Matrix Algebra for Multivariate Analysis Using R

last revised October 24, 2012

Introduction

In these notes, we provide a brief survey and review of select and essential matrix operations using R. Once the student overcomes the initial learning curve (it can be quite steep at times) for performing these operations using R, he or she will experience a tremendous wealth of computing power flexibility at his or her hands, unlike what could ever be experienced using popular packages such as SPSS or SAS. This is not to say that SPSS and SAS are not excellent and sophisticated packages in their own right. They are. Working with R however, the user becomes the body of knowledge, the "stimuli," for which the computer is the response. The user can input even a single request, and immediately receives a response. This is a very powerful way to learn multivariate analysis along with its numerous matrix operations and computations.

Matrices

Recall that a matrix is simply an ordered array of numbers, denoted by *n* rows and *m* columns. For instance, the following matrix **A** is a 3-row by 2-column matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

The contents of the matrix consist of elements a_{11}, a_{12}, a_{21}, etc. A matrix is square if the number of *n* rows equal the number of *m* columns. For example, the matrix **B** is a square matrix:

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Building Matrices in R

We can generate a matrix in R quite easily. For instance, suppose we wished to produce a matrix containing a sequence of numbers from 1 to 8 having 2 rows and 3 columns. We can use the function matrix in R for this purpose. For example, let us define the matrix **S**:

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

The code for the above assigns the matrix to the variable **S**. The matrix is defined by entries 1 through 8 (1:8), having 4 rows and 2 columns. We could expand on the above code and generate a wide variety of matrices of various dimensions. We could also change the contents of the matrix. For intance, suppose that instead of inputing entries from 1 to 8, we chose instead to input entries from 20 to 27. We would have:

```
> S <- matrix(20:27, 4, 2)
> S
     [,1] [,2]
[1,]   20   24
[2,]   21   25
[3,]   22   26
[4,]   23   27
```

Suppose we were to now add two new elements to the matrix, elements 28 and 29. To accommodate these new elements, we would need to expand on the numbers of rows in the matrix. Instead of having 4 rows, we would now have 5:

```
> S <- matrix(20:29, 5, 2)
> S
     [,1] [,2]
[1,]   20   25
[2,]   21   26
[3,]   22   27
[4,]   23   28
[5,]   24   29
```

Constructing a Covariance/Correlation Matrix

Though there are many ways to build matrices in R, we demonstrate a simple way of producing a matrix. Consider the following lower-triangular correlation matrix:

```
1.00000
.343      1.00000
.505       .203   1.00000
.308       .400    .398  1.00000
.693       .187    .303   .205   1.00000
.208       .108    .277   .487    .200  1.00000
.400       .386    .286   .385    .311   .432  1.00000
.455       .385    .167   .465    .485   .310   .365 1.00000
```

We build this matrix in R by first "concatenating" the row vectors. By defining each row as r1, r2, r3, etc., and by specifying c to concatenate each vector, we build the vectors for the matrix:

```
> r1 <- c(1.000, 0.343, 0.505, 0.308, 0.693, 0.208, 0.400, 0.455)
> r2 <- c(0.343, 1.000, 0.203, 0.400, 0.187, 0.108, 0.386, 0.385)
> r3 <- c(0.505, 0.203, 1.000, 0.398, 0.303, 0.277, 0.286, 0.167)
> r4 <- c(0.308, 0.400, 0.398, 1.000, 0.205, 0.487, 0.385, 0.465)
> r5 <- c(0.693, 0.187, 0.303, 0.205, 1.000, 0.200, 0.311, 0.485)
> r6 <- c(0.208, 0.108, 0.277, 0.487, 0.200, 1.000, 0.432, 0.310)
> r7 <- c(0.400, 0.386, 0.286, 0.385, 0.311, 0.432, 1.000, 0.365)
> r8 <- c(0.455, 0.385, 0.167, 0.465, 0.485, 0.310, 0.365, 1.000)
```

Recall that since in a correlation or covariance matrix the lower triangular of the matrix is identical to the upper triangular, it is a simple matter to insert the correct off-diagonal correlations in producing the above column vectors.

We then use the rbind function (i.e., "bind the rows together") to join the rows of the above vectors, identifying the new matrix as cormatrix:

```
> cormatrix <- rbind(r1, r2, r3, r4, r5, r6, r7, r8)
```

To verify that we built the matrix correctly, we request it by its name:

```
 > cormatrix
    [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
c1 1.000 0.343 0.505 0.308 0.693 0.208 0.400 0.455
c2 0.343 1.000 0.203 0.400 0.187 0.108 0.386 0.385
c3 0.505 0.203 1.000 0.398 0.303 0.277 0.286 0.167
c4 0.308 0.400 0.398 1.000 0.205 0.487 0.385 0.465
c5 0.693 0.187 0.303 0.205 1.000 0.200 0.311 0.485
c6 0.208 0.108 0.277 0.487 0.200 1.000 0.432 0.310
c7 0.400 0.386 0.286 0.385 0.311 0.432 1.000 0.365
c8 0.455 0.385 0.167 0.465 0.485 0.310 0.365 1.000
```

We can see that the generated cormatrix was correctly built, and is now ready for future analyses.

Building the Matrix by Concatenating Columns Instead of Rows

We could have just as easily built cormatrix by concatenating columns instead of rows. To proceed this way, we build each vector by their respective columns. To make things easier, we name each column vector 1, 2, 3, etc., by c1, c2, c3, etc., instead of r1, r2, r3, which was used to designate rows:

```
> c1 <- c(1.000, 0.343, 0.505, 0.308, 0.693, 0.208, 0.400, 0.455)
> c2 <- c(0.343, 1.000, 0.203, 0.400, 0.187, 0.108, 0.386, 0.385)
> c3 <- c(.505, 0.203, 1.000, 0.398, 0.303, 0.277, 0.286, 0.167)
> c4 <- c(0.308, 0.400, 0.398, 1.000, 0.205, 0.487, 0.385, 0.465)
> c5 <- c(0.693, 0.187, 0.303, 0.205, 1.000, 0.200, 0.311, 0.485)
> c6 <- c(0.208, 0.108, 0.277, 0.487, 0.200, 1.000, 0.432, 0.310)
> c7 <- c(0.400, 0.386, 0.286, 0.385, 0.311, 0.432, 1.000, 0.365)
> c8 <- c(0.455, 0.385, 0.167, 0.465, 0.485, 0.310, 0.365, 1.000)
```

We then use the cbind function instead of the rbind function to join the column vectors, assigning the matrix once again the name cormatrix:

```
> cormatrix <- cbind(c1, c2, c3, c4, c5, c6, c7, c8)
> cormatrix
        c1    c2    c3    c4    c5    c6    c7    c8
[1,] 1.000 0.343 0.505 0.308 0.693 0.208 0.400 0.455
[2,] 0.343 1.000 0.203 0.400 0.187 0.108 0.386 0.385
[3,] 0.505 0.203 1.000 0.398 0.303 0.277 0.286 0.167
[4,] 0.308 0.400 0.398 1.000 0.205 0.487 0.385 0.465
[5,] 0.693 0.187 0.303 0.205 1.000 0.200 0.311 0.485
[6,] 0.208 0.108 0.277 0.487 0.200 1.000 0.432 0.310
[7,] 0.400 0.386 0.286 0.385 0.311 0.432 1.000 0.365
[8,] 0.455 0.385 0.167 0.465 0.485 0.310 0.365 1.000
```

Because a covariance or correlation matrix is a symmetric matrix, generating it by concatenating rows or columns results in the same matrix.

Transpose & Trace

The *transpose* of a matrix is a process by which one interchanges the rows of a matrix to columns. For instance, the transpose of matrix $\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$ we denote as **B'**:

$$\mathbf{B}' = \begin{bmatrix} b_{11} & b_{21} \\ b_{12} & b_{22} \end{bmatrix}$$

The *trace* of a square matrix is the sum of elements along the main diagonal. For instance, the trace of the 3 x 3 matrix **A**

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

is equal to $a_{11} + a_{22} + a_{33}$. The trace of non-square matrices is not defined. Because elements that make up the trace of a covariance matrix represents variances, the computation of the trace is common in multivariate analysis.

We can easily demonstrate in R some useful results in matrix and linear algebra involving transposes and traces. First, we define two matrices **A** and **B**.

Let matrix $\mathbf{A} = \begin{bmatrix} 5 & 9 \\ 3 & 4 \end{bmatrix}$ and let matrix $\mathbf{B} = \begin{bmatrix} 1 & 3 \\ 7 & 2 \end{bmatrix}$.

Using cbind after producing the relevant column vectors, we generate these matrices in R as follows:

```
> a1 <- c(5, 3)
> a2 <- c(9, 4)
> A <- cbind(a1, a2)
> A
     a1 a2
[1,]  5  9
[2,]  3  4
```

```
> b1 <- c(1, 7)
> b2 <- c(3, 2)
> B <- cbind(b1, b2)

     b1 b2
[1,]  1  3
[2,]  7  2
```

Having generated the matrices A and B, we can now demonstrate a few of the more common results involving traces and transposes:

1. $\mathbf{tr}(\mathbf{A}+\mathbf{B}) = \mathbf{tr}(\mathbf{A}) + \mathbf{tr}(\mathbf{B})$

The above reads that the trace of a sum is equal to the sum of the traces.

In R, we request the trace of a matrix by t. We first request the trace of the sum **A** + **B**:

```
> tr(A + B)
[1] 12
```

The above represents the left-hand side of the result. Computing the right-hand side, we find that the above sum is equal to

```
> tr(A) + tr(B)
[1] 12
```

and hence we have demonstrated that $\mathbf{tr}(\mathbf{A}+\mathbf{B}) = \mathbf{tr}(\mathbf{A}) + \mathbf{tr}(\mathbf{B})$.

2. $(\mathbf{A}+\mathbf{B})' = \mathbf{A}' + \mathbf{B}'$

The above reads that the transpose of a sum is equal to the sum of transposes. We first generate the transpose of the sum **A** + **B**:

```
> t(A+B)
   [,1] [,2]
a1    6   10
a2   12    6
```

We then request the sum of the transposes of **A** and **B**:

```
> t(A) + t(B)
   [,1] [,2]
a1    6   10
a2   12    6
```

Hence, we have demonstrated that $(\mathbf{A}+\mathbf{B})' = \mathbf{A}' + \mathbf{B}'$.

3. $(\mathbf{AB})' = \mathbf{B}'\mathbf{A}'$

The above reads that the transpose of a product is equal to the transpose of products in reverse order.

To multiply matrices in R, we use the operator %*%. Thus, for the expression on the left-hand side, we compute the transpose of the matrix product **AB**:

```
> t(A%*%B)
   [,1] [,2]
b1   68   31
b2   33   17
```

The expression on the right-hand side is computed as the transpose of **B** multiplied by the transpose of **A**:

```
> t(B)%*%t(A)
   [,1] [,2]
b1   68   31
b2   33   17
```

Identity Matrices

An identity matrix is defined as a matrix having zeros everywhere except the main diagonal which has elements equal to 1.

For instance, a 3x3 identity matrix is given as

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can generate a 3x3 identity matrix **I** in R as:

```
> I <- matrix(0, nrow = 3, ncol = 3)
> I[1, 1] <- 1
> I[2, 2] <- 1
> I[3, 3] <- 1
> I
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

In words, the above code asks R to build a 3 x 3 matrix with zeros everywhere (which is why we specified a zero before identifying the rows and columns) except in positions row 1, column 1 ([1, 1]) where we are asking R to generate values of 1. In row 2, column 2, we are again asking R to generate values of 1, and row 3, column 3, we also request values of 1.

If we desired a larger identity matrix, suppose 5 x 5, we would code:

```
> I <- matrix(0, nrow = 5, ncol = 5)
> I[1, 1] <- 1
> I[2, 2] <- 1
> I[3, 3] <- 1
> I[4, 4] <- 1
> I[5, 5] <- 1
> I

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

Notice that if we altered the above general matrix code, we could easily generate a matrix with values, say, 5 everywhere, except for say, values of 2 along the main diagonal. We would accomplish this as follows:

```
> I <- matrix(5, nrow = 3, ncol = 3)
> I[1, 1] <- 2
> I[2, 2] <- 2
> I[3, 3] <- 2
> I
```

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

We can also verify the main diagonal of an identity matrix (or any other matrix) by the function diag. For example, for the matrix **I**, we request the diagonal:

```
> diag(I)
[1] 1 1 1 1 1
```

which also serves to confirm the matrix to be a 5 x 5 matrix.

More generally, we can use the matrix function to generate any matrix with specific values in each position. For instance, if we wanted to generate the matrix $\mathbf{A} = \begin{bmatrix} 5 & 9 \\ 3 & 4 \end{bmatrix}$, we would enter the following code:

```
> A <- matrix(nrow = 2, ncol = 2)
> A[1, 1] <- 5
> A[1, 2] <- 9
> A[2, 1] <- 3
> A[2, 2] <- 4
> A
     [,1] [,2]
[1,]    5    9
[2,]    3    4
```

Though we could generate matrix **A** as we did above, it is usually easier to generate such matrices by simply binding column or row vectors, as we did earlier, especially for large matrices.

Operations on Matrices

Matrix addition and subtraction is defined only for matrices of the same dimension. For example, though we can add matrices **A** and **B**,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{32} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

such that

$$\mathbf{A}+\mathbf{B}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}+\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{32} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{32}+b_{32} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{bmatrix}$$

Addition of matrices **A** and **C**

$$\mathbf{A}+\mathbf{C}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}+\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

on the other hand, is undefined since **A** and **C** are not of the same dimension.

As before, let matrix $\mathbf{A}=\begin{bmatrix} 5 & 9 \\ 3 & 4 \end{bmatrix}$ and matrix $\mathbf{B}=\begin{bmatrix} 1 & 3 \\ 7 & 2 \end{bmatrix}$.

In R, the matrix addition **A** + **B** is therefore:

```
> A + B
     a1 a2
[1,]  6 12
[2,] 10  6
```

The product **AC** is defined only for matrices for which the *number of columns in **A** is equal to the number of rows in **C***. For example, let matrices **A** and **C** be

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \mathbf{C}=\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

Notice that the number of columns in **A** is equal to the number of rows in **C**. Multiplying each element in respective rows of **A** against each element in respective columns of **C**, the multiplication of matrices **A** and **C** proceeds:

$$\mathbf{AC}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$
$$=\begin{bmatrix} a_{11}c_{11}+a_{12}c_{21}+a_{13}c_{31} & a_{11}c_{12}+a_{12}c_{22}+a_{13}c_{32} \\ a_{21}c_{11}+a_{22}c_{21}+a_{32}c_{31} & a_{21}c_{12}+a_{22}c_{22}+a_{32}c_{32} \\ a_{31}c_{11}+a_{32}c_{21}+a_{33}c_{31} & a_{31}c_{12}+a_{32}c_{22}+a_{33}c_{32} \end{bmatrix}$$

so that the product **AC** has the *m* rows in **A** and the *n* columns in **C**.

We can multiply matrices $\mathbf{A} = \begin{bmatrix} 5 & 9 \\ 3 & 4 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 1 & 3 \\ 7 & 2 \end{bmatrix}$ in R as follows:

```
> A%*%B
     b1 b2
[1,] 68 33
[2,] 31 17
```

Recall that in general, *matrix multiplication is not commutative*, meaning that the product **AB** is usually different from the product **BA**. For example, for matrices **A** and **B**, the product **BA** is:

```
> B%*%A
     a1 a2
[1,] 14 21
[2,] 41 71
```

which we note is not equal to the product **AB** just previously computed.

Vectors

A vector is a matrix that consists of a single row or a single column. For instance, vector **C** is a 3 x 1 column vector

$$\mathbf{C} = \begin{bmatrix} c_{11} \\ c_{21} \\ c_{31} \end{bmatrix}$$

whereas vector **C'** is an 1 x 3 column vector

$$\mathbf{C}' = \begin{bmatrix} c_{11} & c_{12} & c_{13} \end{bmatrix}$$

In line with our earlier definition of matrix transposition, $\mathbf{C}'$ is the transpose of $\mathbf{C}$.

Let vector C equal

$$\mathbf{C} = \begin{bmatrix} 5 \\ 2 \\ 9 \end{bmatrix}$$

To generate this vector in R, we compute:

```
> C <- c(5, 2, 9)
> C
[1] 5 2 9
```

The *inner product* is the product produced when two vectors are multiplied by one another. We write the first vector as its transpose, to allow multiplication by the second vector (i.e., so that the matrices are conformable). For instance, consider the vector **D**:

$$\mathbf{D} = \begin{bmatrix} 6 \\ 2 \\ 5 \end{bmatrix}$$

The inner product of vectors **C** and **D**:

$$\mathbf{C'D} = \begin{bmatrix} 5 & 2 & 9 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \\ 5 \end{bmatrix} = 30 + 4 + 45 = 79$$

To demonstrate in R, we generate the vector **D**:

```
> D <- c(6, 2, 5)
> D
[1] 6 2 5
```

The inner product of vectors C and D is therefore

```
> C%*%D
     [,1]
[1,]   79
```

which we note is equal to that computed manually above. Notice that in computing the product **C*D** in R, we did not have to specify vector **C** as its transpose. R makes the adjustement itself and multiplies the vectors. Had R not made this adjustment, we would be multiplying two column vectors, which would not be conformable for multiplication. We could have also specified the transpose directly and obtained the same inner product:

```
> t(C)%*%D
     [,1]
[1,]   79
```

Multiplication of **C** by a scalar *a* proceeds by multiplying each element of **C** by that scalar:

$$a\mathbf{C} = a\begin{bmatrix} c_{11} \\ c_{21} \\ c_{31} \end{bmatrix}$$
$$= \begin{bmatrix} a(c_{11}) \\ a(c_{21}) \\ a(c_{31}) \end{bmatrix}$$

For matrix $\mathbf{C} = \begin{bmatrix} 5 \\ 2 \\ 9 \end{bmatrix}$, multiply by scalar a = 10. The result is

```
> 10*C
[1] 50 20 90
```

Zero Matrix

In a zero matrix (or “null” matrix), every element is equal to 0. We can easily produce a zero matrix in R. For instance, suppose we wanted a 3 x 3 zero matrix:

```
> M <- matrix(0, nrow = 3, ncol = 3)
> M
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

It is easy to see that the zero matrix is the matrix equivalent to zero in scalar algebra, such that pre- or post-multiplication of a matrix by the zero matrix always generates another zero matrix.

Dimension of a Matrix

The dimension of a matrix is defined by the number of rows and columns in the matrix. For instance, consider the zero matrix **M** computed just computed. Matrix **M** has 3 rows and 3 columns. We can easily verify the dimension of a matrix by requesting dim:

```
> dim(M)
[1] 3 3
```

The first “3” indicates the number of rows, while the second “3” indicates the number of columns.

Powers of a Matrix

Taking powers of a matrix is straightforward, and has an analogous interpretation to exponentiation in scalar algebra. To expoentiate a matrix, we exponentiate each element of that matrix by the specified power. For example, raising a matrix to the power of 2 requests R to multiply each element of the matrix by the scalar 2. As an example, consider the previously defined matrix $\mathbf{A} = \begin{bmatrix} 5 & 9 \\ 3 & 4 \end{bmatrix}$ raised to the power of 2:

```
> A^2
     a1 a2
[1,] 25 81
[2,]  9 16
```

Inverse of a Matrix

In scalar algebra, the inverse of an ordinary scalar a is defined as a^{-1} so that

$$a \cdot a^{-1} = \frac{a}{1} \cdot \frac{1}{a} = \frac{a}{a} = 1$$

assuming $a \neq 0$.

In matrix algebra, the equivalent statement of the above scalar relation for matrices **A** and **B** is:

AB = BA = I

We state the above matrix relation more formally: Let **A** be a square matrix. If a matrix **B** can be found such that **AB** = **BA** = **I**, then the matrix **A** is regarded as an invertible matrix, and matrix **B** is regarded as the inverse of matrix **A**.

Generating matrix inverses in R is very easy. For instance, consider once more cormatrix:

```
> cormatrix
         a     b     c     d     e     f     g     h
[1,] 1.000 0.343 0.505 0.308 0.693 0.208 0.400 0.455
[2,] 0.343 1.000 0.203 0.400 0.187 0.108 0.386 0.385
[3,] 0.505 0.203 1.000 0.398 0.303 0.277 0.286 0.167
[4,] 0.308 0.400 0.398 1.000 0.205 0.487 0.385 0.465
[5,] 0.693 0.187 0.303 0.205 1.000 0.200 0.311 0.485
[6,] 0.208 0.108 0.277 0.487 0.200 1.000 0.432 0.310
[7,] 0.400 0.386 0.286 0.385 0.311 0.432 1.000 0.365
[8,] 0.455 0.385 0.167 0.465 0.485 0.310 0.365 1.000
```

We can request R to compute the inverse of cormatrix by solve(cormatrix). We name the new matrix by the name D:

```
> D <- solve(cormatrix)

         [,1]        [,2]        [,3]        [,4]        [,5]        [,6]        [,7]        [,8]
a  2.62391779 -0.32529402 -0.79748547  0.07143344 -1.34883744  0.14277980 -0.27074036 -0.25993622
b -0.32529402  1.45211520  0.03592592 -0.40955548  0.22539180  0.31041745 -0.38009280 -0.29342246
c -0.79748547  0.03592592  1.56495953 -0.48016535  0.04087805 -0.13719304 -0.03786275  0.34747695
d  0.07143344 -0.40955548 -0.48016535  1.85491453  0.19261836 -0.58606745 -0.05349807 -0.54938320
e -1.34883744  0.22539180  0.04087805  0.19261836  2.14973569 -0.07659356 -0.06236400 -0.56556392
f  0.14277980  0.31041745 -0.13719304 -0.58606745 -0.07659356  1.53618210 -0.49852528 -0.14614977
g -0.27074036 -0.38009280 -0.03786275 -0.05349807 -0.06236400 -0.49852528  1.55055676 -0.08044157
h -0.25993622 -0.29342246  0.34747695 -0.54938320 -0.56556392 -0.14614977 -0.08044157  1.77763927
```

To verify that the above matrix is indeed the inverse of cormatrix, we can easily demonstrate that their product is equal to the identity matrix (we leave this as an exercise), hence establishing the relation **AB** = **BA** = **I**.

Using R, we now demonstrate some of the more common properties of matrix inverses:

1. $(\mathbf{A}^{-1})' = (\mathbf{A}')^{-1}$

The above reads that the transpose of the inverse of a matrix is equal to the inverse of its transpose. We demonstrate with matrix $\mathbf{A} = \begin{bmatrix} 5 & 9 \\ 3 & 4 \end{bmatrix}$.

We first request the left-hand side of the property, that of the transpose of the inverse:

```
> t(solve(A))
             a1         a2
[1,] -0.5714286  0.4285714
[2,]  1.2857143 -0.7142857
```

We then compute the inverse of the transpose, and note that it is equal to the transpose of the inverse computed above.

```
> solve(t(A))
             a1         a2
[1,] -0.5714286  0.4285714
[2,]  1.2857143 -0.7142857
```

2. $(\mathbf{AB})^{-1} = (\mathbf{B})^{-1} \cdot (\mathbf{A})^{-1}$

The above reads that the inverse of the product of matrices is equal to the product of inverses in reverse order. Recall matrix $\mathbf{B} = \begin{bmatrix} 1 & 3 \\ 7 & 2 \end{bmatrix}$.

We demonstrate the above property in R by first requesting the inverse of the product (left-hand side):

```
> solve(A%*%B)
         [,1]       [,2]
b1  0.1278195 -0.2481203
b2 -0.2330827  0.5112782
```

We then request the right-hand side, the product of inverses in reverse order, and note that the resulting matrix is identical to that produced above.

```
> solve(B) %*% solve(A)
         [,1]       [,2]
b1  0.1278195 -0.2481203
b2 -0.2330827  0.5112782
```

3. $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$

The above reads that the inverse of the inverse of a matrix is equal to the original matrix.

This is easily demonstrated in R. The left-hand side is computed as:

```
> solve(solve (A))
     a1 a2
[1,]  5  9
[2,]  3  4
```

which is easily shown to be equal to the original matrix **A**:

```
> A
     a1 a2
[1,]  5  9
[2,]  3  4
```

Moore-Penrose Generalized Inverse

As given in Searle (1982, p. 212), the Moore-Penrose inverse **M** of matrix **A** is defined such that the following conditions hold:

$$\mathbf{AMA = A}$$

M is said to be the Moore-Penrose inverse of **A**. To demonstrate the M-P inverse, we require the package MASS to be loaded into R. We now compute the M-P inverse of matrix **A** with the command `ginv(A)`:

```
> ginv(A)
           [,1]       [,2]
[1,] -0.5714286  1.2857143
[2,]  0.4285714 -0.7142857
```

We easily demonstrate that **AMA = A** holds by pre-multiplying ginv(A) by **A** and post-multiplying ginv by **A**:

```
> A%*%ginv(A)%*%A

     a1 a2
[1,]  5  9
[2,]  3  4
```

We note that **AMA** reproduced the original matrix **A**, and thus the matrix computed by ginv(A) is indeed the M-P inverse of matrix **A**.

Symmetric Matrices and Inverses

An *n* x *n* matrix **A** is symmetric if it is equal to its transpose, that is, $\mathbf{A}' = \mathbf{A}$. For instance, consider the matrix **S**:

$$\mathbf{S} = \begin{bmatrix} 6 & 4 & 8 \\ 4 & 9 & 3 \\ 8 & 3 & 7 \end{bmatrix}$$

The matrix is symmetric since the upper triangular is a mirror reflection of the lower triangular such that $\mathbf{S}' = \mathbf{S}$, as we can easily demonstrate in R. First, we construct the matrix **S**:

```
> s1 <- c(6, 4, 8)
> s2 <- c(4, 9, 3)
> s3 <- c(8, 3, 7)
> S <- cbind (s1, s2, s3)
> S
     s1 s2 s3
[1,]  6  4  8
[2,]  4  9  3
[3,]  8  3  7
```

When taking the transpose, we find

```
> t(S)
   [,1] [,2] [,3]
s1    6    4    8
s2    4    9    3
s3    8    3    7
```

which we note is equivalent to the original matrix **S**, and hence matrix **A** is symmetric.

If a matrix is symmetric and invertible, then its inverse is also symmetric. For instance, consider once more the symmetric matrix **S**. Computing the inverse of $\mathbf{S}$, we obtain

```
> solve(S)
          [,1]        [,2]        [,3]
s1 -0.31395349  0.02325581  0.34883721
s2  0.02325581  0.12790698 -0.08139535
s3  0.34883721 -0.08139535 -0.22093023
```

We note that the matrix $\mathbf{S}^{-1}$ is symmetrical, as was the original symmetric matrix $\mathbf{S}$.

Determinants

The determinant of a matrix is a distinct number associated with a matrix. Determinants are defined only for square matrices. Though the computation of determinants for matrices of higher dimensions can quickly become unwieldly and laborious, it is a simple matter to demonstrate the computation for simple lower-dimension matrices. For instance, for a 2 x 2 matrix A, the determinant is computed as

$$|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} + (-1)a_{12}a_{21} = a_{11}a_{22} - a_{12}a_{21}.$$

For a 3 x 3 matrix, the computation of the determinant proceeds as follows:

$$\begin{aligned} |\mathbf{A}| &= \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \\ &= a_{11}(+1)\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{12}(-1)\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}(+1)\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ &= a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31}. \end{aligned}$$

We can obtain the determinant of a matrix quite easily in R. For example, the determinant of cormatrix is computed as:

```
> det(cormatrix)
[1] 0.06620581
```

Some properties and results of determinants include the following:

1. $|\mathbf{A'}| = |\mathbf{A}|$

The above reads that the determinant of the transpose of a matrix is equal to the determinant of the matrix. To demonstrate in R for cormatrix, we first compute the determinant of the transpose:

```
> det(t(cormatrix))
[1] 0.06620581
```

This is equal to the original determinant of cormatrix, since

```
> det(cormatrix)
[1] 0.06620581
```

2. $|\mathbf{AB}| = |\mathbf{A}||\mathbf{B}|$

The above reads that the determinant of a product of matrices (if defined) is equal to the product of determinants:

Let matrix **A** again be defined as

$$\mathbf{A} = \begin{bmatrix} 5 & 9 \\ 3 & 4 \end{bmatrix}$$

and let matrix **B** again be defined as

$$\mathbf{B} = \begin{bmatrix} 1 & 3 \\ 7 & 2 \end{bmatrix}$$

Now, we compute the determinant of **AB**:

```
> det(A%*%B)
[1] 133
```

and equate this to the product of determinants:

```
> det(A) %*% det(B)

     [,1]
[1,]  133
```

Hence, we've demonstrated that $|\mathbf{AB}| = |\mathbf{A}||\mathbf{B}|$.

3. $|k\mathbf{A}| = k^n |\mathbf{A}|$

The above reads that if *k* is a constant, then for a square matrix with *n* rows, the determinant of the product of the constant and the matrix is equal to the the product of the constant raised to the power of *n* and the determinant of the matrix.

This is easily demonstrated in R. We let *k* = 3. Then

```
> det(3*A)
[1] -63
```

is equivalent to

```
> 3^2%*%det(A)
     [,1]
[1,]  -63
```

4. $|\mathbf{AB}| = |\mathbf{A}||\mathbf{B}|$

The above reads that the determinant of a product is equal to the product of determinants. To demonstrate, in R, we compute:

```
> det(A%*%B)
[1] 133
```

which is equal to

```
> det(A)%*% det(B)
     [,1]
[1,]  133
```

Linear Independence and the Determinant Test for Matrix Invertibility

A square matrix that has one or more rows (or columns) as a scalar multiple of another row (or column) will have a determinant equal to 0, and will not be invertible. For instance, consider the following matrix **D**:

$$\mathbf{D} = \begin{bmatrix} 2 & 5 & 7 \\ 3 & 1 & 8 \\ 4 & 10 & 14 \end{bmatrix}$$

We note that row 3 is a scalar multiple of row 1 since row 3 is equal to row 1 multiplied by a factor of 2. Hence, there is *linear dependence* among these two rows, and so the determinant of the matrix will equal 0, and the matrix will not be invertible.

More formally and precisely, we can define *linear independence* by reference to vectors in the following way: If $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ consists of a set of vectors, then it is possible to produce the following linear combination of vectors:

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n = \mathbf{0}$$

where a_1 through a_n are scalars. If the only solution possible for the above linear combination is to let $a_1, a_2 ... a_n$ all equal 0, then the set of vectors $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ is said to be *linearly independent*. If there are other choices for $a_1, a_2 ... a_n$ such that they are not all equal to 0 and the linear combination $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n = \mathbf{0}$ still holds, then the set **V** is said to be a *linearly dependent* set of vectors.

We demonstrate linear dependence using matrix **D**. First we construct the matrix:

```
> c1 <- c(2, 3, 4)
> c2 <- c(5, 1, 10)
> c3 <- c(7, 8, 14)
> D <- cbind(c1, c2, c3)
> D
     c1 c2 c3
[1,]  2  5  7
[2,]  3  1  8
[3,]  4 10 14
```

We now demonstrate the linear dependence within the set of vectors and show that, within rounding error, the matrix **D** has a determinant equal to 0:

```
> det(D)
[1] 0
```

Given that **D** has a determinant equal to 0, we should expect to not be able to compute its inverse. When we attempt to solve in R, we obtain:

```
> solve(D)
Error in solve.default(D) :
  Lapack routine dgesv: system is exactly singular
```

We note that R reports that "system is exactly singular." A matrix is regarded as *singular* when its determinant is equal to 0, and non-singular when its determinant is not equal to 0. Because matrix D had rows that were linearly dependent, the determinant of the matrix was computed to be 0, and the matrix is regarded as singular.

Orthogonality

A square matrix **A** is said to be an orthogonal matrix if (and only if) the following holds:

$$\mathbf{AA'} = \mathbf{A'A} = \mathbf{I}$$

which in words translates to: *a matrix post-multiplied by its transpose is equal to the matrix pre-multiplied by its transpose, and both of these products are equal to the identity matrix*. Under this condition, matrix **A** is defined as *orthogonal*.

Substantive areas of multivariate analysis in which orthogonal matrices play a very important role is in principal components analysis and factor analysis, specifically, the orthogonal rotation of estimated component or factor loadings. Solutions to loadings are multiplied by the following transformation matrix **T**:

$$\mathbf{T} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Rotation by matrix **T** constitutes an orthogonal rotation since $\mathbf{T'T} = \mathbf{I}$, as demonstrated below:

$$\begin{aligned} \mathbf{T'T} &= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} (\cos\theta)(\cos\theta) + (\sin\theta)(\sin\theta) & (\cos\theta)(-\sin\theta) + (\sin\theta)(\cos\theta) \\ (-\sin\theta)(\cos\theta) + (\cos\theta)(\sin\theta) & (-\sin\theta)(-\sin\theta) + (\cos\theta)(\cos\theta) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

The property $\mathbf{T'T} = \mathbf{I}$ holds regardless of the choice of angle θ.

We now demonstrate $\mathbf{T'T} = \mathbf{I}$ in R. We first build the matrices $\mathbf{T'}$ and $\mathbf{T}$, and for the sake of example, we assume an angle of 20 degrees. For $\mathbf{T'}$, we define the requisite vectors as follows:

```
> t1 <- c(cos(20), -sin(20))
> t1
[1]  0.4080821 -0.9129453

> t2 <- c(sin(20), cos(20))
> t2
[1] 0.9129453 0.4080821

> t_transpose <- cbind(t1, t2)
> t_transpose
             t1        t2
[1,]  0.4080821 0.9129453
[2,] -0.9129453 0.4080821
```

The above matrix represents the matrix for $\mathbf{T'}$ just considered:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos(20) & \sin(20) \\ -\sin(20) & \cos(20) \end{bmatrix}$$

For $\mathbf{T}$, we have:

```
> t3 <- c(cos(20), sin(20))
> t4 <- c(-sin(20), cos(20))
> T <- cbind(t3, t4)
> T
            t3         t4
[1,] 0.4080821 -0.9129453
[2,] 0.9129453  0.4080821
```

The above matrix represents the matrix for $\mathbf{T}$ just considered:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos(20) & -\sin(20) \\ \sin(20) & \cos(20) \end{bmatrix}$$

It is then a simple matter to demonstrate that $\mathbf{T'T} = \mathbf{I}$:

```
> T_transpose %*% T
     t3 t4
[1,]  1  0
[2,]  0  1
```

A few other properties of orthogonal matrices (which we neither prove nor demonstrate) include:

1. If **A** is an orthogonal matrix, then its inverse, $\mathbf{A}^{-1}$ is also an orthogonal matrix.

2. If **A** is an orthogonal matrix, and **B** is an orthogonal matrix, then **AB** is an orthogonal matrix.

3. The determinant of an orthogonal matrix is equal to either 1 or -1 (e.g., det(T) = 1).

Eigenvalues & Eigenvectors

The importance and relevance of the concepts of eigenvalues and eigenvectors cannot be overstated as they relate to multivariate analysis. Indeed, most multivariate procedures, at their core, are in part extractions of eigenvalues and their associated eigenvectors from covariance or correlation matrices. Though there are numerous ways of applying eigenvalue and eigenvector decomposition, or EVD to substantive problems in a variety of sciences, the mathematical "reality" of these concepts is entirely abstract (though a look at its history might possibly reveal a more practical motive), and it behooves us to review some of the theory of EVD here, without going into too much and unecessary depth.

Let $\mathbf{A}$ be a $n \cdot n$ square matrix. The eigenvalue problem is to find numbers λ_i and a vector of dimension *n* such that the following equation holds:

$$\mathbf{Av} = \boldsymbol{\lambda}\mathbf{v}$$

Each number λ_i extracted is an eigenvalue of the matrix **A** with corresponding vector in **v**. The equation $\mathbf{Av} = \boldsymbol{\lambda}\mathbf{v}$ can be expressed as

$$(\mathbf{A} - \boldsymbol{\lambda}\mathbf{I})\mathbf{v} = \mathbf{0}$$

We seek to determine values for $\boldsymbol{\lambda}$ for which the system of equations considered has a solution. In linear algebra, it can be shown that $(\mathbf{A} - \boldsymbol{\lambda}\mathbf{I})\mathbf{v} = \mathbf{0}$ has a solution if (and only if) the determinant of $\mathbf{A} - \boldsymbol{\lambda}\mathbf{I}$ equals 0, that is,

$$|\mathbf{A} - \boldsymbol{\lambda}\mathbf{I}| = 0$$

As a simple example of EVD, consider the following matrix **A**:

$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 0 & 6 \end{bmatrix}$$

Then the vector $\mathbf{v} = \begin{bmatrix} 0.4472136 \\ 0.8944272 \end{bmatrix}$ is an eigenvector of **A** that corresponds to the eigenvalue 6, because

$$\begin{bmatrix} 2 & 2 \\ 0 & 6 \end{bmatrix} \begin{bmatrix} 0.4472136 \\ 0.8944272 \end{bmatrix} = 6 \begin{bmatrix} 0.4472136 \\ 0.8944272 \end{bmatrix}$$

as we can easily verify in R. We first construct the matrix **A**:

```
> a1 <- c(2, 0)
> a2 <- c(2, 6)
> A <- cbind(a1, a2)
> A
     a1 a2
[1,]  2  2
[2,]  0  6
```

along with vector **v**:

```
> v <- c(0.4472136, 0.8944272)
> v
[1] 0.4472136 0.8944272
```

The product of A*v is equal to:

```
> A%*%v
         [,1]
[1,] 2.683282
[2,] 5.366563
```

which is equal to the product of

```
> 6*v
[1] 2.683282 5.366563
```

and hence we've demonstrated that $\mathbf{v} = \begin{bmatrix} 0.4472136 \\ 0.8944272 \end{bmatrix}$ is an eigenvector of $\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 0 & 6 \end{bmatrix}$ corresponding to an eigenvalue of 6.

Computing eigenvalues and eigenvectors for matrices of small dimension provides little computational difficulty. However, the majority of times in multivariate analysis, we are required to compute EVD for matrices of much larger dimension and computed manually "by hand," the computations quickly become unwieldy and software is needed.

For instance, consider once more the correlation matrix cormatrix. We can request the eigenvalues and eigenvectors of this matrix quite easily in R, by using the eigen function:

```
> eigen(cormatrix)

$values
[1] 3.4470520 1.1572358 0.9436513 0.8189869 0.6580753 0.3898612 0.3360577
[8] 0.2490798

$vectors
           [,1]        [,2]         [,3]        [,4]         [,5]       [,6]
[1,] -0.4125682 -0.45771854 -0.098873465  0.08795002  0.066755981 -0.1352881
[2,] -0.3034726  0.11443307  0.637320040  0.47138991  0.088175982 -0.4697628
[3,] -0.3180940 -0.06869755 -0.546391282  0.58136533 -0.121757436  0.1921707
[4,] -0.3730602  0.43006317 -0.001725853  0.11149001 -0.471416291  0.1757057
[5,] -0.3572744 -0.54341592 -0.067983885 -0.31379342  0.005351703 -0.2192800
[6,] -0.3008318  0.49347718 -0.380799221 -0.40057953  0.065440460 -0.5409419
[7,] -0.3664721  0.21605328  0.060830978 -0.06123129  0.775010839  0.4437028
[8,] -0.3806410 -0.04717545  0.363552650 -0.39618322 -0.381782069  0.3945174
           [,7]        [,8]
[1,]  0.0911831  0.75610613
[2,] -0.1315217 -0.14378501
[3,] -0.3684690 -0.26465820
[4,]  0.6393505  0.03954700
[5,]  0.3112839 -0.57354688
[6,] -0.2453502  0.05840491
[7,]  0.1046301 -0.05574971
[8,] -0.5116712  0.02337248
```

The first eigenvalue extracted is equal to 3.4470520 with associated eigenvector

$$\begin{bmatrix} -0.4125682 \\ -0.3034726 \\ -0.3180940 \\ -0.3730602 \\ -0.3572744 \\ -0.3008318 \\ -0.3664721 \\ -0.3806410 \end{bmatrix}$$

In line with our earlier discussion of EVD, then it should be true that the product of cormatrix*eigenvector is equal to the eigenvalue*eigenvector. We verify this in R by first producing the eigenvector, then generating the two products:

```
 > cormatrix%*%eigenvec
            [,1]
c1 -1.422144
c2 -1.046086
c3 -1.096486
c4 -1.285958
c5 -1.231543
c6 -1.036983
c7 -1.263249
c8 -1.312089

> 3.4470520*eigenvec
[1] -1.422144 -1.046086 -1.096487 -1.285958 -1.231543 -1.036983 -1.263248
[8] -1.312089
```

We note that both vectors, that of cormatrix*eigenvec and eigenvalue*eigenvec are equal, confirming that the stated eigenvector is indeed an eigenvector of the eigenvalue 3.4470520.

A few properties of eigenvalues and eigenvectors:

1. $\prod \lambda_i = |\mathbf{A}|$

The above reads that the product eigenvalues for matrix **A** is equal to the determinant of matrix **A**. We can verify this for cormatrix. We first produce the vector of extracted eigenvalues then request the product (prod) of these eigenvalues :

```
> eigenvalues <- c(3.4470520, 1.1572358, 0.9436513, 0.8189869, 0.6580753,
0.3898612, 0.3360577, 0.2490798)
> prod(eigenvalues)
[1] 0.06620582
```

We note that the value of 0.06620582 matches that value of the determinant of cormatrix computed earlier [det(cormatrix) = 0.06620581]

Rank

A central problem of linear algebra, and hence one that also appears in multivariate analysis, is to obtain a solution for $\mathbf{x}$ in the equation $\mathbf{Ax} = \mathbf{b}$. The solution for $\mathbf{x}$ exists only if we can solve for $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. That is, the solution for $\mathbf{x}$ exists only if the inverse of $\mathbf{A}$ exists. *The rank of a matrix is defined as the number of linearly independent rows and columns of a matrix*. If the number of columns or rows in a matrix is equal to the number of linearly independent columns or rows, the matrix is said to be of full rank, and $\mathbf{A}^{-1}$ exists.

Checking whether rows or columns of a matrix are linearly independent or dependent can be a lengthy process. Fortunately, and as already discussed, if linear independence does hold, then the determinant of a matrix will not equal 0. Hence, as a quick check to ensure a matrix is of full rank, one simply needs to compute the determinant of the matrix in question and ensure it is not equal to 0.

Recall matrix $\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 0 & 6 \end{bmatrix}$. The determinant of this matrix is equal to 12. Hence, we know the matrix is of full rank because the determinant is not equal to 0. In other words, the rows and columns in **A** are linearly independent.

In R, we can obtain the rank of a matrix by the function qr. Because the rows and columns of matrix **A** are linearly independent, we should expect the rank to be equal to 2. We confirm this in R (the function qr will produce more output than we need, we only show the output for matrix rank below:

```
> qr(A)

$rank
[1] 2
```

As we had properly concluded, the rank of matrix **A** is equal to 2.

Least-Squares Solution in R Using Matrix Operations

One of the joys of performing matrix algebra in R is that one can reproduce virtually any analysis by the simple manipulation and construction of elementary matrices. For instance, let us consider the problem of least-squares, and instead of invoking a "routine" in commercial software, we demonstrate how the solution can be obtained by simple matrix construction and operations in R. For readers unfamiliar with the matrices of linear models, we review the necessary theory in chapter 2 on simple linear regression. For now, we assume the reader is already familiar with the matrix algebra of linear models, and provide a quick overview of how these computations can be performed using simple matrix operations by referring to an empirical example used in chapter 2.

In chapter 2, we consider the following problem in simple linear regression:

Subject	Quantitative	Verbal
1	5	2
2	2	1
3	6	3
4	9	7
5	8	9
6	7	8
7	9	8
8	10	10
9	10	9

We are wanting to predict one's quantitative score based on one's verbal score. That is, we want to estimate a model that uses one's score on verbal to predict their quantitative score. We generate the following vectors for Q (quantitative) and V(verbal):

```
> Q <- c(5, 2, 6, 9, 8, 7, 9, 10, 10)
> V <- c(2, 1, 3, 7, 9, 8, 8, 10, 9)
```

We can represent the linear model of $\mathbf{Y} = \mathbf{X}\beta + \boldsymbol{\varepsilon}$ as:

$$\mathbf{Y} = \begin{bmatrix} Y_{i=1} \\ Y_{i=2} \\ Y_{i=3} \\ . \\ . \\ . \\ Y_{i=n} \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 1 & X_{i=1} \\ 1 & X_{i=2} \\ 1 & X_{i=3} \\ . & . \\ . & . \\ . & . \\ 1 & X_{i=n} \end{bmatrix} \qquad \beta = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \qquad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ . \\ . \\ . \\ \varepsilon_n \end{bmatrix}$$

The normal equations for least-squares can be found as follows:

$$\mathbf{X'X\,b} = \mathbf{X'Y}$$

To get the matrix $\mathbf{X}$, we generate the two relevant vectors, then bind them together. We name the first vector by the name of I (for "intercept") then bind both I and V into X:

```
> I <- c(1, 1, 1, 1, 1, 1, 1, 1, 1)
> I
```

```
[1] 1 1 1 1 1 1 1 1 1
> X <- cbind (I, V)
> X
      I  V
 [1,] 1  2
 [2,] 1  1
 [3,] 1  3
 [4,] 1  7
 [5,] 1  9
 [6,] 1  8
 [7,] 1  8
 [8,] 1 10
 [9,] 1  9
```

We now can solve for $\mathbf{X'X}$ quite easily:

```
> XTX <- t(X)%*%X
> XTX
   I   V
I  9  57
V 57 453
```

Likewise, we can solve for $\mathbf{X'Y}$

```
> XTY <- t(X)%*%Q
> XTY
  [,1]
I   66
V  483
```

Making the appropriate substitutions, our equation thus far reads:

$$\begin{bmatrix} 9 & 57 \\ 57 & 453 \end{bmatrix} \mathbf{b} = \begin{bmatrix} 66 \\ 483 \end{bmatrix}$$

We proceed to then solve for **b**:

$$\mathbf{b} = (\mathbf{X'X})^{-1}\mathbf{X'Y}$$

$$\mathbf{b} = \begin{bmatrix} 9 & 57 \\ 57 & 453 \end{bmatrix}^{-1} \begin{bmatrix} 66 \\ 483 \end{bmatrix}$$

The inverse of $\mathbf{X'X}$ is

```
> XTXI <- solve(XTX)

            I            V
```

```
I  0.54710145 -0.06884058
V -0.06884058  0.01086957
```

The solution for **b** is thus

```
> XTXI %*% XTY
       [,1]
I 2.8586957
V 0.7065217
```

Hence, the estimate of the intercept term is 2.859 while the estimate of the slope parameter is 0.707.

As a check on our computations, we request the corresponding linear model in R. We use the function lm to request the linear regression:

```
> lm(Q~V)

Call:
lm(formula = Q ~ V)

Coefficients:
(Intercept)            V
     2.8587       0.7065
```

We note that the estimated intercept of 2.8587 and the slope of .7065 match up with those we computed "manually" through matrices using R.

END.