R Tutorial 2: Matrix and Vector Operations; Functions

Assignment Type	Non-assessed, Individual
Module	CSE413 Social Network Analysis

Overview

Notion

1. Bold **v** is a vector
2. Capitalised, bold **M** is a matrix
3. R code is either written in fixed-width text:
   `x <- c(1, 2, 3)`
4. or in a code box:
   '>' in the beginning indicates current line is an input
   it is **not** part of the code

```
> x <- c(1, 2, 3)     #input
> x                   #input
[1] 1 2 3             #output
```

Overview

1. Notion
2. Recap on Vector, List and Matrix
3. Vector and Matrix Operations
4. Create Your Functions in R
5. Exercise

Some basics

`class()` function to know the type of an object in R

```
> class(1)
[1] "numeric"
> class(class)
[1] "function"
> class(list(c(1,2,3)))
[1] "list"
```

Confused about a function?

`?` and `??` is your friend

try `?class` and `??class` in R or R Studio

Creating a Vector

`c()` function to concatenate objects to create a vector

```
> x <- c(0.5, 0.6)          ## numeric
> x <- c(TRUE, FALSE)       ## logical
> x <- c(T, F)              ## logical
> x <- c("a", "b", "c")     ## character
> x <- 9:29                 ## integer
> x <- c(1+0i, 2+4i)        ## complex
```

`vector()` function to initialize vectors of certain type

```
> x <- vector("numeric", length = 10)
0 0 0 0 0 0 0 0 0 0
```

Creating a Matrix

`matrix()` function to create a matrix of certain shape, e.g. 2 by 3 matrices are constructed by columns

```
> matrix(nrow = 2, ncol = 3)
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

`dim()` function to check the shape of the matrix

```
> dim(m)
[1] 2 3
```

From vector to matrix 1/2

Matrices can also be created directly from vectors by adding a dimension attribute.

```
> m <- 1:10
> m
[1] 1 2 3 4 5 6 7 8 9 10
> dim(m) <- c(2, 5)
> m
[,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
```

From vector to matrix 2/2

Matrices can be created by column-binding or row-binding with the `cbind()` and `rbind()` functions.

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
x y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
[,1] [,2] [,3]
x 1 2 3
y 10 11 12
```

List

Lists are a special type of vector that can contain elements of different classes

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1
[[2]]
[1] "a"
[[3]]
[1] TRUE
[[4]]
[1] 1+4i
```

From vector to list

`vector()` can also create a list

```
> x <- vector("list", length = 5)
> x
[[1]]
NULL
[[2]]
NULL
[[3]]
NULL
[[4]]
NULL
[[5]]
NULL
```

Overview

Addition of two vectors

$$\mathbf{x} = (1, 2, 3, 4) \quad \mathbf{y} = (6, 7, 8, 9)$$

$$\mathbf{z} = \mathbf{x} + \mathbf{y}$$

```
> x <- 1:4
> y <- 6:9
> z <- x + y
> z
[1] 7 9 11 13
```

Element-wise multiplication of two vectors

$\mathbf{x} \circ \mathbf{y}$ or sometimes $\mathbf{x} \odot \mathbf{y}$

```
> x <- 1:4
> y <- 6:9
> x * y
[1]  6 14 24 36
```

Element-wise division of two vectors

$$\mathbf{x} \oslash \mathbf{y}$$

```
> x <- 1:4
> y <- 6:9
> x / y
[1] 0.1666667 0.2857143 0.3750000 0.4444444
```

Element-wise matrix multiplication (Hadamard product)

$$\mathbf{X} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix}$$

```
> x <- matrix(1:6, 2, 3)
> y <- matrix(rep(10, 6), 2, 3)
```

$$\mathbf{X} \circ \mathbf{Y}$$

```
> x * y
[,1] [,2] [,3]
[1,]   10   30   50
[2,]   20   40   60
```

Element-wise matrix division

$$\mathbf{X} \oslash \mathbf{Y}$$

```
> x / y
[,1] [,2] [,3]
[1,]   0.1   0.3   0.5
[2,]   0.2   0.4   0.6
```

True matrix multiplication (dot product)

Since the number of columns in **X** does not match the number of rows in **Y**, we need to transpose one of the two matrices.

$$\mathbf{X}^T\mathbf{Y} \text{ or sometimes } \mathbf{X}^T \cdot \mathbf{Y}$$

```
> t(x) %*% y
[,1] [,2] [,3]
[1,]   30   30   30
[2,]   70   70   70
[3,]  110  110  110
```

$$\mathbf{X}\mathbf{Y}^T \text{ or sometimes } \mathbf{X} \cdot \mathbf{Y}^T$$

```
> x %*% t(y)
[,1] [,2]
[1,]   90   90
[2,]  120  120
```

Matrix crossproduct

$$\mathbf{X} \times \mathbf{Y} = \mathbf{X}^T\mathbf{Y}$$

```
> crossprod(x, y)
[,1] [,2] [,3]
[1,]   30   30   30
[2,]   70   70   70
[3,]  110  110  110
```

and R also has a `tcrossprod()` so we can do $\mathbf{XY}^T$

```
> tcrossprod(x, y)
[,1] [,2]
[1,]   90   90
[2,]  120  120
```

Other Matrix Operators

try `diag`, `upper.tri` and `lower.tri`

```
> diag(1,3)
[,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> matrix(1, 3,3) - diag(1,3)
[,1] [,2] [,3]
[1,]    0    1    1
[2,]    1    0    1
[3,]    1    1    0
```

Getting elements

Sometimes you may need to access an element, a row, a column or even a sub-matrix of a matrix. Indexing in R enables that ability. To index a vector, a list or a matrix, use square brackets []

```
> x
[,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x[2,3]
[1] 6
```

Getting a scalar

5th element of $(1, 2, \cdots, 9)$

```
> (1:9)[5]
[1] 5
```

Element at row 2, column 3 of $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$ as a scalar

```
> matrix(1:9, 3,3)[2,3]
[1] 8
```

Getting a vector

To get a vector form a vector or a matrix, use a vector as the index

5th-9th elements of $(1, 2, \cdots, 9)$ as a vector

```
> (1:9)[5:9]
[1]  5  6  7  8  9
```

2nd row of $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$ as a vector

```
> matrix(1:9, 3,3)[2,c(1,2,3)]
[1]  2  5  8
```

or simply

```
> matrix(1:9, 3,3)[2,]
[1]  2  5  8
```

Getting a sub-matrix

Similarly, to get a sub-matrix consisted of the 2nd and 3rd row, and first three columns of $\begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$

```
> matrix(1:12, 3,4)[c(2,3), 1:3]
[,1] [,2] [,3]
[1,]    2    5    8
[2,]    3    6    9
```

Note: the order of the index is important

```
> matrix(1:12, 3,4)[c(3,2), 1:3]
[,1] [,2] [,3]
[1,]    3    6    9
[2,]    2    5    8
```

More on subsetting 1/3

R support even more complex yet handy ways of getting the elements (aka subsetting) based on conditions
for instance, to get elements that are greater than 5 from $(1, 2, ..., 9)$
First, get a boolean vector stating which elements satisfies the condition (> 5)

```
> a <- (1:9)
> a > 5
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
   TRUE  TRUE
```

More on subsetting 2/3

Then use this boolean vector to index the original vector to get the elements

```
> a[ a > 5 ]
[1] 6 7 8 9
```

The same applies to matrices, differences exist though

More on subsetting 3/3

Say, to get elements that are greater than 5 in $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$

```
> m <- matrix(1:9, 3, 3)
> m > 5
[,1]   [,2]  [,3]
[1,] FALSE FALSE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE  TRUE TRUE
```

The boolean vector is of the same shape as `m`, but after subsetting we have a vector, of course it has to be a vector

```
> m [m > 5]
[1] 6 7 8 9
```

Overview

Functions in R

Functions in R are "first class objects", which means that they can be treated much like any other R object. Importantly,

- Functions can be passed as arguments to other functions. This is very handy for the various apply funtions, like `lapply()` and `sapply()`.
- Functions can be nested, so that you can define a function inside of another function

Function without arguments or return value

`function()` to create a simple function

```
> meow <- function()
+ cat("Meow Meow Meow")
> meow()
Meow Meow Meow
```

Note: '+' in the beginning indicates the input is unfinished and has wrapped to the next line; it is also **not** part of the code

Function with single arguments but without return value

When the function has more than one line of code, we need to use curly brackets '{' and '}' to wrap the code

```
> meow <- function(n_meows){
+     for (i in seq_len(n_meows) ){
+         cat("Meow\n")
+     }
+ }
> meow(3)
Meow
Meow
Meow
```

Function with multiple arguments and a return value

We want our function to return something, just like `class(a)` returns the class of variable a, this is achieved by adding the desired return value to the last line in the function. Say we'd like to find 2 to the power of 3...

```
> kittypower <- function(n, m){
+       cat("Kitty says")
+       p = n ** m
+       p
+ }
> kittypower(2, 3)
Kitty says[1] 8
```

Overview

Vector Exercise 1/3

Create the following vectors in R:

1. (1, 2, 3, ..., 20)
2. (1, 2, 3, ..., 19, 20, 19, ..., 3, 2, 1)
3. assign (4, 6, 3) to variable `tmp`
4. (4, 6, 3, 4, 6, 3,..., 4, 6, 3) where there are 10 occurrences of 4
   hint: repeat `tmp` via `rep`
5. (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
   hint: `?rep`

Vector Exercise 2/3

1. Create a vector of the values of $e^x cos(x)$ at $x = 3, 3.1, 3.2, ..., 6$.
   hint: `?seq`
2. Calculate $\sum_{i=10}^{100}(i^3 + 4i^2)$
3. Use the function `paste` to create the following character vectors of length 30:
   (a) ("label 1", "label 2", ..., "label 30").
   Note that there is a single space between label and the number following.
   (b) ("fn1", "fn2", ..., "fn30").
   In this case, there is no space between fn and the number following.

Vector Exercise 3/3

Following lines create two vectors of random integers which are chosen with replacement from the integers $[0, 999]$. Both vectors have length 250.

```
set.seed(50)
xVec <- sample(0:999, 250, replace=T)
yVec <- sample(0:999, 250, replace=T)
```

1. Pick out the values in `yVec` which are ≥ 600.
2. What are the index positions in `yVec` of the values which are ≥ 600?

Vector Exercise Solution 1/3

```
1:20
```

```
c(1:20,19:1)
```

```
tmp <- c(4,6,3)
```

```
rep(tmp,10)
```

```
rep(tmp,times=c(10,20,30))
```

Vector Exercise Solution 2/3

```
tmp <- seq(3,6,by=0.1)
exp(tmp)*cos(tmp)
```

```
tmp <- 10:100
sum(tmp^3+4*tmp^2)
```

```
(a) paste("label", 1:30)
(b) paste("fn", 1:30,sep="")
```

Vector Exercise Solution 3/3

```
yVec[yVec >= 600]
```

```
(1:length(yVec))[yVec >= 600]
#or
which(yVec >= 600)
```

Matrix Exercise 1/2

Given

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

1. Check whether $\mathbf{A}^3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.
2. Replace the 3rd column of $\mathbf{A}$ by the sum of the 2nd and 3rd columns.

Matrix Exercise 2/2

Given

$$\mathbf{B} = \begin{bmatrix} 10 & -10 & 10 \\ 10 & -10 & 10 \\ \dots & & \\ 10 & -10 & 10 \end{bmatrix}$$

1. Create matrix **B** with 15 rows
2. Calculate the 3×3 matrix $\mathbf{B}^T\mathbf{B}$
   hint: `?crossprod`

Matrix Exercise Solution 1/2

```
tmp <- matrix( c(1,5,-2,1,2,-1,3,6,-3), nr=3)
tmp %*% tmp %*% tmp == matrix(0, 3, 3)
```

```
tmp[,3] <- tmp[,2] + tmp[,3]
```

Matrix Exercise Solution 2/2

```
tmp <- matrix(c(10,-10,10), b=T, nc=3, nr=15)
```

```
t(tmp)%*%tmp
#or
crossprod(tmp)
```

Function Exercise 1/2

Suppose `xVec` is a vector $(x_1, x_2, \dots, x_n)$

1. Write a function `tmpFn1` that returns the vector $(x_1, x_2^2, \dots, x_n^n)$
2. Write a function `tmpFn2` that returns the vector $(x_1, \frac{x_2^2}{2}, \dots, \frac{x_n^n}{n})$

Function Exercise 2/2

Suppose `xVec` is a vector $(x_1, x_2, ..., x_n)$ Write a function `tmpFn(xVec)` that returns the vector of moving averages:

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

Try out your function; for example, try `tmpFn( c(1:5,6:1) )`

Function Exercise Solution 1/2

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

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

Function Exercise Solution 2/2

`tmpFn( c(1:5,6:1) )` returns vector (2, 3, 4, 5, 5.333, 5, 4, 3, 2)

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

Alternatively,

```
tmpFn <- function(xVec) {
    n <- length(xVec)
    (xVec[-c(n-1,n)] + xVec[-c(1,n)] +
                                   xVec[-c(1,2)])/3
}
```

The End