Lecture 4. Matrices and Arrays

R and Data Visualization

BIG2006, Hanyang University, Fall 2022

Matrices and Arrays

A *matrix* is a vector with two additional attributes:

- the number of rows and the number of columns

Matrices are special cases of a more general R type of object: *arrays*.

Arrays can be multidimensional, e.g., a three-dimensional array would consists of rows, columns, and layers.

Note: Since matrices and arrays are vectors, they also have modes, such as numeric and character.

Creating Matrices

- Matrix row and column subscripts begin with 1, i.e., a$[1, 1]$

```
y <- matrix(c(1,2,3,4), nrow=2, ncol=2)
y
```

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

```
y[1,1]
```

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

- The internal storage of matrix is in *column-major order*.

```r
matrix(1:6,2,3)
```

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

```r
m <- matrix(1:6, nrow=2, byrow=T)
m
```

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

General Matrix Operations

Some common operations performed with matrices, including

- linear algebra operations
- matrix indexing
- matrix filtering

Linear Algebra Operations and Indexing

- linear algebra operations: matrix multiplication, matrix scalar multiplication, matrix addition, and so on

```
y <- matrix(c(1,2,3,4),nrow=2,ncol=2)
y %*% y # mathematical matrix multiplication
```

```
##      [,1] [,2]
## [1,]    7   15
## [2,]   10   22
```

```
3*y # mathematical multiplication of matrix by scalar
```

```
##      [,1] [,2]
## [1,]    3    9
## [2,]    6   12
```

```r
y+y # mathematical matrix addition
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

Note: More details of linear algebra operations on matrices will be discussed later (Math and simulations).

- You can assign values to submatrices:

```r
y <- matrix(1:6,nrow=3,ncol=2)
y[c(1,3),]
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    3    6
```

```r
y[c(1,3),] <- matrix(c(1,1,8,12),nrow=2)
y
```

```
##      [,1] [,2]
## [1,]    1    8
## [2,]    2    5
## [3,]    1   12
```

```r
x <- matrix(nrow=3,ncol=3)
y <- matrix(c(4,5,2,3),nrow=2)
y
```

```
##      [,1] [,2]
## [1,]    4    2
## [2,]    5    3
```

```r
x[2:3,2:3] <- y
x
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA    4    2
## [3,]   NA    5    3
```

- Negative subscripts, used with vectors to exclude certain elements, work the same way with matrices:

```
y
```

```
##      [,1] [,2]
## [1,]    4    2
## [2,]    5    3
```

```
y[-2,]
```

```
## [1] 4 2
```

Filtering on Matrices

- Filtering can be done with matrices, just as with vectors.

```
x <- matrix(c(1,2,3,2,3,4),nrow=3,ncol=2)
x
```

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

```
x[x[,2]>=3,]
```

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

```r
j <- x[,2] >= 3
j
```

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

```r
x[j,]
```

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

```r
m <- matrix(1:6,nrow=3,ncol=2)
m
```

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

```r
m[m[,1]>1 & m[,2]>5,]
```

```
## [1] 3 6
```

Note: $\&$ is the vector Boolean AND operators. FYI $\&\&$ is the scalar Boolean AND operator.

Example: Generating a Covariance Matrix

Suppose that we are working with an n-variate normal distribution.

Our matrix will have n rows and n columns, and we wish each of the n variables to have variance 1, with correlation `rho` between pairs of variables.

For `n` = 3 and `rho` = 0.2, the desired matrix is as follows:

$$\begin{pmatrix} 1 & 0.2 & 0.2 \\ 0.2 & 1 & 0.2 \\ 0.2 & 0.2 & 1 \end{pmatrix}$$

```r
makecov <- function(rho,n){
  m <- matrix(nrow=n,ncol=n)
  m <- ifelse(row(m)==col(m),1,rho)
  return(m)
}
makecov(0.2,5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  1.0  0.2  0.2  0.2  0.2
## [2,]  0.2  1.0  0.2  0.2  0.2
## [3,]  0.2  0.2  1.0  0.2  0.2
## [4,]  0.2  0.2  0.2  1.0  0.2
## [5,]  0.2  0.2  0.2  0.2  1.0
```

Applying Functions to Matrix Rows and Columns

One of the most famous and most used features of R is `apply()` family of functions, such as `apply()`, `tapply()`, and `lapply()`.

```
apply(m, dimcode, f, fargs)
```

- `m` is the matrix.
- `dimcode` is the dimension, e.g., 1 (the function applies to rows) or 2 (columns).
- `f` is the function to be applied.
- `fargs` is an optional set of arguments to `f`.

```r
z <- matrix(1:6,nrow=3)
z
```

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

```r
apply(z,1,mean) # rowMeans()
```

```
## [1] 2.5 3.5 4.5
```

```r
apply(z,2,mean) # colMeans()
```

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

- A function you write yourself is just as legitimate for use in `apply()` as any R bulit-in function such as `mean()`.

```
f <- function(x) x/c(2,8)
y <- apply(z,1,f)
y
```

```
##      [,1]  [,2] [,3]
## [1,]  0.5 1.000 1.50
## [2,]  0.5 0.625 0.75
```

```
f <- function(x) x/c(2,8)
y <- apply(z,1,f)
```

- Our `f()` function divides a two-element vector by the vector (2,8).
- The call to `apply()` asks R to call `f()` on each of the rows of `z`.
- The first row is (1,4), so in the call to `f()`, the actual argument corresponding to the formal argument x is (1,4).
- Thus, R computes the value of (1,4)/(2,8), which is (0.5,0.5).

- t(): matrix transpose function

```
t(apply(z,1,f))
```

```
##      [,1]  [,2]
## [1,]  0.5 0.500
## [2,]  1.0 0.625
## [3,]  1.5 0.750
```

Adding and Deleting Matrix Rows and Columns

Matrices can be reassigned, and thus we can achieve the same effect as if we had directly done additions or deletions.

```r
# Recall how we reassign vectors to change their size
x <- c(12,5,13,16,8,20)
x <- c(x[1:3],20,x[4:6]) # insert 20
x
```

```
## [1] 12  5 13 20 16  8 20
```

```r
x <- x[-2:-4] # delete elements 2 through 4
x
```

```
## [1] 12 16  8 20
```

```r
one <- 1:3
z <- matrix(c(1:3,1,1,0,1,0,1),nrow=3,ncol=3)
cbind(one,z) # column bind
```

```
##      one
## [1,]   1 1 1 1
## [2,]   2 2 1 0
## [3,]   3 3 0 1
```

```r
rbind(one,z) # row bind
```

```
##     [,1] [,2] [,3]
## one    1    2    3
##        1    1    1
##        2    1    0
##        3    0    1
```

More on the Vector/Matrix Distinction

Following functions are useful when you are writing a general-purpose library function whose argument is a matrix.

```r
z <- matrix(1:8,nrow=4)
length(z)
```

```
## [1] 8
```

```r
class(z)
```

```
## [1] "matrix" "array"
```

```
attributes(z)
```

```
## $dim
## [1] 4 2
```

```
dim(z)
```

```
## [1] 4 2
```

```
c(nrow(z), ncol(z))
```

```
## [1] 4 2
```

Avoiding Unintended Dimension Reduction

```r
z <- matrix(1:8, nrow=4, ncol=2)
r <- z[2,]
r
```

```
## [1] 2 6
```

```r
str(z)
```

```
##  int [1:4, 1:2] 1 2 3 4 5 6 7 8
```

```r
str(r)
```

```
##  int [1:2] 2 6
```

- R has a way to suppress the dimension reduction: the `drop` argument.

```r
r <- z[2,,drop=FALSE]
r
```

```
##      [,1] [,2]
## [1,]    2    6
```

```r
dim(r)
```

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

- If you want to treat a vector as a matrix, you can use `as.matrix()`.

```r
u <- 1:3
v <- as.matrix(u)
attributes(u)
```

```
## NULL
```

```r
attributes(v)
```

```
## $dim
## [1] 3 1
```

Naming Matrix Rows and Columns

```r
z <- matrix(1:4,nrow=2)
colnames(z)
```

```
## NULL
```

```r
colnames(z) <- c("a","b")
rownames(z) <- c("c","d")
z
```

```
##   a b
## c 1 3
## d 2 4
```

Higher-Dimensional Arrays

In a statistical context, a typical matrix in R has rows corresponding to observations (e.g., various people), and columns corresponding to variables (e.g., weight and blood pressure). The matrix is then a two-dimensional data structure.

Suppose we also have taken at different times, one data point per person per variable per time. Then time becomes the third dimension.

In R, such data sets are called *arrays*.

```r
first_test <- matrix(c(46,21,50,30,25,50),nrow=3)
first_test
```

```
##      [,1] [,2]
## [1,]   46   30
## [2,]   21   25
## [3,]   50   50
```

```r
second_test <- matrix(c(46,41,50,43,35,50),nrow=3)
second_test
```

```
##      [,1] [,2]
## [1,]   46   43
## [2,]   41   35
## [3,]   50   50
```

- In the argument `dim` $= \texttt{c}(3, 2, 2)$, we specify two layers, each consisting of three rows and two columns.
- Each elements of `tests` now has three subscripts.

```
tests <- array(data=c(first_test,second_test),
               dim=c(3,2,2))
attributes(tests)
```

```
## $dim
## [1] 3 2 2
```

```
tests[3,2,1]
```

```
## [1] 50
```

```
tests
```

```
## , , 1
##
##      [,1] [,2]
## [1,]   46   30
## [2,]   21   25
## [3,]   50   50
##
## , , 2
##
##      [,1] [,2]
## [1,]   46   43
## [2,]   41   35
## [3,]   50   50
```

Reference

- Matloff, N. The Art of R Programming: A Tour of Statistical Software Design. No Starch Press. Chapter 3.