Matrix algebra

Matrix algebra is very prevalently used in Statistics because it provides representations of models and computations in a much simpler manner than without its use. The purpose of this section is to provide the basics of commonly used matrix algebra operations in R. The program used in this section is basic_matrix_algebra.R.

Basics

A matrix is simply a rectangular arrangement of elements in rows and columns. For example, consider the matrix $\mathbf{A}$ as

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Letters that represent matrices are always bolded or written with a line underneath as $\underline{A}$ when bolding is not possible, like on a chalkboard. The symbolic elements of a matrix are written with a subscript for the row and column numbers (row, column). The dimension of the matrix is its number of rows and columns. For the above matrix, the dimension is 2×3.

Consider the following two matrices:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} -1 & 10 & -1 \\ 5 & 5 & 8 \end{bmatrix}$$

Below is how we can enter these matrices into R and also perform addition and subtraction operations with them.

```
> A <- matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE)
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

```
> class(A)
[1] "matrix"
> A[1, 1]
[1] 1
> A[1, ]
[1] 1 2 3
> A[, 1]
[1] 1 4
> B <- matrix(data = c(-1, 10, -1, 5, 5, 8), nrow = 2, ncol = 3,
     byrow = TRUE)
> B
     [,1] [,2] [,3]
[1,]   -1   10   -1
[2,]    5    5    8
> A + B
     [,1] [,2] [,3]
[1,]    0   12    2
[2,]    9   10   14
> A - B
     [,1] [,2] [,3]
[1,]    2   -8    4
[2,]   -1    0   -2
```

Comments:

- The matrix function allows one to enter a matrix. Often, it is best to enter the matrix as

  ```
  A <- matrix(data = c(1, 2, 3,
                               4, 5, 6),
              nrow = 2, ncol = 3, byrow = TRUE)
  ```

  so that the columns line up properly. This is how I usually write code in my own programs, but LyX re-formats the code here so it does not appear to occur.

- The `byrow` argument is very important! This instructs R to enter the elements listed in the `data` argument by rows. Thus, row #1 is filled with elements first before row #2. The default for this argument is `byrow = FALSE`. Thus,

```
> matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

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

results in the elements in the `data` argument filling column 1 first.

- Notice that the class of **A** is `matrix`.
- Elements of a matrix can be referenced by their row and column numbers.

A mathematical vector can be represented as a `matrix` class type or a type of its own:

```
> y <- matrix(data = c(1, 2, 3), nrow = 3, ncol = 1, byrow = TRUE)
> y
     [,1]
[1,]    1
[2,]    2
[3,]    3
> class(y)
[1] "matrix"
> x <- c(1, 2, 3)
> x
[1] 1 2 3
> class(x)
[1] "numeric"
> is.vector(x)
[1] TRUE
```

This latter representation can present some confusion when vectors are multiplied with other vectors or matrices because no specific row or column dimensions are given. How to handle these situations will be discussed more shortly.

Question: How can a 4×4 matrix of 1's be created?

Matrix multiplication

Consider the following two matrices:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} 3 & 0 \\ 1 & 2 \\ 0 & 1 \end{bmatrix}$$

The syntax to multiply these matrices together is not simply `*` but `%*%`:

```
> A <- matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE)
> B <- matrix(data = c(3, 0, 1, 2, 0, 1), nrow = 3, ncol = 2, byrow = TRUE)
> A %*% B
     [,1] [,2]
[1,]    5    7
[2,]   17   16
> B %*% A
     [,1] [,2] [,3]
[1,]    3    6    9
[2,]    9   12   15
[3,]    4    5    6
> A * B
Error:  non-conformable arrays
> A * A
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
> 3 * A
     [,1] [,2] [,3]
[1,]    3    6    9
[2,]   12   15   18
```

The $*$ alone denotes element-by-element multiplication. When $*$ is used with a scalar value and a matrix, each element of the matrix is multiplied by the scalar as would be expected.

Consider the mathematical vector of

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Below are some examples when working with what R considers to be a vector:

```
> x <- c(1, 2, 3)
> A %*% x
     [,1]
[1,]   14
[2,]   32
> # Inner product
> x %*% x  # x'x
     [,1]
[1,]   14
> # Outer product
> x %o% x  # xx'
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
```

How does R know to make $\mathbf{x}$ a 3×1 vector for $\mathbf{Ax}$? How does R know to compute the result of x%*%x as $\mathbf{x}'\mathbf{x}$? Here's what R says in its help for %*%:

> Multiplies two matrices, if they are conformable. If one argument is a vector, it will be promoted to either a row or column matrix to make the two arguments conformable. If both are vectors it will return the inner product.

An inner product produces a scalar value. If you wanted $\mathbf{xx}'$, one can use the outer product %o%. Of course, if $\mathbf{x}$ is defined as `x <- matrix(data = c(1, 2, 3), nrow = 3, ncol = 1, byrow = TRUE)`, all calculations occur as one would normally expect with matrix algebra.

Inverse of a matrix

There is no function named "inverse" to find an inverse of a matrix. Rather, the function is named `solve()`:

```
> A <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)
> solve(A)
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
> A %*% solve(A)
     [,1]     [,2]
[1,]    1 1.11e-16
[2,]    0 1.00e+00
> solve(A) %*% A
         [,1] [,2]
[1,] 1.00e+00    0
[2,] 1.11e-16    1
```

The `solve()` function can also be used to "solve" for $\mathbf{x}$ in $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b} \Rightarrow \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. The syntax is `solve(A,b)`.

Question: Why is there no "inverse" function?

More matrix operations

Commonly used operations with matrices include finding the diagonal elements, forming a diagonal matrix, finding determinants, and taking a transpose of a matrix:

```
> # Find diagonal elements of a matrix
> A <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)
> A
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> diag(x = A)
[1] 1 4
> # Create diagonal matrix using same function
> diag(x = c(1, 2, 3, 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
> # Determinant
> det(A)
[1] -2
> # Transpose
> t(A)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Question: How can a 4×4 identity matrix be created?

Eigenvalues

Eigenvalues and eigenvectors are found by using the `eigen()` function.

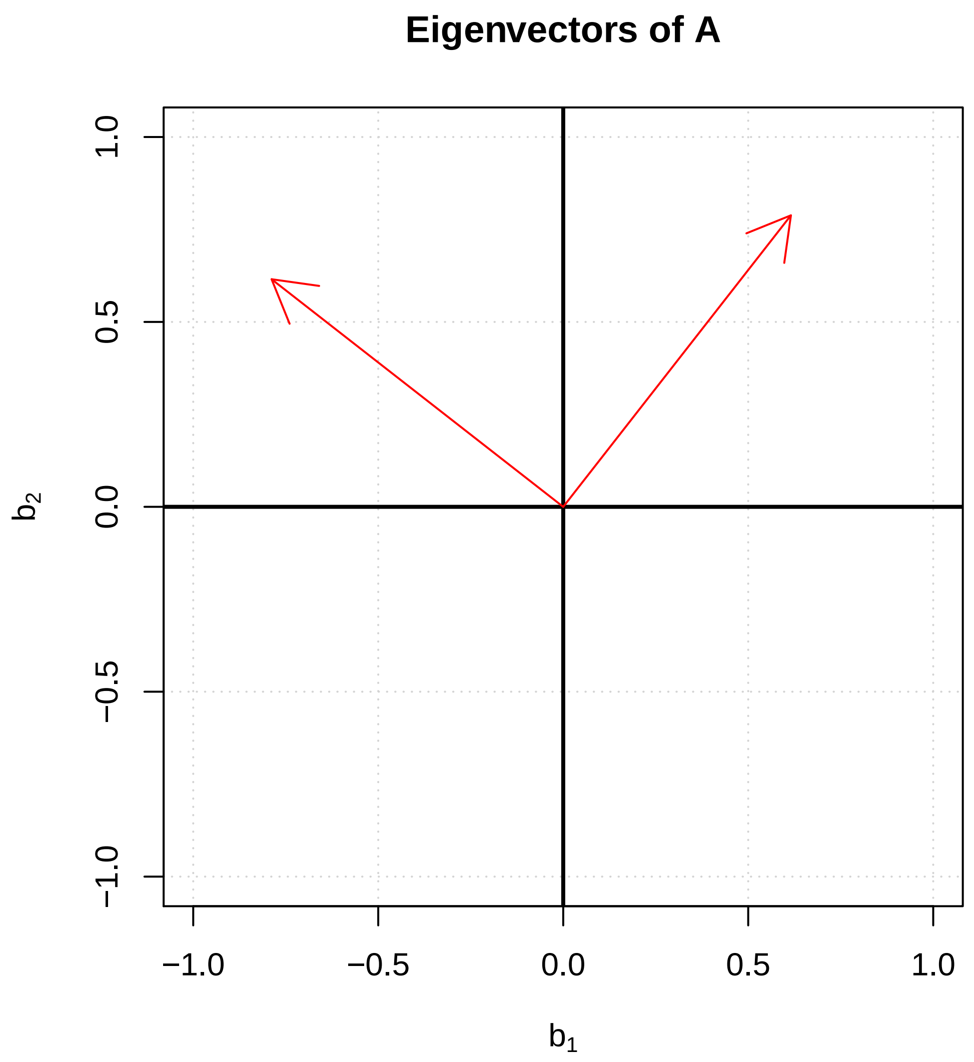
```
> A <- matrix(data = c(1, 0.5, 0.5, 1.25), nrow = 2, ncol = 2,
    byrow = TRUE)
> A
     [,1] [,2]
[1,]  1.0 0.50
[2,]  0.5 1.25
> eigen(A)
$values
[1] 1.6404 0.6096
```

```
$vectors
       [,1]    [,2]
[1,] 0.6154 -0.7882
[2,] 0.7882  0.6154
> save.eig <- eigen(A)
> names(save.eig)
[1] "values"  "vectors"
> # Verify
> A %*% save.eig$vectors[, 1]
      [,1]
[1,] 1.010
[2,] 1.293
> save.eig$values[1] * save.eig$vectors[, 1]
[1] 1.010 1.293
> # Verify
> A %*% save.eig$vectors[, 2]
        [,1]
[1,] -0.4805
[2,]  0.3752
> save.eig$values[2] * save.eig$vectors[, 2]
[1] -0.4805  0.3752
> # Length of a vector
> sqrt(sum(save.eig$vectors[, 1]^2))
[1] 1
> sqrt(sum(save.eig$vectors[, 2]^2))
[1] 1
```

To verify the eigenvectors, I used the relationship that eigenvectors of a matrix $\mathbf{A}$ satisfy $\mathbf{Ab} = \lambda\mathbf{b}$ where $\mathbf{b}$ is an eigenvector. Please remember that eigenvectors are not unique! Eigenvectors in R are scaled to have a length of 1. Still, there is more than one vector that can have a length of 1. I have seen 32-bit and 64-bit R produce different eigenvectors with the exact same code (each element multiplied by -1).

Below is a plot of the eigenvectors to demonstrate that they are orthogonal:

```
> # Square plot (default is 'm' which means maximal region)
> par(pty = "s")
> # Set up some dummy values for plot
> b1 <- c(-1, 1)
> b2 <- c(-1, 1)
> plot(x = b1, y = b2, type = "n", main = "Eigenvectors of A",
     xlab = expression(b[1]), ylab = expression(b[2]), panel.first = grid())
> # Draw line on plot - h specifies a horizontal line
> abline(h = 0, lty = "solid", lwd = 2)
> # v specifies a vertical line
> abline(v = 0, lty = "solid", lwd = 2)
> # Draw eigenvectors
> arrows(x0 = 0, y0 = 0, x1 = save.eig$vectors[1, 1], y1 = save.eig$vectors[2,
     1], col = "red", lty = "solid")
> arrows(x0 = 0, y0 = 0, x1 = save.eig$vectors[1, 2], y1 = save.eig$vectors[2,
     2], col = "red", lty = "solid")
```



Additional comments

- All of the functions examined here are in R's base package. A web page that summarizes many useful functions for matrix algebra is `http://statmethods.net/advstats/matrix.html`.
- There are many other packages available for matrix algebra. For example, the `Matrix` package has functions that are useful for sparse matrices. The `bigalgebra` package has functions that are useful for big matrices. See the Numerical Mathematics task view at `https://cran.r-project.org/web/views/`

`NumericalMathematics.html` for more packages.

- A data frame can be *coerced* into a matrix by using `as.matrix()`.
- There are times when one needs to add a row or column of numbers to a matrix. For example, suppose `A` is a 10×2 matrix, and we need to add a leading column of 1's to it. This can be done using `cbind(1,A)`. R *recycles* the `1` so that a matrix of the same number of rows combines with `A` leading to a 10×3 matrix. The `rbind()` function can be used in a similar manner to add a row to a matrix. Because `rbind()` and `cbind()` are generic functions, they can also be useful with other types of objects like data frames.
- The `dimnames` argument of `matrix()` can be used to create names for the row and column dimensions. An example is given in my program.