Lecture 11 Module 4 – Matrices

CSCI E-5a: Introduction to R

Let's clear the global computing environment:

```
rm( list = ls() )
```

Module Overview

In this module, we will investigate one more compound data structure, the *matrix*.

- In Section 1, we'll define the concept of a matrix.
- In Section 2, we'll learn how to create a matrix.
- In Section 3, we'll see how to combine matrices.
- In Section 4, we'll learn how to index a matrix.

When you've completed this module, you'll be able to:

- Define the concept of a matrix
- Create a matrix
- Combine matrices
- Index a matrix

There are two new built-in R functions in this module:

- `matrix()`
- `dimnames()`

All right! Let's get started by defining the concept of a matrix.

Section 1: Definition

Main Idea: *We can define the concept of a matrix*

In this section, we'll define the concept of a matrix.

So far, we've seen three compound data structures: vectors, data frames, and lists.

Another compound data structure in R is called a *matrix*, which is similar to a vector.

By the way, the plural of "matrix" is "matrices".

Recall the two defining properties of a vector:

- A vector is always a one-dimensional structure.
- A vector always consists of elements of the same atomic data type (e.g. numeric, logical, or character).

For a matrix, the first condition is changed, so that a matrix is defined to be a two-dimensional structure.

However, the second condition still holds: all the elements of a matrix must be of the same atomic data type.

This is the opposite of the list data structure, which relaxes the second condition, so that it can contain elements of different classes, but maintains the first condition, so that it is still one-dimensional.

We'll use the matrix data structures a few times in our course, but it won't be one of primary topics.

Instead, I want to show you just some of the basic ideas, so that if you encounter a matrix in your work you'll have some idea of what it is.

Matrices are very important in more advanced courses in statistics and data science, but this is beyond the scope of our course.

So that's how to define the concept of a matrix.

Now let's see how to create a matrix.

Section 2: Creating a matrix

Main Idea: *We can create a matrix*

In this section, we'll learn how to create a matrix.

The function for creating a matrix is `matrix()`, and there are a number of different ways to use it.

In all approaches, the first input argument is a vector, but then we control how this is converted to a matrix by setting options.

Here's a simple way to make a matrix, in which we specify the number of rows:

```
create.example.matrix.vector <- 1:12

example.matrix <-
    matrix( create.example.matrix.vector, nrow = 4 )

example.matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Notice that for a given input vector once the number of rows has been specified then the number of columns has been implicitly determined, and R does this automatically.

Here's another way to create a matrix, this time by specifying the number of columns:

```
matrix( create.example.matrix.vector, ncol = 3 )
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Notice that in these two examples the matrix was populated with the values from the input vector by going down the first column, then down the second column, and finally down the third column.

You can change this so that the matrix is populated by row, not by column, by using the `byrow` parameter:

```
matrix( create.example.matrix.vector, nrow = 4, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

Do you see how the values from the input vector now populate the matrix in a different order?

Incidentally, sometimes you just want to allocate space for a matrix, without necessarily populating it.

In that case, you can just call the `matrix()` function with no data, but specifying both the number of rows and the number of columns.

```
matrix(
    nrow = 3,
    ncol = 3
)
```

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

We can also change the names on the rows and the columns by using the `dimnames()` function, which takes a list with two elements: the first element is a vector of the row names, and the second element is a vector of the column names.

```
names.example.matrix <-
    matrix( 1:6, nrow = 2)

names.example.matrix
```

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

We can assign names to the rows using the `rownames()` function:

```
rownames( names.example.matrix ) <-
    c( "Bob", "Anita" )

names.example.matrix
```

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

We can assign names to the columns using the `colnames()` function:

```
colnames( names.example.matrix ) <-
    c( "Monday", "Tuesday", "Wednesday" )

names.example.matrix
```

```
##       Monday Tuesday Wednesday
## Bob        1       3         5
## Anita      2       4         6
```

```
paste( c("Hedgehog", "Armadillo"), c("Red", "White", "Blue") )
```

```
## [1] "Hedgehog Red"    "Armadillo White" "Hedgehog Blue"
```

We can assign both rowname and column names using the `dimnames()` function:

```
dimnames( names.example.matrix ) <-
    list( c("Hedgehog", "Armadillo"), c("Red", "White", "Blue") )

names.example.matrix
```

```
##           Red White Blue
## Hedgehog    1     3    5
## Armadillo   2     4    6
```

You can flip the rows and columns of a matrix by using the `t()` function:

```
example.matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
t( example.matrix )
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

So that's how to create a matrix.

Now let's see how to combine matrices.

Exercise

Can you think of another way to create this matrix?

```
matrix( 1:12, ncol = 4, byrow = TRUE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

Solution

```
# Type your answer in here
```

Here's my answer:

```
matrix( 1:12, nrow = 3, byrow = TRUE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

Section 3: Combining Matrices

Main Idea: *We can combine matrices*

In this section, we'll see how to combine matrices.

If you have two matrices with the same number of columns, you can bind them together using `rbind()`.

Let's make an example matrix:

```
rbind.example.matrix.1 <-
    matrix( 1:6, nrow = 2, byrow = TRUE)

rbind.example.matrix.1
```

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

Now we'll make a second example matrix:

```
rbind.example.matrix.2 <-
    matrix( 7:12, nrow = 2, byrow = TRUE)

rbind.example.matrix.2
```

```
##      [,1] [,2] [,3]
## [1,]    7    8    9
## [2,]   10   11   12
```

Now we can combine these two matrices, because they both have three columns:

```
rbind( rbind.example.matrix.1, rbind.example.matrix.2 )
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

You can think of this as binding the rows together, so that we join the rows of the second matrix to the end of the rows of the first matrix.

That's why the function is named `rbind()` – because it's binding rows together.

You can also bind a vector to a matrix:

```
example.vector <- 21:23

rbind( rbind.example.matrix.1, example.vector )
```

```
##                 [,1] [,2] [,3]
##                    1    2    3
##                    4    5    6
## example.vector    21   22   23
```

If you attempt to bind a row to a matrix where the number of columns don't match, R will recyle the row values.

```
weird.matrix <- matrix( 1:24, ncol = 8, byrow = TRUE)

bad.vector <- 1:4

rbind( weird.matrix, bad.vector )
```

```
##             [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
##                1    2    3    4    5    6    7    8
##                9   10   11   12   13   14   15   16
##               17   18   19   20   21   22   23   24
## bad.vector     1    2    3    4    1    2    3    4
```

In general, I'm not crazy about exploiting this recycling feature.

There's also the `cbind()` function, which takes two matrices that have the same number of rows and binds them together as columns.

Our two example matrices both have 2 rows, so we can use them with `cbind()`:

```
cbind(
    rbind.example.matrix.1,
    rbind.example.matrix.2
)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    7    8    9
## [2,]    4    5    6   10   11   12
```

So that's how to combine matrices.

Now let's learn how to index a matrix.

Section 4: Indexing a matrix

Main Idea: *We can index a matrix*

In this section, we'll learn how to index a matrix.

A matrix is a two-dimensional object, so we need to use two numbers to specify a location.

Recall our example matrix:

```
example.matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Then, to select the element in the third row and second column, we use a single opening square bracket, then the row index, then the column index, and finally a single closing square bracket:

```
example.matrix[ 3, 2 ]
```

```
## [1] 7
```

You can extract an individual column of a matrix by leaving the row field empty:

```
example.matrix[ , 2 ]
```

```
## [1] 5 6 7 8
```

Notice that when we extract this column, it is returned as a vector.

Similarly, you can extract an individual row by leaving the column field empty:

```
example.matrix[ 2, ]
```

```
## [1]  2  6 10
```

Again, this row is returned as a vector.

We can also index the matrix using row and column names.

```
names.example.matrix
```

```
##           Red White Blue
## Hedgehog    1     3    5
## Armadillo   2     4    6
```

To obtain the element for the row "Hedgehog" and the column "Blue", we have:

```
names.example.matrix[ "Hedgehog", "Blue"]
```

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

We can also extract rows and columns using names and a blank entry:

```
names.example.matrix[ , "Red" ]
```

```
##  Hedgehog Armadillo
##         1         2
```

So that's how to index a matrix.

Now let's review what we've learned in this module.

Module Review

In this module, we investigated one more compound data structure, the *matrix*.

- In Section 1, we defined the concept of a matrix.
- In Section 2, we learned how to create a matrix.
- In Section 3, we saw how to combine matrices.
- In Section 4, we learned how to index a matrix.

Now that you've completed this module, you should be able to:

- Define the concept of a matrix
- Create a matrix
- Combine matrices
- Index a matrix

There were two new built-in R functions in this module:

- `matrix()`
- `dimnames()`

All right! That's it for Module 4: Matrices.

Now let's move on to Module 5: Two-Way Tables and Barplots.