## 13.1 *R* Tutorial 7

*Introduction to Computational Science:*
*Modeling and Simulation for the Sciences, 2nd Edition*
Angela B. Shiflet and George W. Shiflet
Wofford College


### Introduction

We recommend that you work through this tutorial with a copy of *R*, answering all Quick Review Questions. *R* is available from http://www.r-project.org.

The prerequisites to this tutorial are *R* Tutorials 1-6. For those starting their use of *R* with Chapter 13, refer instead to Alternative Tutorial 7, *RCTTutorial7Alt.m*, which has no prerequisite. Tutorial 7 prepares you to use *R* to complete projects for this chapter. The tutorial introduces the following functions and concepts: *dot*, matrix addition and multiplication, *any*, *all*, *eig*, *unique*, *sortrows*, and *union*. The module also gives examples along with Quick Review Questions for you to do with *R*. Execute all input cells to view the results of the examples.

### Matrix Operations for Modules 13.2-13.4

When two matrices have the same size, we can perform the **+** or * operation. The resulting matrix has the same size and each element of the result is the sum or product, respectively, of corresponding elements of the two operands. The following commands record the elements of matrices one row at a time using, ***byrow = TRUE***, and give examples of these operators:

```
matA = matrix(c(3, 6, 5, -2, 0, 3), nrow = 2, byrow = TRUE)
matB = matrix(c(4, -1, 0, 7, 8, 3), nrow = 2, byrow = TRUE)
matA + matB
matA * matB
```

As the results show, adding element by element, the sum *matA* + *matB* = $\begin{bmatrix} 3 & 6 & 5 \\ -2 & 0 & 3 \end{bmatrix}$ + $\begin{bmatrix} 4 & -1 & 0 \\ 7 & 8 & 3 \end{bmatrix}$ is $\begin{bmatrix} 7 & 5 & 5 \\ 5 & 8 & 6 \end{bmatrix}$. Similarly, multiplying element-by-element for *matA* .* *matB*, we obtain $\begin{bmatrix} 12 & -6 & 0 \\ -14 & 0 & 9 \end{bmatrix}$.

For vectors that have the same number of elements, we can perform the **dot product**, which returns a number, the sum of the product of corresponding elements. The *R* function ***sum*** performs the operation on the elements of a vector, which is the element-by-element product of two vectors, accomplished with *. Thus, the following command returns the dot product of [2 7 -1] and [5 3 4], which is 2 · 5 + 7 · 3 + -1 · 4 = 10 + 21 -4 = 27:

```
sum(c(2, 7, -1) * c(5, 3, 4))
```

The matrix-times-vector or matrix product operator is *. Consider vector *vecCol* = $\begin{bmatrix} 7 \\ 1 \\ -2 \end{bmatrix}$, which the following command creates:

```
vecCol = c(7, 1, -2)
```

The product of *matA* and *vecCol*, *matA* **%*%** *vecCol*, produces a 2-by-1 column vector, $\begin{bmatrix} 17 \\ -20 \end{bmatrix}$. With *matC* = $\begin{bmatrix} 7 & -3 \\ 1 & 0 \\ -2 & 6 \end{bmatrix}$, the following is an example of matrix multiplication, producing the result $\begin{bmatrix} 27 & 42 & 26 \\ 3 & 6 & 5 \\ -18 & -12 & 8 \end{bmatrix}$:

```
matC = matrix(c(7, -3, 1, 0, -2, 6), nrow = 3, byrow = TRUE)
matC %*% matA
```

**Quick Review Question 1** Start a new *Script*. In opening comments, have "R Tutorial 7 Answers" and your name. Save the file under the name *RCTTutorial7Ans.R*. In the file, preface this and all subsequent Quick Review Questions with a comment that has "## QRQ" and the question number, such as follows:

```
## QRQ 1 a
```

**a.** Generate a 4-by-2 matrix *mA*, where the *i*-*j* element is the sum of *i* and *j*. Thus, after forming *mA*, *mA*[3, 2] should be 5, which is 3 + 2.
**b.** Generate a 4-by-2 matrix *mO* of all ones.
**c.** Give matrix sum of *mA* and *mO*.
**d.** Define a vector *u* with elements 2 and 7.
**e.** Define a vector *v* with elements 5 and 3.
**f.** Give dot product of *u* and *v*.
**g.** Generate a 2-by-3 matrix *mB*, where the *i*-*j* element is the difference of *i* and *j*, *i* - *j*.
**h.** Give the matrix product of *mA* and *mB*.

We can take the matrix product of a square matrix by itself, which accomplishes the matrix power operation. For example, suppose *mS* is defined as follows:

```
mS = matrix(c(8, 4, 6, 7, 7, 1, 4, 6, 2), nrow = 3, byrow = TRUE)
```

The matrix product of *mS* with itself, *mS* %*% *mS*, is the 3 × 3 matrix $\begin{bmatrix} 116 & 96 & 64 \\ 109 & 83 & 51 \\ 82 & 70 & 34 \end{bmatrix}$. Mathematically, we can generate the same result by squaring *mS*, $mS^2$. In *R*, for larger powers, such as 5, we can start by assigning the identity matrix, ***diag*(3)**,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

to a variable, such as *prod*, and with a *for* loop repeatedly multiply by *mS*, as follows:

```
prod = diag(3)
for (i in 1:5) {
    prod = prod %*% mS
}
```

**Quick Review Question 2**

**a.** Generate a 3 × 3 matrix *mC*, where the *i*-*j* element is 2*i* - *j*.
**b.** Calculate the matrix power $mC^7$ using a loop as above.
**c.** Show that the calculation of the matrix product of *mC* with itself seven times has the same value as the answer to Part b.

To test if all elements of a matrix or vector satisfy a condition, we employ the command ***all***. Similarly, to test if any of these elements satisfy a condition, we use ***any***. Thus, the answer to the following segment is *TRUE*:

```
b = c(3, 5, 2)
all(b == b)
```

Moreover, because 5 > 4, the following command returns *TRUE*:

```
any(b > 4)
```

## Eigenvalues and Eigenvectors for Modules 13.3-13.4

For square matrix *M*, the constant λ is an **eigenvalue** and *v* is an **eigenvector** if multiplication of the constant by the vector accomplishes the same results as multiplying the matrix by the vector, that is, the following equality holds:

$$Mv = \lambda v$$

The **dominant eigenvalue** for a matrix is the largest eigenvalue for that matrix. The *R* function ***eigen*** returns a vector containing the eigenvalues and a matrix with the associated eigenvectors for a square matrix argument, as the following illustrates:

```
mat = matrix(c(0, 3, 6, 0.1, 0, 0, 0, 0.4, 0), nrow=3, byrow=TRUE)
eigen(mat)

$values
[1]  0.7796379+0.0000000i -0.3898189+0.3948119i -0.3898189-0.3948119i

$vectors
                  [,1]                    [,2]                    [,3]
[1,] 0.98976800+0i -0.9761933+0.0000000i -0.9761933+0.0000000i
[2,] 0.12695227+0i  0.1236176+0.1252010i  0.1236176-0.1252010i
[3,] 0.06513397+0i  0.0016143-0.1268359i  0.0016143+0.1268359i
```

In this case, two of the eigenvalues are complex numbers, and the dominant eigenvalue is the first element of lambda, 0.7796379. (0.0000000i is 0.) The corresponding eigenvectors are in the columns of *$vectors*. Thus, the corresponding eigenvector,

(0.98976800, 0.12695227, 0.06513397), of the dominant eigenvalue is the first column of *$vectors*, ignoring 0i.

To place the eigenvectors in separate variables, we store the results in a variable and employ ***$values*** and ***$vectors***, as follows:

```
eig = eigen(mat)
lambdaLst = eig$values
vecLst = eig$vectors
```

The following illustrates that the matrix-vector product of *mat* and this vector equals the dominant eigenvalue times the vector.

```
lambda = lambdaLst[1]
v = vecLst[,1]
mat %*% v
lambda * v
```

Each of the products returns the vector $\begin{bmatrix} 0.7717 \\ 0.0990 \\ 0.0508 \end{bmatrix}$.

**Quick Review Question 3**

**a.** Define the matrix *mD* as $\begin{bmatrix} 4 & 6 \\ 3 & 1 \end{bmatrix}$.

**b.** Return a list of eigenvalues of *mD* without calculating eigenvectors.

**c.** Calculate a list of eigenvalues and eigenvectors of *mD*, and store the results in variables *lLst* and *vLst*, respectively.

**d.** Verify that the matrix-vector product of *mD* and the dominant eigenvalue equals the product of that eigenvalue and the corresponding eigenvector.

## Additional Commands Used in Module 13.5

In this section, we consider several *R* commands that are used in the file associated with Module 13.5.

As execution of the following shows, the *R* command ***unique*** with a list argument returns a list with the same elements but without duplicates, in this case (1, 5, 4). The function does not change the argument, *lst*.

```
lst = c(1, 5, 5, 4, 1, 5)
unique(lst)
lst
```

The function ***union*** returns the sorted union of two list arguments with no duplicates. Thus, output of the following is the list 1 5 4 8 7 3:

```
lst1 = c(1, 5, 5, 4, 1, 5)
lst2 = c(4, 8, 7, 8, 3)
union(lst1, lst2)
```

Using the command ***order***, we can sort rows of a matrix in ascending or descending order based on a particular column. Consider the matrix $\begin{bmatrix} 5 & 4 & 5 \\ 1 & 7 & 8 \\ 2 & 3 & 9 \\ 3 & 5 & 6 \end{bmatrix}$, which we define as follows:

```
triples = matrix(c(5, 4, 5, 1, 7, 8, 2, 3, 9, 3, 5, 6),
                 nrow = 4, byrow = TRUE)
```

To sort in ascending order, based on the values in the second column, in brackets after the matrix name, we use *order* with an argument of the second column and a comma, as follows:

```
triples[order(triples[,2]), ]
```

The sorted output, with the second column elements in ascending order, is as follows:

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

With a negative in front of the argument for *order*, the sort is in descending order. Consider the following command:

```
triples[order(-triples[,2]), ]
```

The results reveal a sorting on the second column in descending (reverse) order:

```
ans =

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

**Quick Review Question 4**

**a.** Define a list (matrix), *lst1*, of a five numbers, where each element is a random integer between 0 and 2. Display *lst1*.

**b.** Using *unique* and an assignment statement, eliminate the duplicate pairs in *lst1*. Display *lst1*after the *unique* command.

**c.** Form another list, *lst2*, of 5 numbers, where each element is a random integer between 10 and 12. Assign the union of *lst1* and *lst2* to *lst3*. Display *lst2* and *lst3*.

**d.** Define a list, *dup1*, of a five ordered pairs of numbers, where the first element in each pair is a random integer between 0 and 2 and the second element is a random integer between 10 and 12. Write a command to return *dup1*, sorted by the first elements from highest to lowest.