Coding and Mathematical Definitions

Brian D. Gerber

December 10, 2024

Definitions of R Code

R script

the lines of code and comments that you are writing (filename.R).

R project

your R script, any variables you have created, and your current R environment (filename.Rproj).

R package

a set of functions/code that you can load into your script (examples - dplyr, sp).

R Comment: does not run as code, starts with #. You use comments to explain what your code is doing in plain langauge.

Object or Variable

a unit of information that is stored in the workspace (computer memory) and can be recalled or manipulated. 'a' is an object. Specifically, a vector of length 3.

```
a = c(3, 2, 1)
```

Element

a piece of information within an object. The 1st element of a is...

```
a[1]
```

[1] 3

Function

a command to take inputs (objects or elements) and manipulates it to provide an output, which can be saved as a new object.

```r
fun.text = function(x) {
    paste("Your input variable is ", x, sep = "")
}

fun.text(4)
```

[1] "Your input variable is 4"

```r
fun.text("INPUT")
```

[1] "Your input variable is INPUT"

Argument

a specific command within a function. Many functional arguments are preset and do not need to be explitly stated.

```r
x = c(1, 2, 3, NA)
mean(x)
```

[1] NA

```r
# na.rm is an argument of the function mean to ignore the missing value. Its
# preset is FALSE. If you have na's you need to change the argument to 'TRUE'.
mean(x, na.rm = TRUE)
```

[1] 2

Types of R Objects and Mathematical Notation

Vector

1 row, many columns OR 1 column many rows. Can be numbers of characters.

```r
v = c(4, 1, 3)
is.vector(v)
```

[1] TRUE

```r
length(v)
```

[1] 3

Math notation (capitalized & lower case): $\vec{v}$ or $\mathbf{v}$

$$\mathbf{v} = \begin{bmatrix} 4 & 1 & 3 \end{bmatrix} \tag{1}$$

Matrix

Generalization of vectors. Can have 1 or more rows and columns. Only numbers.

```r
M = matrix(1:10, nrow = 2)
M
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```r
dim(M)
```

```
## [1] 2 5
```

```r
is.vector(M)
```

```
## [1] FALSE
```

```r
is.matrix(M)
```

```
## [1] TRUE
```

Math notation (capitalized & upper case): $\mathbf{M}$

$$\mathbf{M} = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 \\ 2 & 4 & 6 & 8 & 10 \end{bmatrix} \tag{2}$$

Array

Generalization of matrices. Can be n dimensional. Only numbers.

```r
arr = array(1:100, dim = c(3, 4, 3))
arr
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]   25   28   31   34
## [2,]   26   29   32   35
## [3,]   27   30   33   36
```

```r
dim(arr)
```

```
## [1] 3 4 3
```

```r
is.vector(arr)
```

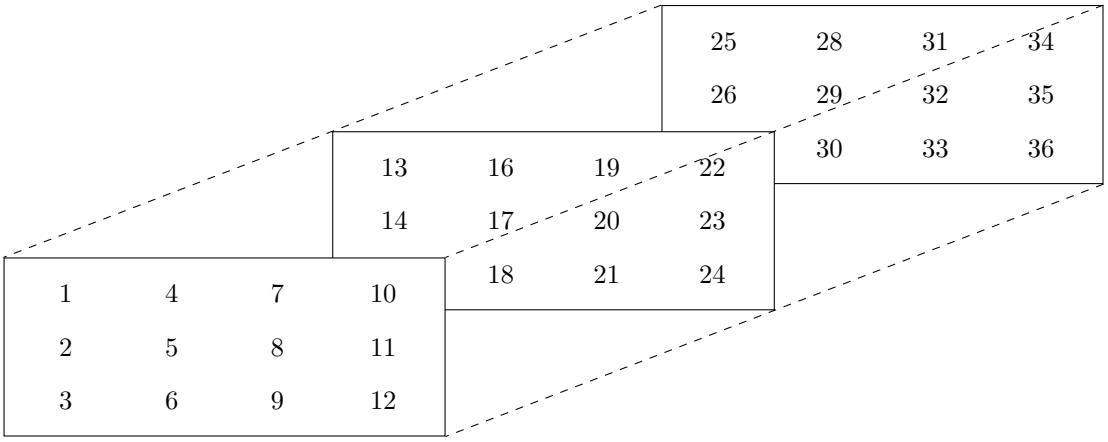
```
## [1] FALSE
```

```r
is.matrix(arr)
```

```
## [1] FALSE
```

```r
is.array(arr)
```

```
## [1] TRUE
```



List

Can store any type of objects together.

```r
list1 = vector("list", 2)
list1[[1]] = v
list1[[2]] = M
list1[[3]] = arr

is.list(list1)
```

```
## [1] TRUE
```

```r
length(list1)
```

```
## [1] 3
```

```r
list1[[1]]
```

```
## [1] 4 1 3
```

Important Functions

For Loop

To do a task many times. Simple, but inefficient

```r
#First, create a large matrix
x <- matrix(rnorm(400*4000), ncol=400)

#Second, create a vector to store results
mx <- rep(NA, nrow(x))

#Third, for each row of the matrix, find the maximum value and store it in mx using
#a loop. We will do this from index 1 to the max number of rows (nrow(x)). We will
#iterate using index i.

for(i in 1:nrow(x)){
  mx[i] <- max(x[i,])
}
```

ifelse

used to return an output based on specified conditions

```r
# Syntax:
If (test_expression) {
  Statement
}
```

```r
# Example:
a = c(5,7,2,9)
ifelse(a %% 2 == 0,"even","odd")
```

```
## [1] "odd"  "odd"  "even" "odd"
```

```r
#OR

x <- 5
if(x > 0){
  print("Positive number")
}
```

```
## [1] "Positive number"
```

apply

To do a task many times using vectorization. Much faster then a for loop.

```r
x <- matrix(rnorm(400*4000), ncol=400)

# The 1 indicates to the function 'max' on the rows of the object 'x'. A 2 would
# make this function should be applied to each column of the object 'x'.
mx2 <- apply(x, 1, max)
```

Another Example

```r
dataTable <- cbind(x1 = 1, x2 = (c(4:0, 2:6)))
dataTable
```

```
##       x1 x2
##  [1,]  1  4
##  [2,]  1  3
##  [3,]  1  2
```

```
##  [4,]  1  1
##  [5,]  1  0
##  [6,]  1  2
##  [7,]  1  3
##  [8,]  1  4
##  [9,]  1  5
## [10,]  1  6
```

```
# Sum each of the rows of the data table
apply(dataTable, MARGIN = 1, FUN = sum)
```

```
##  [1] 5 4 3 2 1 3 4 5 6 7
```

```
# Now sum the columns
apply(dataTable, MARGIN = 2, FUN = sum)
```

```
## x1 x2
## 10 30
```

lapply

To do a task many times using vectorization for each list element.

```
list1 <- list(1:20, 1:5, 1:100)
listOutput <- lapply(list1, FUN = quantile, probs = c(0.025, 0.975))
listOutput
```

```
## [[1]]
##   2.5%  97.5%
##  1.475 19.525
##
## [[2]]
##  2.5% 97.5%
##   1.1   4.9
##
## [[3]]
##   2.5%  97.5%
##  3.475 97.525
```

```
is.list(listOutput)
```

```
## [1] TRUE
```

sapply

takes lists, vectors or data frames as input and gives output in vector or matrix.

```
listOutput2 <- sapply(list1, FUN = quantile, probs = c(0.025, 0.975))
listOutput2
```

```
##         [,1] [,2]   [,3]
## 2.5%   1.475  1.1  3.475
## 97.5% 19.525  4.9 97.525
```

```
is.list(listOutput2)
```

```
## [1] FALSE
```

```
is.matrix(listOutput2)
```

```
## [1] TRUE
```

mapply

allows you to apply a function to elements of a matrix.

```
l1 <- list(a = c(1:10), b = c(11:20))
l2 <- list(c = c(80:89), d = c(90:99))

# sum the corresponding elements of l1 and l2
mapply(sum, l1$a, l1$b, l2$c, l2$d)
```

```
##  [1] 182 186 190 194 198 202 206 210 214 218
```