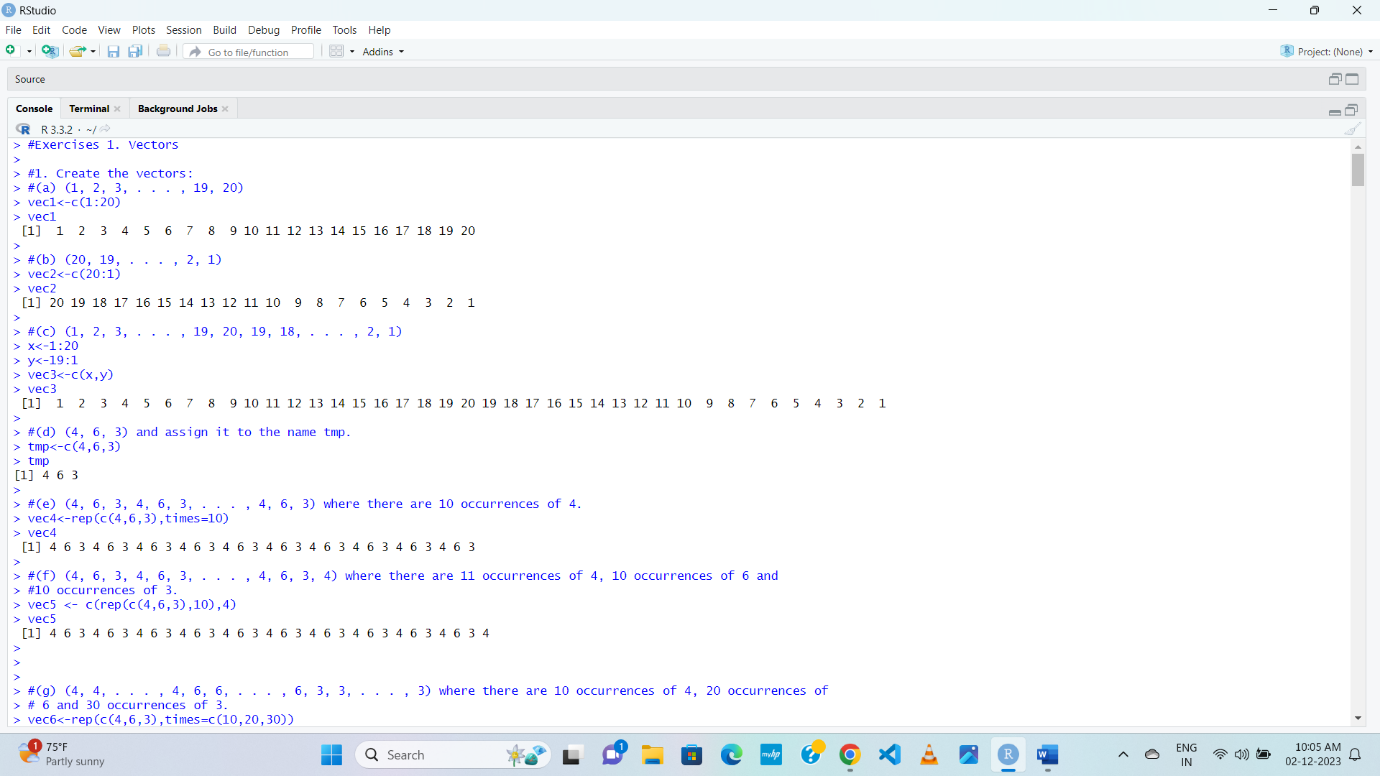
stepWalks[,j] <- stepWalks[,j] + stepWalks[,j-1]

stepWalks <- cbind(0, stepWalks)

rowSums( stepWalks[,1:n] + stepWalks[,2:(n+1)]>0 )

}

**output:**

\_