R13

Matrices

Covered in R13

- Creating matrices
- Naming matrices
- Indexing matrices
- Matrix arithmetic
- Other matrix functions

1 Creating matrices

Recall from a previous chapter that a vector is a one-dimensional ordered collection of data of the *same type* (numeric, character, logical, complex or raw). In this chapter we look at matrices.

A matrix is a two-dimensional ordered collection of data of the *same type*. For example:

$$\begin{pmatrix} 3 & -2.1 \\ 4.9 & 8.6 \end{pmatrix} \quad \begin{pmatrix} \text{"barry"} & \text{"alice"} \\ \text{"harry"} & \text{"belinda"} \\ \text{"larry"} & \text{"chelsea"} \end{pmatrix} \quad \begin{pmatrix} \text{TRUE} & \text{TRUE} & \text{FALSE} \\ \text{FALSE} & \text{TRUE} & \text{FALSE} \end{pmatrix}$$

We can find out whether an object is a matrix by using is.matrix(<object>). The dimensions are called rows and columns and the numbers of each can be found by using the nrow(<matrix object>) and ncol(<matrix object>), respectively. Alternatively, you could use the dimensions command dim(<object>) to get both the number of rows and columns.

To create a matrix we use the matrix( ) command:

matrix(<data>, nrow = .. , ncol = ..)

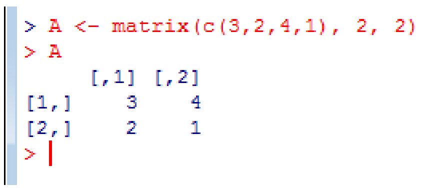
Suppose we want to create a 2×2 matrix called A containing the values 3, 2, 4 and 1. We'll need to use the concatenate, c( ), function in the first argument to let R know these 4 values are the data. Otherwise it will think we have one value 3 in a matrix with 2 rows and 4 columns!

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

Or as long as we enter the arguments in the correct order we can omit their names:

```
A <- matrix(c(3,2,4,1), 2, 2)
```

Notice how by default R will fill the elements of the matrix by column:



```
> A <- matrix(c(3,2,4,1), 2, 2)
> A
     [,1] [,2]
[1,]    3    4
[2,]    2    1
>
```

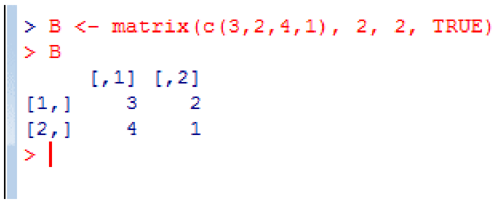
This is because R thinks of a matrix as a collection of vectors of the same length.

To specify that we want to fill the elements of the matrix by row we need to specify another argument of the matrix function, byrow, as TRUE.

To create a matrix B with these same data values but filled by row, we would type:

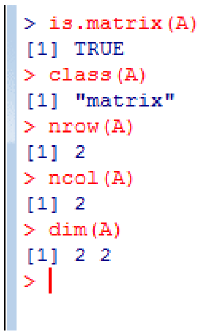
```
B <- matrix(c(3,2,4,1), nrow = 2, ncol = 2, byrow = TRUE)
```

Or again, as long as we enter the arguments in the correct order we can omit their names:



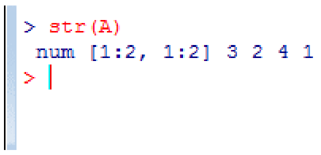
```
> B <- matrix(c(3,2,4,1), 2, 2, TRUE)
> B
     [,1] [,2]
[1,]    3    2
[2,]    4    1
> 
```

Let's look at the properties of matrix A:



```
> is.matrix(A)
[1] TRUE
> class(A)
[1] "matrix"
> nrow(A)
[1] 2
> ncol(A)
[1] 2
> dim(A)
[1] 2 2
> 
```

Recall that another way of obtaining all this information in one go is to use the str(<object>) command which displays the structure of an R object:

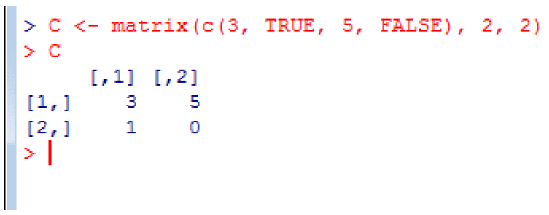


```
> str(A)
 num [1:2, 1:2] 3 2 4 1
> 
```

It says it is numeric data, gives the dimensions (rows, columns) and the contents listed in column order (in this case all of the contents, but for larger objects it will only give some of them).

Coercing

Since a matrix is a collection of data of the same type, if we try to create a matrix of different data types, R will try to coerce them all to the same type. For example:

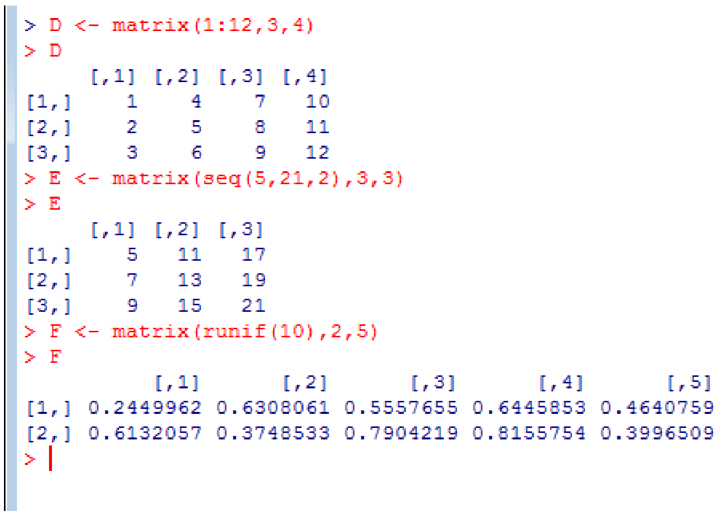


```
> C <- matrix(c(3, TRUE, 5, FALSE), 2, 2)
> C
     [,1] [,2]
[1,]    3    5
[2,]    1    0
> 
```

In this case it has converted the logical data into numeric data (TRUE becomes 1 and FALSE becomes 0) so that it is now a matrix of numeric data .

Alternative ways of specifying the elements

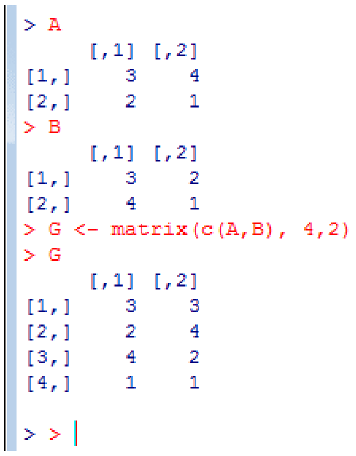
Just like we did with vectors, we can specify the elements using the consecutive integer function a:b or the sequence function seq(from, to, by, length) or the simulation functions such as runif(n) or rnorm(n):



```
> D <- matrix(1:12,3,4)
> D
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> E <- matrix(seq(5,21,2),3,3)
> E
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]    7   13   19
[3,]    9   15   21
> F <- matrix(runif(10),2,5)
> F
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.2449962 0.6308061 0.5557655 0.6445853 0.4640759
[2,] 0.6132057 0.3748533 0.7904219 0.8155754 0.3996509
> |
```

Creating matrices from other objects

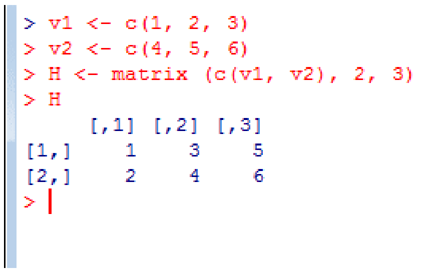
We can create matrices from vectors or other matrices. We can do this using the matrix function. This will take the elements from the objects and use them in order regardless of the length of the vectors of sizes of the matrices. Again, by default it will assign the elements in columns:



```
> A
     [,1] [,2]
[1,]    3    4
[2,]    2    1
> B
     [,1] [,2]
[1,]    3    2
[2,]    4    1
> G <- matrix(c(A,B), 4,2)
> G
     [,1] [,2]
[1,]    3    3
[2,]    2    4
[3,]    4    2
[4,]    1    1

> > |
```

So in the above example we can see that all 4 elements from Matrix A have been read in columns and then used to fill up the first column of the new matrix. Likewise all 4 elements from Matrix B form the second column.

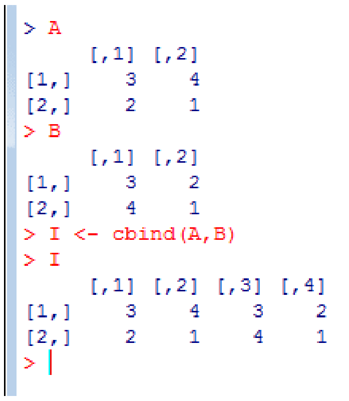
Here's an example where we form a matrix from 2 vectors:



```
> v1 <- c(1, 2, 3)
> v2 <- c(4, 5, 6)
> H <- matrix (c(v1, v2), 2, 3)
> H
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> 
```

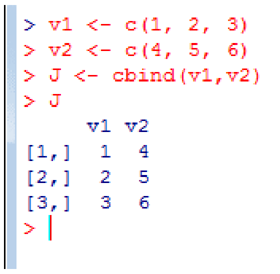
Creating matrices using cbind and rbind

An alternative method of creating matrices from vectors or other matrices is to use the column bind (cbind) or row bind (rbind) functions. This binds the columns or rows together to form a new matrix. First let's use column bind:



```
> A
     [,1] [,2]
[1,]    3    4
[2,]    2    1
> B
     [,1] [,2]
[1,]    3    2
[2,]    4    1
> I <- cbind(A,B)
> I
     [,1] [,2] [,3] [,4]
[1,]    3    4    3    2
[2,]    2    1    4    1
> 
```

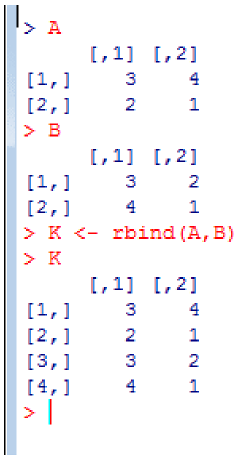
We can see the new matrix is made up of the two new matrices stuck side by side (*ie* by column). Similarly when we column bind two vectors:



```
> v1 <- c(1, 2, 3)
> v2 <- c(4, 5, 6)
> J <- cbind(v1,v2)
> J
     v1 v2
[1,]  1  4
[2,]  2  5
[3,]  3  6
> 
```

We can see that the vectors are combined size by side (*ie* by columns) to form a matrix. This is what we would expect as they are column vectors even though they are displayed horizontally.

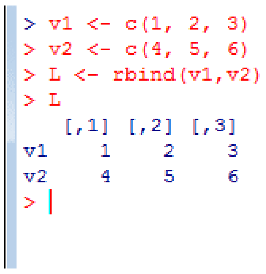
Let’s now use row bind on the two matrices:



```
> A
     [,1] [,2]
[1,]    3    4
[2,]    2    1
> B
     [,1] [,2]
[1,]    3    2
[2,]    4    1
> K <- rbind(A,B)
> K
     [,1] [,2]
[1,]    3    4
[2,]    2    1
[3,]    3    2
[4,]    4    1
> 
```

We can see the new matrix is made up of the two new matrices stuck one on the other (*ie* by row).

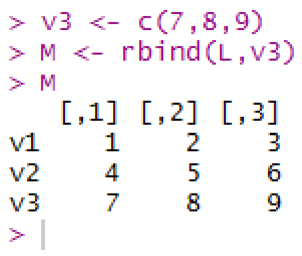
Combining the two vectors using row bind gives:



```
> v1 <- c(1, 2, 3)
> v2 <- c(4, 5, 6)
> L <- rbind(v1,v2)
> L
   [,1] [,2] [,3]
v1    1    2    3
v2    4    5    6
> 
```

We can see that the vectors have been treated as row vectors and put on top of each other.

We can also use these functions if we want to add another row or column to a matrix:



```
> v3 <- c(7,8,9)
> M <- rbind(L,v3)
> M
   [,1] [,2] [,3]
v1    1    2    3
v2    4    5    6
v3    7    8    9
> 
```

2 Naming matrices

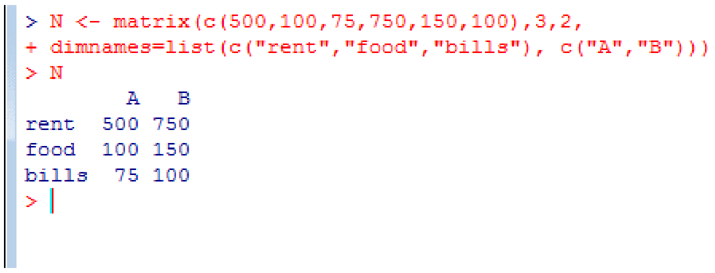
Recall that we could name each of the elements in our vectors. For matrices, we can name the rows and columns. There are two ways of doing this. The first is to use the option dimnames in the matrix command:

matrix(<data>, nrow = .., ncol = .. , byrow = FALSE, dimnames = list(<row names>, <col names>))

The "list" object for dimension names is because it is just one argument of the command but we need names for both dimensions. We use the concatenate, c, command to combine together the rows and the columns. Finally since the names are characters they should be enclosed in quotes.

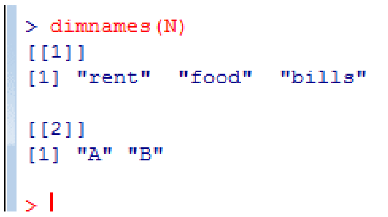
For example suppose we want to create a matrix, N, that contains the expenditure on rent, food and bills for two individuals A and B:

$$N = \begin{array}{r} \\ \text{rent} \\ \text{food} \\ \text{bills} \end{array} \begin{array}{c} \begin{array}{cc} A & B \end{array} \\ \begin{pmatrix} 500 & 750 \\ 100 & 150 \\ 75 & 100 \end{pmatrix} \end{array}$$



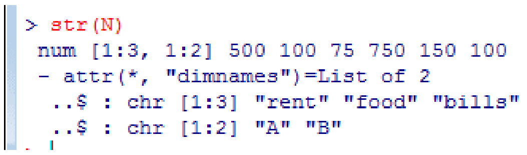
```
> N <- matrix(c(500,100,75,750,150,100),3,2,
+ dimnames=list(c("rent","food","bills"), c("A","B")))
> N
        A   B
rent  500 750
food  100 150
bills  75 100
> 
```

Using the dimnames(<matrix object>) command lists out the row and column names:



```
> dimnames(N)
[[1]]
[1] "rent"  "food"  "bills"

[[2]]
[1] "A" "B"

> 
```

These are also displayed when using the structure command, str(<object>):

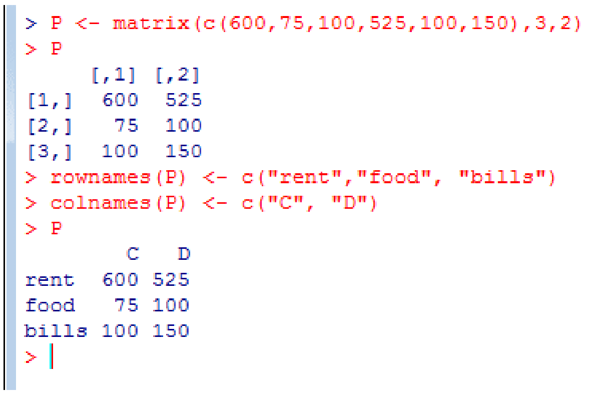


```
> str(N)
 num [1:3, 1:2] 500 100 75 750 150 100
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:3] "rent" "food" "bills"
  ..$ : chr [1:2] "A" "B"
```

An alternative way of naming the rows and columns of a matrix is to use the rownames or colnames functions.

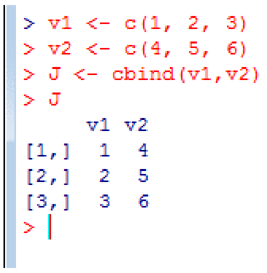
For example suppose we want to now create a matrix, P, that contains the expenditure on rent, food and bills for two different individuals C and D:

$$P = \begin{array}{r} \\ \text{rent} \\ \text{food} \\ \text{bills} \end{array} \begin{array}{c} \begin{array}{cc} C & D \end{array} \\ \begin{pmatrix} 600 & 525 \\ 75 & 100 \\ 100 & 150 \end{pmatrix} \end{array}$$



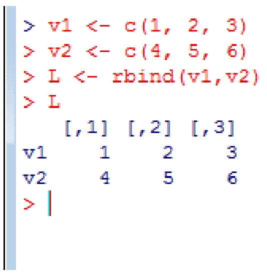
```
> P <- matrix(c(600,75,100,525,100,150),3,2)
> P
     [,1] [,2]
[1,]  600  525
[2,]   75  100
[3,]  100  150
> rownames(P) <- c("rent","food", "bills")
> colnames(P) <- c("C", "D")
> P
        C   D
rent  600 525
food   75 100
bills 100 150
> 
```

Note that if you create matrices using the cbind or rbind functions then the names of those matrices will be used for the columns or rows, respectively:



```
> v1 <- c(1, 2, 3)
> v2 <- c(4, 5, 6)
> J <- cbind(v1,v2)
> J
     v1 v2
[1,]  1  4
[2,]  2  5
[3,]  3  6
> 
```

We see here that the names of the vectors have become the names of the columns.



```
> v1 <- c(1, 2, 3)
> v2 <- c(4, 5, 6)
> L <- rbind(v1,v2)
> L
   [,1] [,2] [,3]
v1    1    2    3
v2    4    5    6
> 
```

And in this second case the names of the vectors have become the names of the rows.

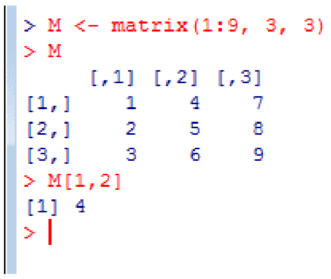
3 Indexing (or subsetting) matrices

Recall that indexing is the process of selecting only some of the elements of an object. We give the position of the element in square brackets after the name of the object.

Since matrices have two dimensions we will need to give the row and column of the element we want. So let's define a bigger matrix:

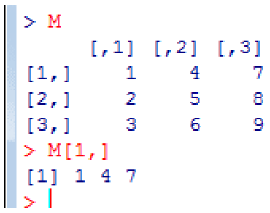
$$\mathbf{M} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

We can choose the element in the first row and second column by writing M[1,2]:



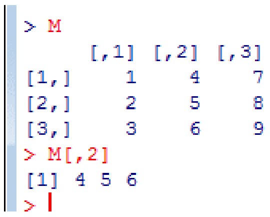
```
> M <- matrix(1:9, 3, 3)
> M
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> M[1,2]
[1] 4
> 
```

To display all the elements of, say, the first row, we omit the figure for the column and type M[1,]:



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

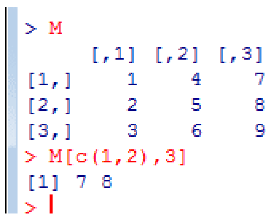
Similarly, to display all the elements of the second column, we omit the figure for the row and type M[,2]:



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

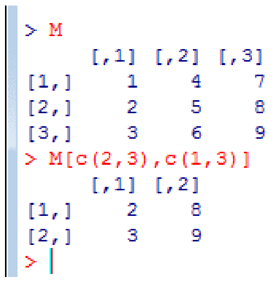
You notice that even though we have selected the column it displays it horizontally as R does for vectors.

To display more than one row or more than one column we simply enter the multiple values using the c( ) command. For example, to display the elements in the 1st and 2nd rows of the 3rd column:



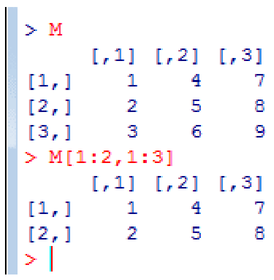
```
> M
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> M[c(1,2),3]
[1] 7 8
> 
```

Or the 2nd and 3rd rows of the 1st and 3rd columns:



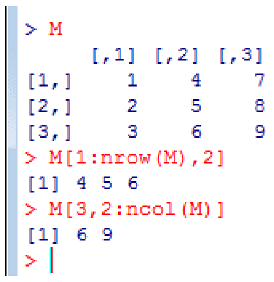
```
> M
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> M[c(2,3),c(1,3)]
     [,1] [,2]
[1,]    2    8
[2,]    3    9
> 
```

We could also select multiple rows or columns using a:b, the consecutive integer function:



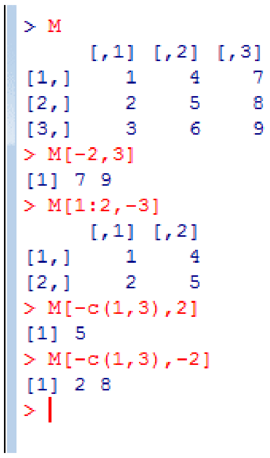
```
> M
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> M[1:2,1:3]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
> 
```

We can use the dimension commands nrow( ) or ncol( ) to specify all the rows or columns until the end:



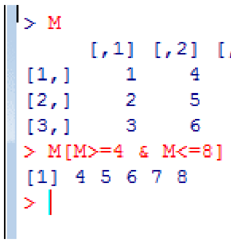
```
> M
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> M[1:nrow(M),2]
[1] 4 5 6
> M[3,2:ncol(M)]
[1] 6 9
> 
```

To specify all row/column elements *except* the specified ones we use a negative in front of their positions:



```
> M
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> M[-2,3]
[1] 7 9
> M[1:2,-3]
     [,1] [,2]
[1,]    1    4
[2,]    2    5
> M[-c(1,3),2]
[1] 5
> M[-c(1,3),-2]
[1] 2 8
> 
```

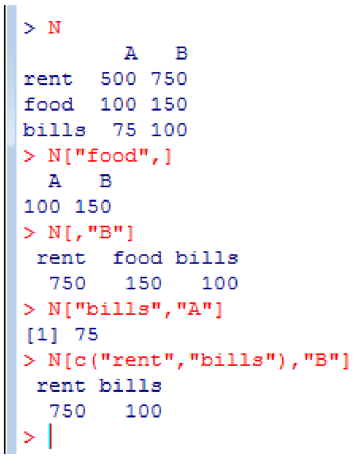
We can select elements using the results of logical tests. For example we could select all the values of M whose value is between 4 and 8 inclusive as follows:



```
> M
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> M[M>=4 & M<=8]
[1] 4 5 6 7 8
> 
```

We can see it returns all the elements from the matrix for which the test is TRUE. These are collected together in a vector. However, it doesn't give their original positions in the matrix.

Finally, if we have named our rows and columns, then we can select the elements using their names. For example, using our matrix N from earlier which had row names of rent, food and bills, and column names of A and B we have:

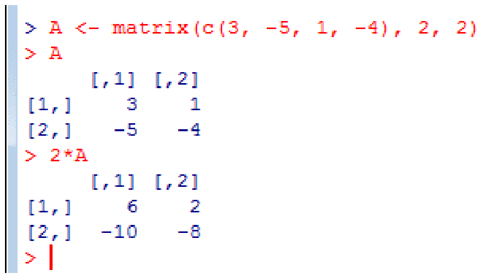


```
> N
        A   B
rent  500 750
food  100 150
bills  75 100
> N["food",]
  A   B
100 150
> N[,"B"]
 rent  food bills
  750   150   100
> N["bills","A"]
[1] 75
> N[c("rent","bills"),"B"]
 rent bills
  750   100
> 
```

4 Matrix arithmetic

We multiply or divide a non-character matrix by a scalar as you'd expect:

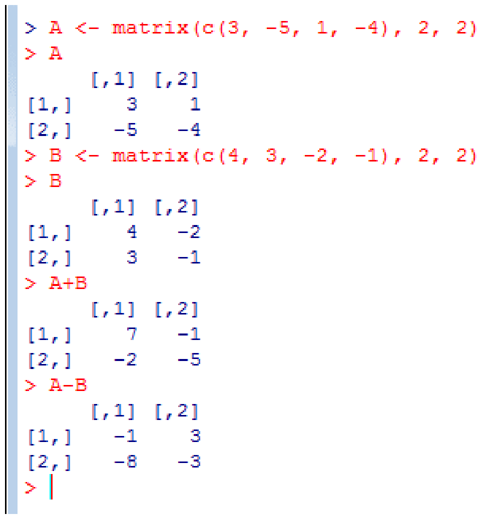
$$\mathbf{A}=\begin{pmatrix} 3 & 1 \\ -5 & -4 \end{pmatrix} \Rightarrow 2\mathbf{A}=\begin{pmatrix} 6 & 2 \\ -10 & -8 \end{pmatrix}$$



```
> A <- matrix(c(3, -5, 1, -4), 2, 2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> 2*A
     [,1] [,2]
[1,]    6    2
[2,]  -10   -8
> 
```

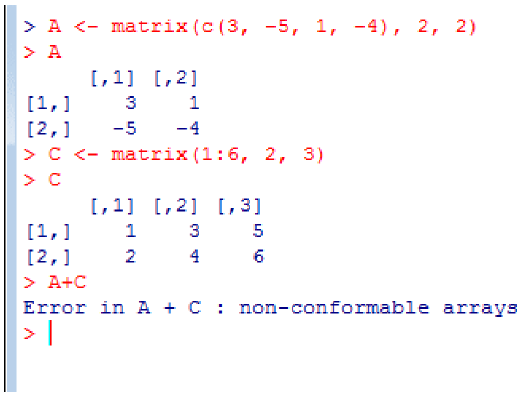
We can add and subtract matrices in the usual way:

$$\mathbf{A}=\begin{pmatrix} 3 & 1 \\ -5 & -4 \end{pmatrix} \quad \mathbf{B}=\begin{pmatrix} 4 & -2 \\ 3 & -1 \end{pmatrix} \Rightarrow \mathbf{A}+\mathbf{B}=\begin{pmatrix} 7 & -1 \\ -2 & -5 \end{pmatrix} \quad \mathbf{A}-\mathbf{B}=\begin{pmatrix} -1 & 3 \\ -8 & -3 \end{pmatrix}$$



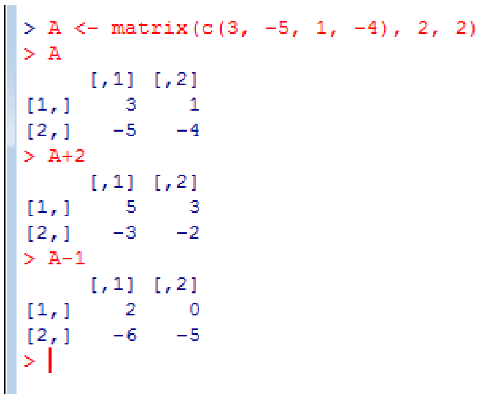
```
> A <- matrix(c(3, -5, 1, -4), 2, 2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> B <- matrix(c(4, 3, -2, -1), 2, 2)
> B
     [,1] [,2]
[1,]    4   -2
[2,]    3   -1
> A+B
     [,1] [,2]
[1,]    7   -1
[2,]   -2   -5
> A-B
     [,1] [,2]
[1,]   -1    3
[2,]   -8   -3
> 
```

Just like in maths, you can only perform this operation on matrices of the same dimensions:



```
> A <- matrix(c(3, -5, 1, -4), 2, 2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> C <- matrix(1:6, 2, 3)
> C
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> A+C
Error in A + C : non-conformable arrays
> 
```

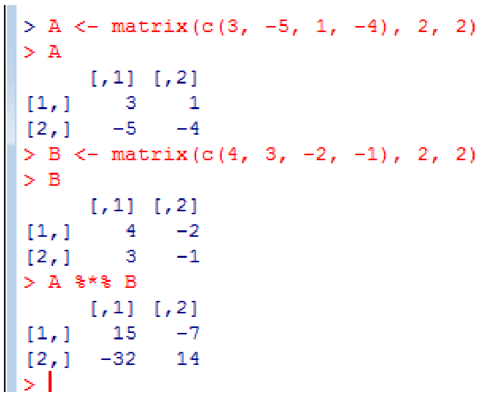
However, unlike maths, if you add or subtract a scalar to a matrix, it does this to each element:



```
> A <- matrix(c(3, -5, 1, -4), 2, 2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> A+2
     [,1] [,2]
[1,]    5    3
[2,]   -3   -2
> A-1
     [,1] [,2]
[1,]    2    0
[2,]   -6   -5
> 
```

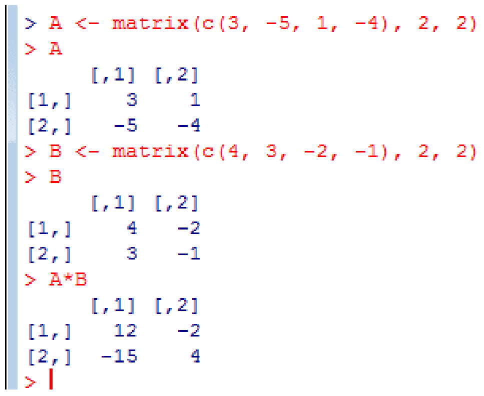
To multiply matrices you should use the operator %*% rather than * :

$$\mathbf{A}=\begin{pmatrix}3&1\\-5&-4\end{pmatrix}\qquad \mathbf{B}=\begin{pmatrix}4&-2\\3&-1\end{pmatrix}\quad\Rightarrow\quad \mathbf{AB}=\begin{pmatrix}(3\times 4)+(1\times 3) & (3\times -2)+(1\times -1)\\ (-5\times 4)+(-4\times 3) & (-5\times -2)+(-4\times -1)\end{pmatrix}=\begin{pmatrix}15&-7\\-32&14\end{pmatrix}$$



```
> A <- matrix(c(3, -5, 1, -4), 2, 2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> B <- matrix(c(4, 3, -2, -1), 2, 2)
> B
     [,1] [,2]
[1,]    4   -2
[2,]    3   -1
> A %*% B
     [,1] [,2]
[1,]   15   -7
[2,]  -32   14
> 
```

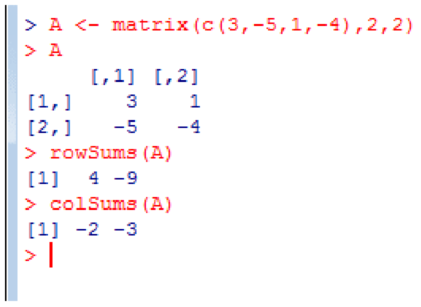
Using just * multiplies the matrices on an element by element basis:



```
> A <- matrix(c(3, -5, 1, -4), 2, 2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> B <- matrix(c(4, 3, -2, -1), 2, 2)
> B
     [,1] [,2]
[1,]    4   -2
[2,]    3   -1
> A*B
     [,1] [,2]
[1,]   12   -2
[2,]  -15    4
> 
```

Row and column sums

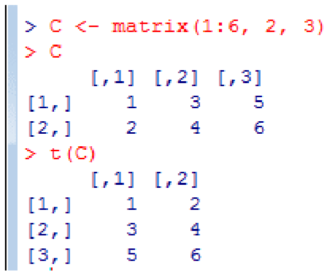
There are some handy functions available in R to calculate the row and column sums of a matrix. These are rowSums(<matrix>) and colSums(<matrix>). Note that the capital letter in Sums is necessary:



```
> A <- matrix(c(3,-5,1,-4),2,2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> rowSums(A)
[1]  4 -9
> colSums(A)
[1] -2 -3
> 
```

Transpose

To obtain the transpose of a matrix we use the t(<matrix object>) function:



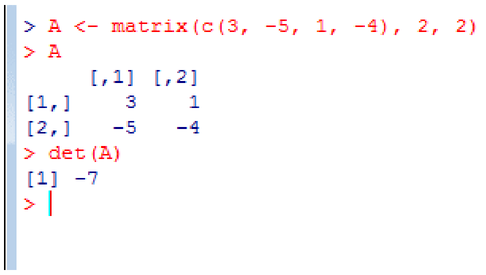
```
> C <- matrix(1:6, 2, 3)
> C
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> t(C)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

5 Other matrix functions

Determinants

These can be found using, unsurprisingly, the det(<matrix object>) function:

$$\mathbf{A} = \begin{pmatrix} 3 & 1 \\ -5 & -4 \end{pmatrix} \Rightarrow \det \mathbf{A} = (3 \times -4) - (1 \times -5) = -7$$


```
> A <- matrix(c(3, -5, 1, -4), 2, 2)
> A
     [,1] [,2]
[1,]    3    1
[2,]   -5   -4
> det(A)
[1] -7
> |
```

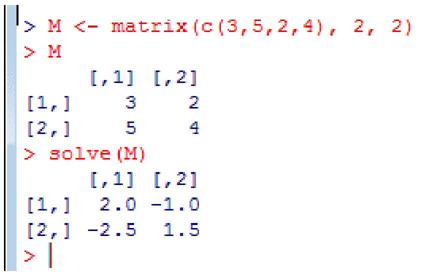
Inverses

We can find the inverse of a matrix using the solve(<matrix object>) function. Recall for a 2×2 matrix:

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \Rightarrow \mathbf{A}^{-1} = \frac{1}{ad - bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Hence, we have:

$$\mathbf{M} = \begin{pmatrix} 3 & 2 \\ 5 & 4 \end{pmatrix} \Rightarrow \mathbf{M}^{-1} = \frac{1}{2}\begin{pmatrix} 4 & -2 \\ -5 & 3 \end{pmatrix}$$


```
> M <- matrix(c(3,5,2,4), 2, 2)
> M
     [,1] [,2]
[1,]    3    2
[2,]    5    4
> solve(M)
     [,1] [,2]
[1,]  2.0 -1.0
[2,] -2.5  1.5
> |
```

The solve function can be used more generally to solve a set of equations of the form:

$$\mathbf{Ax} = \mathbf{b}$$

For example to solve:

$$2x - 3y = 14$$

$$3x + 4y = 4$$

We write the equations in vector/matrix form as follows:

$$\mathbf{Ax} = \mathbf{b}$$

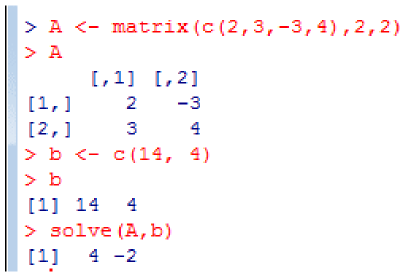
$$\begin{pmatrix} 2 & -3 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 14 \\ 4 \end{pmatrix}$$

Then:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 & -3 \\ 3 & 4 \end{pmatrix}^{-1}\begin{pmatrix} 14 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ -2 \end{pmatrix}$$

To solve this in R we would type solve(A,b):



```
> A <- matrix(c(2,3,-3,4),2,2)
> A
     [,1] [,2]
[1,]    2   -3
[2,]    3    4
> b <- c(14, 4)
> b
[1] 14  4
> solve(A,b)
[1]  4 -2
```

Eigenvalues and eigenvectors

Recall that if $\mathbf{Av} = \lambda\mathbf{v}$ where λ is a constant, then λ is an eigenvalue and $\mathbf{v}$ is an eigenvector of matrix $\mathbf{A}$.

We can obtain the eigenvalues by finding the values of λ for which $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$.

$$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ 2 & 5 \end{pmatrix}$$

$$\det\begin{pmatrix} 2-\lambda & -1 \\ 2 & 5-\lambda \end{pmatrix} = 0$$

$$(2-\lambda)(5-\lambda) + 2 = 0$$

$$\lambda^2 - 7\lambda + 12 = 0$$

$$(\lambda - 3)(\lambda - 4) = 0 \quad\Rightarrow\quad \lambda = 3, 4$$

For $\lambda = 3$ this means we have:

$$\begin{pmatrix} -1 & -1 \\ 2 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad\Rightarrow\quad \begin{pmatrix} -x-y \\ 2x+2y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad\Rightarrow\quad x = -y$$

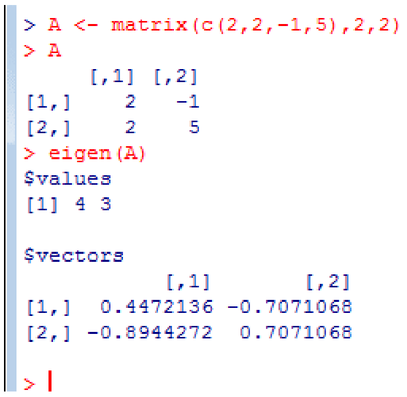
So the eigenvectors corresponding to $\lambda = 3$ are of the form $k\begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

For $\lambda = 4$ this means we have:

$$\begin{pmatrix} -2 & -1 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} -2x - y \\ 2x + y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow y = -2x$$

So the eigenvectors corresponding to $\lambda = 4$ are of the form $k\begin{pmatrix} 1 \\ -2 \end{pmatrix}$.

To obtain the eigenvalues and eigenvectors in R we use the function eigen(<matrix object>):



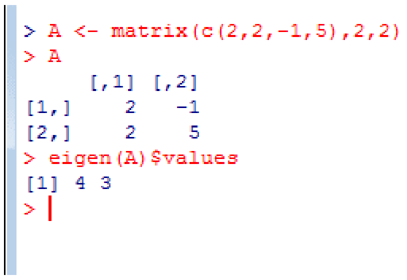
```
> A <- matrix(c(2,2,-1,5),2,2)
> A
     [,1] [,2]
[1,]    2   -1
[2,]    2    5
> eigen(A)
$values
[1] 4 3

$vectors
           [,1]       [,2]
[1,]  0.4472136 -0.7071068
[2,] -0.8944272  0.7071068

> 
```

Notice that it gives the “normalised” version of the eigenvectors – that is a vector with a modulus of 1. So:

$$k\begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix} \quad \text{and} \quad k\begin{pmatrix} 1/\sqrt{5} \\ -2/\sqrt{5} \end{pmatrix}$$

Notice how it says $values and $vectors? If you just wanted the eigenvalues you could type:

eigen(A)$values



```
> A <- matrix(c(2,2,-1,5),2,2)
> A
     [,1] [,2]
[1,]    2   -1
[2,]    2    5
> eigen(A)$values
[1] 4 3
> 
```

Similarly, if we just wanted the eigenvectors we could have typed eigen(A)$vectors.

We’ll discuss this $ notation more in the next chapter on data frames.

6 Summary

Key terms

Matrix	A two-dimensional (row, column) ordered collection of data of the *same type*
Concatenate	Join together
Coercing	The process of changing the data types so that they are all the same
Indexing	The process of selecting an element from a matrix object

Key commands

matrix(<data>, nrow = , ncol = , byrow = FALSE, dimnames = list(<row names>, <col names>))

	Creates a matrix of dimensions nrow by ncol out of the data
	Last two arguments are optional
	By default it fills the matrix by column, change byrow=TRUE to change this to rows
	Use dimnames to give names to the rows and columns
is.matrix(<object>)	Logical test of whether <object> is a matrix
	Returns TRUE or FALSE
dim(<matrix>)	Gives the dimensions (*ie* rows and columns) of <matrix>
nrow(<matrix>)	Gives the number of rows of <matrix>
ncol(<matrix>)	Gives the number of columns of <matrix>
str(<object>)	Displays the structure of the <object>
cbind(<object1>, <object2>, …)	Creates a matrix by combining the objects in columns
rbind(<object1>, <object2>, …)	Creates a matrix by combining the objects in rows
rownames(<matrix>)	Names the rows of the <matrix>
colnames(<matrix>)	Names the columns of the <matrix>
<matrix>[<row>,<column>]	Returns the element from <matrix> given in <row> and <column>.
t(<matrix>)	Returns the transpose of <matrix>
det(<matrix>)	Gives the determinant of <matrix>

%*%	Matrix multiplication operator
solve(<matrix>)	Gives the inverse of <matrix>
solve(**A,b)**	Gives the vector $\mathbf{x}$ that solves $\mathbf{Ax}=\mathbf{b}$.
eigen(<matrix>)	Gives the eigenvalues and eignevectors of <matrix>
rowSums(<matrix>)	Displays a vector of the sums of each row
colSums(<matrix>)	Displays a vector of the sums of each column

7 Have a go

You will only get proficient at R by practising.

1. Create a matrix $\mathbf{C}$:

$$\mathbf{C}=\begin{pmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\end{pmatrix}$$

(a) using the c function

(b) using the colon function

(c) using the seq function

2. Use an appropriate function (other than matrix( )) to create a new matrix, $\mathbf{D}$, which is the same as matrix $\mathbf{C}$ but with an additional row containing the elements 13, 14, 15, 16.

3. What type of matrix (numeric, logical, character) would be formed from each of the following:

(a) matrix(c(3, FALSE, 1, TRUE), 2, 2)

(b) matrix(c(TRUE, “bob”, FALSE, “harry”), 2, 2)

(c) matrix(c(“larry”, 7, 2, -1), 2, 2)

(d) matrix(c(5, TRUE, “ginger”, 0), 2, 2)

Check your answers using the is.numeric, is.logical, is.character functions.

4. (a) Create the following named matrix of temperatures using the dimnames option:

$$\mathbf{T}=\begin{array}{c} \\ \text{London}\\ \text{Paris}\\ \text{Stockholm}\end{array}\begin{array}{c}\begin{array}{cc}\textit{Mon} & \textit{Tue}\end{array}\\ \begin{pmatrix}18 & 20\\ 20 & 19\\ 15 & 13\end{pmatrix}\end{array}$$

(b) Create the following named matrix of indices using rownames and colnames functions:

$$\mathbf{T}=\begin{array}{c} \\ \text{FTSE}\\ \text{Dow Jones}\\ \text{Nikkei}\end{array}\begin{array}{c}\begin{array}{cc}\textit{Mon} & \textit{Tue}\end{array}\\ \begin{pmatrix}6{,}125.7 & 6{,}078.2\\ 17{,}140.2 & 17{,}160.7\\ 15{,}323.1 & 15{,}323.8\end{pmatrix}\end{array}$$

5. Use indexing on the matrix **C** from Q1 to display:

   (a) the element 7

   (b) the third row

   (c) the first column

   (d) all but the 2nd row

   (e) a 2×2 matrix of the bottom right $\begin{pmatrix} 7 & 8 \\ 11 & 12 \end{pmatrix}$

   (f) the 2nd and 3rd columns of the 1st and 2nd rows

   (g) all elements which are smaller than 10.

6. Create the following matrix in R:

$$\mathbf{M} = \begin{pmatrix} 1 & -3 \\ 2 & 5 \\ 0 & 9 \end{pmatrix}$$

   (a) Use R to find its dimensions, $3\mathbf{M}$, $\mathbf{M}^T$ and $\mathbf{M}\mathbf{M}^T$.

   (b) Perform an operation in R to subtract 3 from each element in matrix $\mathbf{M}$.

   (c) Use R to obtain the column sums and row sums for matrix $\mathbf{M}$.

7. Create the following matrices in R:

$$\mathbf{A} = \begin{pmatrix} 1 & 9 \\ 2 & 10 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 1 & 2 \\ 3 & -4 \end{pmatrix}$$

   (a) Use R to calculate $\mathbf{AB}$ and $\mathbf{BA}$. Hence show that matrices are not commutative (*ie* $\mathbf{AB} \neq \mathbf{BA}$).

   (b) Use R to find its determinant, $\mathbf{B}^{-1}$ and hence show that $\mathbf{B}\mathbf{B}^{-1} = \mathbf{I}$.

   (c) Use R to find the eigenvalues and eigenvectors of $\mathbf{B}$.

8. Use matrices in R to solve the following set of simultaneous equations:

$$2p - 4q = -14$$

$$2q - 3p = 13$$