MTH 2520 R Notes 4

2 Matrices

2.1 Creating and Examining Matrices

- ***Matrices*** are one way of storing data in a two-dimensional layout (i.e. in rows and columns). They're easily created and examined in R using the functions:

```
matrix()            # Create a matrix, from a vector, with nrow rows
                    # and ncol columns
dim()               # Returns the dimensions (number of rows and
                    # columns) of a matrix
nrow(); ncol()      # Number of rows, number of columns of a matrix
is.matrix()         # Indicates whether or not an object is a matrix
```

- Here's an example showing how to create a matrix from a vector (`1:9`) using `matrix()`:

```
> x <- matrix(1:9, nrow = 3, ncol = 3)
> x

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

> is.matrix(x)

[1] TRUE

> dim(x)

[1] 3 3
```

  Note that `dim()` returns a two-element *vector* giving the number of rows (first element) and number of columns (second element).

- By default, the matrix is created by filling in its *columns*, left to right. An optional argument, `byrow`, can be set to `TRUE` if we want to create it by filling its *rows*:

```
> x <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
> x

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

- If there are too few elements in the vector passed to `matrix()` to fill the matrix, the vector elements are recycled until the matrix has been filled.

  If the length of the vector isn't a multiple (or sub-multiple) of the number of rows (or columns if `byrow = TRUE`), a warning message is printed:

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

Warning message:
In matrix(c(1, 2, 3, 4), nrow = 3, ncol = 3) :
  data length [4] is not a sub-multiple or multiple of the number of rows [3]

> x

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

- If only one of `nrow` or `ncol` is specified, R infers the dimensions of the matrix based on the length of the vector.
- We can also create matrices using:

```
cbind()            # Create a matrix by "binding" vectors together
                   # in columns.
rbind()            # Like cbind(), but "binds" vectors together in
                   # rows.
```

- `cbind()` "binds" two or more vectors of the same length into columns of a matrix, and `rbind()` "binds" them into rows. For example:

```
> x <- cbind(c(1,2,3), c(4,5,6), c(7,8,9))
> x

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

Section 2.1 Exercises

Exercise 1 Here's a matrix `x`:

```
     [,1] [,2] [,3]
[1,]    8    8    8
[2,]    4    4    4
```

Guess what the result of each of the following will be, then check your answers.

a) `> dim(x)`

b) `> dim(x)[1]`

c) `> dim(x)[2]`

d) `> is.vector(dim(x))`

e) `> nrow(x)`

f) `> ncol(x)`

Exercise 2 If there are too few elements in the vector passed to `matrix()` to fill the matrix, the vector elements are recycled until the matrix has been filled.

If the length of the vector isn't a multiple (or sub-multiple) of the number of rows (or columns if `byrow = TRUE`), a warning message is printed.

Guess what the result of each of the following will be, then check your answers.

a) `> matrix(c(1, 2, 3), nrow = 3, ncol = 2)`

b) `> matrix(c(1, 2, 3), nrow = 4, ncol = 2)`

Exercise 3 If only one of `nrow` or `ncol` is specified in `matrix()`, R infers the dimensions of the matrix based on the length of the vector. Guess what the result of each of the following commands will be, then check your answers.

a) `> matrix(1:6, nrow = 2)`

b) `> matrix(1:6, ncol = 2)`

Exercise 4 Here's a matrix `x`:

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

a) Create the matrix using `matrix()`.

b) Create the matrix using `cbind()`.

c) Create the matrix using `rbind()`

2.2 General Matrix Operations

2.2.1 Matrix Arithmetic and Recycling

Matrix Arithmetic

- The operators '+', '-', '*', '/', and '^' operate on matrices elementwise. For example, using the matrices `x` and `y`,

```
> x
```

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

```
> y
```

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

we get

```
> x + y
```

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

```
> x + 2
```

```
     [,1] [,2] [,3]
[1,]    7    7    7
[2,]    6    6    6
[3,]    5    5    5
```

- The built-in R functions that are vectorized, such as `log()`, `sqrt()`, `cos()`, etc. also operate elementwise on matrices.
- (For those familiar with matrix algebra, matrix multiplication is performed using the `%*%` operator.)

Recycling

- We can also carry out arithmetic operations ('+', '-', '*', '/', and '^') using a *vector* and a matrix. For example:

```
> x.mat
```

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

```
> y.vec
```

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

```
> x.mat - y.vec
```

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

Notice that elements of the vector are matched with the columns of the matrix, from left to right.

- If the vector is too short, its elements are recycled. If the length of the vector isn't a sub-multiple of the total number of elements in the matrix, a warning message is printed.

2.2.2 Matrix Indexing Using `[ ]`

Accessing Matrix Elements, Rows, or Columns

- We access matrix elements, rows, or columns using square brackets:

```
[ , ]        # Access matrix elements via their row and column indices
             # (separated by a comma)
```

- To extract a specific element, specify its row and column indices in square brackets `[ ]`, separated by a comma. For example, using the matrix `x`:

```
> x
```

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

if we want to extract the element in the 2nd row and 3rd column, we type:

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

```
[1] 8
```

- An entire row is accessed by leaving the column index blank. For example the 2nd row of `x` is obtained via:

```
> x[2, ]
```

```
[1] 2 5 8
```

Likewise, an entire column is accessed by leaving the row index blank:

```
> x[, 3]
```

```
[1] 7 8 9
```

Replacing Matrix Elements, Rows, or Columns

- The assignment operator `<-` can be used to assign a new value to a matrix element:

```
> x[2, 3] <- 0
> x
```

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

- Here's how to replace an entire column:

```
> x[, 3] <- c(0, 0, 0)
> x
```

```
     [,1] [,2] [,3]
[1,]    1    4    0
[2,]    2    5    0
[3,]    3    6    0
```

Replacing an entire row is done in a similar manner, but the row index is specified *before* the comma in the square brackets.

Adding and Deleting Matrix Rows or Columns

- Negative indices in square brackets (e.g. `x[-1, ]`) return all but that row or column of a matrix.
- We can add a row or column to an existing matrix using `rbind()` and `cbind()`:

```
> x
```

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

```
> cbind(x, c(10, 10, 10))
```

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

Rearranging Matrix Rows or Columns

- Square brackets can also be used to rearrange (permute) the rows or columns of a matrix. To do so, we specify the desired permutation before or after the comma, depending on whether we're rearranging rows or columns. For example:

```
> x
```

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

```
> x[, c(3, 1, 2)]            # Rearranges columns
```

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

Transposing a Matrix

- Sometimes we need to ***transpose*** a matrix, i.e. turn its rows into columns and its columns into rows. We can use the function:

```
t()                        # Transpose a matrix
```

- Here's an example:

```
> x
```

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

```
> t(x)
```

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

Filtering on Matrices

- We can use a `"logical"` vector in square brackets to extract certain rows (or columns) from a matrix. For example:

```
> x
```

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

```
> x[c(TRUE, FALSE, TRUE), ]        # Extracts rows for which the index is TRUE
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    3    6    9
```

  Above, the rows corresponding to `TRUE` in the `"logical"` vector are extracted from `x`.

- Now consider the following data set:

Title	Age	Years Experience	Paid Sick Days
Assembler	43	17	3
Machinist	22	4	7
Technician	37	6	2
Assembler	35	6	0
Engineer	31	3	0
Assembler	27	4	1
Engineer	55	16	0
Machinist	45	19	2
Assembler	35	7	3
Assembler	39	8	1
Machinist	40	14	0
Machinist	44	11	1
Technician	60	21	1

- One way to store the data is as a `"character"` vector of job titles and a matrix containing the numeric values:

```
> title
```

```
 [1] "Assembler"  "Machinist"  "Technician" "Assembler"  "Engineer"   "Assembler"  "Engineer
[10] "Assembler"  "Machinist"  "Machinist"  "Technician"
```

```
> values
```

```
      Age Years Experience Paid Sick Days
 [1,]  43                17              3
 [2,]  22                 4              7
 [3,]  37                 6              2
 [4,]  35                 6              0
 [5,]  31                 3              0
 [6,]  27                 4              1
 [7,]  55                16              0
 [8,]  45                19              2
 [9,]  35                 7              3
[10,]  39                 8              1
[11,]  40                14              0
[12,]  44                11              1
[13,]  60                21              1
```

(We'll see later how the column headings were added to the `values` matrix).

- To extract from `values` just the rows corresponding to Machinists, we can type:

```
> values[title == "Machinist", ]          # title=="Machinist" is a "logical" vector
```

```
     Age Years Experience Paid Sick Days
[1,]  22                 4              7
[2,]  45                19              2
[3,]  40                14              0
[4,]  44                11              1
```

Note that the expression `title == "Machinist"` is actually a `"logical"` vector, whose `TRUE` values indicate which rows of `values` to extract.

- Likewise, to extract from `values` just the rows corresponding to employees who are older than 40, we type:

```
> values[values[, 1] > 40, ]              # values[, 1]>40 is a "logical" vector
```

```
     Age Years Experience Paid Sick Days
[1,]  43                17              3
[2,]  55                16              0
[3,]  45                19              2
[4,]  44                11              1
[5,]  60                21              1
```

Note that above, the expression `values[, 1] > 40` is a `"logical"` vector, whose `TRUE` values indicate which rows of `values` to extract.

Section 2.2 Exercises

Exercise 5 Consider the matrices `x` and `y`:

```
> x
```

```
     [,1] [,2]
[1,]    1    9
[2,]    4   16
```

```
> y
```

```
     [,1] [,2]
[1,]    2    2
[2,]    1    1
```

Guess what the result of each of the following will be, then check your answers.

a) `> x * y`

b) `> x + 1`

c) `> sqrt(x)`

Exercise 6 Consider the following matrix and vector:

```
> x.mat
```

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

```
> y.vec
```

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

Guess what the result of the following will be, then check your answer.

```
> x.mat + y.vec
```

Exercise 7 Consider the following matrix and vector:

```
> x.mat
```

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

```
> y.vec
```

```
[1] 1 2 3
```

Guess what the result of the following will be, then check your answer.

```
> x.mat - y.vec
```

Exercise 8 Consider the following matrix:

```
> x
```

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

Guess what the result of each of the following will be, then check your answers.

a) `> x[1, 3]`

b) `> x[1, ]`

c) `> x[, 3]`

d) `> x[c(1, 2), ]`

e) `> x[c(1, 2), 3]`

f) `> x[-1, ]`

g) `> x[-1, 3]`

Exercise 9 Consider the following matrix:

```
> x
```

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

Guess what the result of each of the following will be, then check your answers.

a) `> x[, c(3, 1, 2)]`

b) `> x[c(2, 1), ]`

Exercise 10 Consider the following matrix:

```
> x
```

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

Guess what the result of the following will be, then check your answer.

```
> x[c(TRUE, TRUE, FALSE), ]
```

Exercise 11 Consider the following data set:

Gender	Age	Number of Siblings	Number of Pets	Age of Oldest Pet
Male	11	1	2	5
Male	8	2	1	3
Female	12	1	2	2
Female	9	0	0	NA
Male	12	3	0	NA
Female	10	4	1	6

These data can be stored as a `"character"` vector and numeric matrix:

```
> gender <- c("Male", "Male", "Female", "Female", "Male", "Female")
> values <- cbind(c(11, 8, 12, 9, 12, 10), c(1, 2, 1, 0, 3, 4),
                  c(2, 1, 2, 0, 0, 1), c(5, 3, 2, NA, NA, 6))
```

To add column headings to the `values` matrix, we type:

```
> colnames(values) <- c("Age", "Number of Siblings", "Number of Pets",
                        "Age of Oldest Pet")
```

a) Write a command that will extract from `values` just the rows corresponding to males.

b) Write a command that will extract just the rows corresponding kids who are over the age of 10.

2.3 The `apply()` Function

- Sometimes we need to apply a function separately to each row (or each column) of a matrix. To do so, we use:

```
apply()          # Apply a function separately to each row (or each
                 # column) of a matrix
```

- `apply()` has three main arguments, `X`, a matrix, `FUN`, the function to be applied to the rows (or columns) of `X`, and `MARGIN` (`MARGIN = 1` to apply a function to rows and `MARGIN = 2` to apply it to columns).
- For example, below, `apply()` is used to compute the sum of each row:

```
> x

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

> apply(X = x, MARGIN = 1, FUN = sum)

[1] 12 15 18
```

Note that `apply()` returns a vector, each element of which is the sum of the corresponding row in `x`.

- If the function specified by `FUN` requires additional arguments, they're passed through `apply()` after `FUN`. For example, here are the arguments for the function `sample()`:

```
> args(sample)

function (x, size, replace = FALSE, prob = NULL)
NULL
```

Its two main arguments are `x`, a vector, and `size`, a sample size. It generates a random sample of the specified `size` from the elements of the vector.

To use `apply()` to randomly select one value (i.e. a sample of size one, so `size = 1`) from each row of a matrix, we type:

```
> apply(X = x, MARGIN = 1, FUN = sample, size = 1)

[1] 1 5 3
```

Note that the sample `size` was specified after `FUN` in the call to `apply()`.

Section 2.3 Exercises

Exercise 12 Consider the following matrix `x`:

```
> x <- matrix(c(8, 6, 3, 6, 5, 7, 2, 1, 9, 4, 7, 6, 8, 3, 4, 3),
              nrow = 4, ncol = 4)
> x
```

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

Guess what the result of each of the following will be, then check your answers.

a) `> apply(x, MARGIN = 1, FUN = min)`

b) `> apply(x, MARGIN = 2, FUN = min)`

c) `> apply(x, MARGIN = 1, FUN = which.min)`

Exercise 13 R has several built-in datasets, one of which is `USPersonalExpenditure`, a matrix with five rows and five columns consisting of U.S. personal expenditures (in billions of dollars) in the categories: food and tobacco, household operation, medical and health, personal care, and private education for the years 1940, 1945, 1950, 1955 and 1960.

Type

```
> USPersonalExpenditure
```

to view the matrix.

a) Use `apply()`, with `FUN = mean`, to find the mean expenditure for each of the five expenditure categories. **Hint**: Should `MARGIN` be `1` or `2`?

b) Use `apply()`, with `FUN = mean`, to find the mean expenditure for each of the five years. **Hint**: See the hint for part *a*.

Exercise 14 Consider the following matrix `x`:

```
> x <- matrix(c(8, 6, 3, 6, 5, 7, 2, 1, 9, 4, 7, 6, 8, 3, 4, 3),
              nrow = 4, ncol = 4)
> x
```

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

Write a command that uses `apply()`, with `FUN = sort`, to sort each column of `x`.

Exercise 15 Consider the following matrix `x`:

```
> x <- matrix(c(8, NA, 3, NA, 5, 7, 2, 1, NA, NA, 7, 6, 8, 3, 4, 5),
              nrow = 4, ncol = 4)
> x
```

```
     [,1] [,2] [,3] [,4]
[1,]    8    5   NA    8
[2,]   NA    7   NA    3
[3,]    3    2    7    4
[4,]   NA    1    6    5
```

What do you think the following command will do? Note that the argument `na.rm` (for removing `NA`s before doing computations) is passed to `mean()` by specifying it after `FUN` in the call to `apply()`.

```
> apply(x, MARGIN = 1, FUN = mean, na.rm = TRUE)
```

2.4 Naming Matrix Rows and Columns

- We can assign names to the rows or columns of a matrix, or get existing names, using:

```
rownames()          # Get or assign the row names of a matrix
colnames()          # Get or assign the column names of a matrix
```

- Here's an example:

```
> x
```

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

```
> colnames(x) <- c("ColA", "ColB",  "ColC")       # Assigns column names to x
> x
```

```
     ColA ColB ColC
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
> colnames(x)                    # Gets the column names from x

[1] "ColA" "ColB" "ColC"
```

- To remove the row or column names, we assign `NULL` to them:

```
> colnames(x) <- NULL
```

Section 2.4 Exercises

Exercise 16 Consider the following matrix `x`:

```
> x <- matrix(1:9, nrow = 3, ncol = 3)
```

a) After creating the matrix, write a command that assigns the names `"a", "b", "c"` to its columns.

b) Now that you've created column names, type the following command. What does it do?

```
> colnames(x) <- NULL
```

2.5 Matrices are Vectors with a Dimensions Attribute

- In R, an ***attribute*** is a piece of *metadata* about an object. *Metadata* are data about other data. We can get or assign attributes to an object using:

```
attributes()      # Get or assign all the attributes of an object
attr()            # Get or assign specific attributes of an object
```

- `attributes()` accesses all the attributes of an object, `attr()` accesses only specific ones via their names.
- All matrices have a dimensions attribute that indicates the number of rows and columns:

```
> x

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

> attributes(x)
```

```
$dim
[1] 3 3

> attr(x, "dim")

[1] 3 3
```

- In fact, technically, a matrix is just a vector with a dimensions attribute. To see, watch what happens when we assign a dimensions attribute to a vector:

```
> x <- 1:9
> x

[1] 1 2 3 4 5 6 7 8 9

> is.vector(x)

[1] TRUE

> attr(x, "dim") <- c(3, 3)          # We could also use dim(x) <- c(3, 3)
> x

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

> is.matrix(x)

[1] TRUE
```

- Because a matrix is a vector (with a dimensions attribute), it will often behave like a vector (consisting of the matrix's columns placed end to end). For example:

```
> x                                  # x is a matrix

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

> length(x)                          # x behaves like a vector here

[1] 9

> x[4]                               # and here

[1] 4
```

- We can explicitly coerce a matrix to a vector using:

```
as.vector()                    # Coerce an object to a vector
```

- For example, to coerce the matrix `x` from above to a vector, type:

```
> as.vector(x)

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

Section 2.5 Exercises

Exercise 17 An ***attribute*** is a piece of *metadata* containing information about an R object. All matrices have a dimensions attribute. Consider the following matrix `x`:

```
> x

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

Guess what the result of the following command will be, then check your answer:

```
> attributes(x)
```

Exercise 18 Technically, a matrix is just a vector with a dimensions attribute. So when you pass a matrix to a function, like `sum()`, that's expecting a vector, R behaves as if the matrix is a vector.

Consider the following matrix `x`:

```
> x

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

Guess what the result of each of the following will be, then check your answers:

a) `> sum(x)`

b) `> max(x)`

c) `> x[5]`

Exercise 19 Consider the following matrix `x`:

```
> x
```

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

Guess what the result of the following will be, then check your answer:

```
> as.vector(x)
```