Working with matrices in R

Philip Dixon
15 Feb 2018

Vectors, scalars, and matrices:

- Most data in R are stored in vectors. Data frames are collections of the vectors of the same length. That means a data frame looks like a matrix (has rows and columns) and can often by manipulated as if it is a matrix, but it isn't. The most important distinction is that a matrix is all numbers or all character strings but a data frame can be a mix.

  ```
  > temp <- data.frame(a=1:5, b=c(1,6,4,2,23), c=c('1','2','< 5', '4','3'))
  > temp

    a  b   c
  1 1  1   1
  2 2  6   2
  3 3  4 < 5
  4 4  2   4
  5 5 23   3

  > temp3 <- as.matrix(temp)
  > temp3

       a   b    c
  [1,] "1" " 1" "1"
  [2,] "2" " 6" "2"
  [3,] "3" " 4" "< 5"
  [4,] "4" " 2" "4"
  [5,] "5" "23" "3"

  > temp2 <- as.matrix(temp[,1:2])
  > temp2

       a  b
  [1,] 1  1
  [2,] 2  6
  [3,] 3  4
  [4,] 4  2
  [5,] 5 23
  ```

  temp has 2 columns of numbers and one of characters. Even though most values of the *c* variable are numbers, the value of `< 5` forces the column to be character strings.

when the temp data frame is converted to the temp3 matrix, everything becomes character strings, because that is what's necessary to represent all three columns.
when only the first two columns are converted to the temp2 matrix, the matrix is numeric.

- A scalar, e.g. the number 5, is really a vector with length 1.
- R is smart about arithmetic operations on combinations of vectors. It tries to do what is logical.

  ```
  > a <- 1:4
  > b <- c(10,20)
  > a + 5
  ```

  ```
  [1] 6 7 8 9
  ```

  ```
  > a + b
  ```

  ```
  [1] 11 22 13 24
  ```

  e.g., by replicating shorter vectors when necessary.

  ```
  > a2 <- 1:5
  > a2 + b
  ```

  ```
  [1] 11 22 13 24 15
  ```

  and giving a warning in the console window when you might be asking for something funny. Notice that vectors are printed across the row.

- A matrix has rows and columns. Both can be named if desired (see ?dim.names)
- You can create a matrix:
  - from a data frame, using as.matrix(). `temp3 <- as.matrix(temp)` If some columns of the data frame are meant as identifying information (e.g. species names in a species by sites abundance matrix), make sure to omit those columns when creating the matrix.
  - by concatenating rows, using rbind(), or columns, using cbind(). You can label columns or rows if desired.

    ```
    > cbind(1:3, c(3,7,20))
    ```

    ```
         [,1] [,2]
    [1,]    1    3
    [2,]    2    7
    [3,]    3   20
    ```

    ```
    > cbind(a=1:3, b=c(3,7,20))
    ```

    ```
         a  b
    [1,] 1  3
    [2,] 2  7
    [3,] 3 20
    ```

```
> rbind(c(1,3), c(2,7), c(3,20))

     [,1] [,2]
[1,]    1    3
[2,]    2    7
[3,]    3   20
```

– using matrix(). You provide a vector of the numbers and matrix() will reshape that to the desired matrix. nrow= and ncol= specify the number of rows and number of columns. This is an especially easy way to create a matrix with all the same value, because short vectors are replicated when necessary.

```
> matrix(0, nrow=2, ncol=3)

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

> matrix(1:6, nrow=2)

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

> matrix(1:6, ncol=3)

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

> matrix(1:6, nrow=2, byrow=T)

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

> matrix(1:3, nrow=2, ncol=3, byrow=T)

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

Notice that matrix() "fills" the matrix by going down the columns. `byrow=T` tells matrix() to go across the rows when filling a matrix. When one of the matrix dimensions is implicit (e.g., the 3rd example, where 6 values are given to fill a matrix with 2 rows, or the 4th example, where 6 values are given to fill a matrix with 2 columns, the missing dimension is assumed. The last piece of code creates a matrix with 6 values (2 rows, 3 columns) but the vector to fill it only has 3 values. That is replicated, which means that each row has 1, 2, 3.

– using diag() to create a diagonal matrix from specified values

```
> diag(1:4)

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

```
[2,]    0    2    0    0
[3,]    0    0    3    0
[4,]    0    0    0    4
```

```
> diag(rep(1, 4))
```

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

The specified values are put along the diagonal, the rest of the matrix is 0. diag() can also extract the diagonal. The result is a vector.

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

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

```
> diag(a)
```

```
[1] 1 5 9
```

- Matrix arithmetic
  - addition and subtraction are element-by-element. Matrix with vector operations are allowed and always "go down the columns", so `a+d` below adds 0 to the value in the first row of each column, 0 to the value in the second row, and 1 to the value in the third row. Operations with a scalar apply to all values in the matrix.

    ```
    > a <- matrix(1:9, nrow=3)
    > b <- diag(rep(1,3))
    > a + b
    ```

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

    ```
    > a - b
    ```

    ```
         [,1] [,2] [,3]
    [1,]    0    4    7
    [2,]    2    4    8
    [3,]    3    6    8
    ```

    ```
    > d <- c(0,0,1)
    > a + d
    ```

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

```
> a + 1
```

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

– R checks that the objects are conformable. Here there is a difference between a scalar and a 1x1 matrix. A scalar can be added to all elements, a 1x1 matrix can only be added to another 1x1 matrix. You get an error on the console about non-conformable arrays. When necessary, as.vector() will convert the 1x1 matrix back to a scalar (actually a vector of length 1).

```
> a <- matrix(1:9, nrow=3)
> b <- 1
> b2 <- matrix(b, nrow=1)
> a + b
```

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

```
> #  a + b2  # not run, because generates an error
> b2
```

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

```
> as.vector(b2)
```

```
[1] 1
```

```
> a + as.vector(b2)
```

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

– multiplication and division using `*` and `/` are element-by-element. Again, shorter vectors are replicated.

```
> a <- matrix(1:9, nrow=3)
> b <- diag(rep(1,3))
> d <- c(0,0,1)
> a * b
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    5    0
[3,]    0    0    9
```

```
> a * d
```

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    3    6    9
```

```
> b / a
```

```
     [,1] [,2]      [,3]
[1,]    1  0.0 0.0000000
[2,]    0  0.2 0.0000000
[3,]    0  0.0 0.1111111
```

– %*% is the matrix multiplication operator. The matrices have to be conformable. Since matrix multiplication is the sum of across the row times down the column, the number of columns of the first matrix must equal the number of rows of the second

```
> a <- matrix(1:9, nrow=3)
> b <- diag(rep(1,3))
> d <- matrix(c(0,0,1), nrow=3)
> a %*% b
```

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

```
> a %*% d
```

```
     [,1]
[1,]    7
[2,]    8
[3,]    9
```

```
> #  d %*% a  # not run because generates an error
```

– t() is the matrix transpose function.

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

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

```
> t(a)
```

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

`a %*% d` is computable because a has 3 columns while d has 3 rows. `d %*% a` is not, because computable because d has 1 column, which does not match the 3 rows of a.

– matrix division does not exist, instead you multiply by the inverse. solve() computes the matrix inverse.

```
> a <- rbind(c(10,5,5), c(5,10,2), c(5,2,10))
> ainv <- solve(a)
> ainv
```

```
            [,1]         [,2]         [,3]
[1,]  0.17142857 -0.071428571 -0.071428571
[2,] -0.07142857  0.133928571  0.008928571
[3,] -0.07142857  0.008928571  0.133928571
```

```
> a %*% ainv
```

```
             [,1]         [,2]         [,3]
[1,] 1.000000e+00 2.116363e-16 2.775558e-17
[2,] 1.110223e-16 1.000000e+00 5.551115e-17
[3,] 1.665335e-16 9.020562e-17 1.000000e+00
```

```
> round(a %*% ainv, 6)
```

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

```
> round(ainv %*% a, 6)
```

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