MAS115 R programming, Lab Class 3

Bryony Moody (Original notes by Prof. P. G. Blackwell) *

2021-22

1 Matrices and arrays

1.1 Creating a matrix

A matrix is just a vector with an additional structure of rows and columns. As such, all of its elements must be of the same mode i.e. double, integer, ... It represents the standard mathematical idea of a two-dimensional matrix.

To create a matrix you use the command `matrix()`. You pass the values that you want to fill in the matrix with e.g. 1:12 and then tell R the number of rows and/or columns you want.

```
# For example:
(mat <- matrix(1:12, nrow = 3, ncol = 4))
```

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

As you will see you need to specify the number of rows or the number of columns or both. As usual R will try and recycle if the data you want to put in isn't of the correct length.

Note: `matrix()` will fill up the entries in the matrix working down the columns unless you specify `byrow = TRUE` as one of the arguments to `matrix()`.

You can also add new rows/columns to a matrix using the `rbind()` and `cbind()` commands. This can also be used to create a matrix by combining two or more vectors.

```
rbind(mat,c(1,2,3,4))
```

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

*Thanks to Dr. T Heaton & Dr. R Ripley for suggestions

```
cbind(mat, c(1,2,3))
```

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

1.2 Operating on a matrix

If you have two numeric matrices which are the same size then you can add, subtract, multiply,... *elementwise* using the standard operations.

```
matx <- matrix(1:4, nrow = 2, ncol = 2)
maty <- matrix(1:4, nrow = 2, ncol = 2)
matx*maty
```

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

Also if the matrices are of the correct size you can do *matrix multiplication* (don't worry if you haven't seen this yet) by using the **%*%** operator

```
matz <- matrix(1:6, nrow = 2, ncol = 3)
matx %*% matz
```

```
     [,1] [,2] [,3]
[1,]    7   15   23
[2,]   10   22   34
```

1.3 Indexing elements

As you might guess from the way R has output the result above, you can identify individual elements by specifying their index using `[,]` brackets. The row subscript that you want goes before the comma while the column subscript goes after the comma.

```
mat[2,3]
```

```
[1] 8
```

You don't need to specify both the row and the column but can just leave one blank (although you must still have the comma). This allows you to pull out rows/columns too.

```
mat[1,]
```

```
[1]  1  4  7 10
```

```
mat[,2]
```

```
[1] 4 5 6
```

This works very similarly to how you index vectors but with one entry for the indices of the row and another for the column. You can also do more complicated combinations.

```
# Think about what elements this selects
mat[c(1,3), -3]
```

```
     [,1] [,2] [,3]
[1,]    1    4   10
[2,]    3    6   12
```

If the elements that you select have only 1 row or only 1 column then R will drop that dimension and just return a vector. This is often not what we want (and can cause errors if we are then going to use the extracted part later) but we can stop it doing this by specifying `drop = FALSE` as an extra argument in the indexing (note the extra comma!):

```
mat[1,] # Just a vector
```

```
[1]  1  4  7 10
```

```
mat[1,, drop = FALSE] # Still a matrix
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
```

As well as indexing by rows and/or columns in this way, it is possible to select an arbitrary set of elements, by using `[,]` with *one* argument which is itself a two-column matrix of integer indices.

1.4 Changing elements

Finally you can replace elements using this indexing or conditioning on the value of the matrix.

```
matx[1,] <- c(4,5)
matx
```

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

```
mat # To remind you of the values originally entered for this matrix
```

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

```
mat[mat > 4] <- 10
mat
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4   10   10
[2,]    2   10   10   10
[3,]    3   10   10   10
```

1.5 Arrays

A higher-dimensional array works very similarly to a matrix, but has more than two dimensions. Indexing normally takes a number of arguments equal to the number of dimensions of the array.

1.6 Tasks

1. Using the `matrix()` function, create (and store as `A`) the matrix

$$\begin{pmatrix} 1 & 9 & 14 \\ 5 & 21 & 36 \end{pmatrix}$$

2. Create the same matrix by creating two separate vectors and combining them using the `rbind()` command.
3. Extract the second row of `A` as a matrix.
4. Extract the third column of `A` as a vector.
5. Without altering elements individually, replace all the values of `A` that are bigger than 15 by `NA` (the special value for missing data).
6. Optional: investigate selecting individual elements of a matrix using a 2-column matrix as the index.
7. Optional: investigate higher-dimensional arrays.

2 While and repeat loops

2.1 The `while` loop

Using a `while` loop

As well as `for` loops, R provides `while` loops which allow you to repeat a statement until a particular condition becomes `FALSE`. Run the following code and try to figure out what it does.

```
# A while loop
nobs <- 10
x <- runif(nobs)
while((mean(x) < 0.49) || (mean(x) > 0.51)) {
  nobs <- 2 * nobs
  cat("Mean is", mean(x), "\n")
  cat("Need to try number = ", nobs, "\n")
  x <- runif(nobs)
}
```

```
Mean is 0.4522553
Need to try number =  20
```

```
nobs
```

```
[1] 20
```

```
mean(x)
```

```
[1] 0.5059124
```

Syntax

The format to construct a while loop is as follows:

```
while(condition) statement
```

This construction will cause `statement` to be repeated until `condition` is `FALSE`. For it to work then you need the condition to take a single value (i.e. be a vector of length 1). The `statement` can be compound as in the example above in which case you need to enclose it in brackets `{ }`

2.2 The `repeat` loop

Using a `repeat` loop

R's final method of producing loops is to use the command `repeat`. Try the following code and make sure that you understand what it does.

```
# A repeat loop
i <- 0
repeat {
  i <- i+1
  x <- rnorm(1)
  cat("Try", i, "realisation is", x, "\n")
  if(x > 1.2) break
}
```

```
Try 1 realisation is -0.3076564
Try 2 realisation is -0.9530173
Try 3 realisation is -0.6482428
Try 4 realisation is 1.224314
```

```
i
```

```
[1] 4
```

Syntax

The syntax for a `repeat` loop is as follows:

```
repeat   statement
```

If it is used then `statement` will be repeated until the flow is transferred out of the loop using the `break` statement. Typically we will therefore combine `repeat` loops with `if` statements which determine when we want to break out of the loop.

Note: If you can't remember the `break` *statement then look back at last week's practical sheet.*

Subtle differences between `while` and `repeat`

The different loops available in R can normally be transferred between e.g. you can rewrite a `for` loop as a `while` or `repeat` loop and so on. It is however worth pointing out a small difference between the computer's response when writing a `while` loop and a `repeat` loop:

- In the case of the `while` loop, the loop will only be entered if the initial condition is `TRUE`. Hence it is possible that the loop will never be performed.
- In the case of the `repeat` loop, the loop will always be entered at least once since it will only be exited when the `break` command is encountered.

2.3 Killing `while` and `repeat` loops if you've made a mistake

Sometimes when you write either a `while` or a `repeat` loop you will make an error in your code which means that the condition required to exit the loop will never be attained. In such cases then your computer will try and run through the loop forever and you will need to stop the code running manually yourself. In order to do this you will need to click in the R console window (where the code is actually running) and then press the `Esc` key, or click on the red STOP icon. Try this on the following code:

```
# A loop which never ends
while(TRUE) {
    cat("Trapped inside a never-ending loop", "\n")
}
```

3 Homework — due 10am Tuesday week 4

Your solutions must be in the form of a PDF document produced from an R markdown file, including a suitable title and your name in the header and code, results and explanation for each task. Please submit your PDF via blackboard by **10am Tuesday 1st March 2022**.

1. **Triangular numbers** The kth triangular number, T_k, is

$$\sum_{j=1}^{k} j \;=\; \frac{k(k+1)}{2}.$$

   Use a loop to search for triangular numbers that are also perfect squares; keep going until you have found seven of them. For each of the seven, record k and T_k, and also the number that T_k is the square of.

2. **Cumulative sums**

Consider a sequence of random variables $X_1, X_2, \ldots$ where each $X_i \sim f$ for some distribution f. Suppose we are interested in finding out how many of the random variables we need until the sum exceeds 100 i.e. we want to find out the number N such that

$$\sum_{i=1}^{N-1} X_i \leq 100 \quad \text{but} \quad \sum_{i=1}^{N} X_i > 100.$$

This number N will itself be a random variable as it depends upon the values of the individual X_i's.

For the experiments below, take the individual X_i's to have an Exponential distribution with mean 5 (look at `?rexp`). Note: you need to be careful with defining an Exponential random variable as some people use the rate while others define the scale.

(a) Plan out code for generating a single value of N, and implement it in R.

(b) Plan out code for generating a sample of 1000 values for N, for example by placing your previous solution inside a FOR loop.

(c) Implement your program in R; show your code, and a small sample of the values of N, for example by using `head`.

(d) Plot a histogram of N. Look at `?hist`" for how to create a histogram in R if you haven't already come across this command.

(e) What is the probability that $N > 30$?