StartR Linear Algebra and Projection

The computations for performing linear algebra operations are among the most important in science. It’s so important that the unit used to measure computer performance for scientific computation is called a “flop”, standing for “floating point operation” and is defined in terms of a linear algebra calculation.

For you, the issue with using the computer to perform linear algebra is mainly how to set up the problem so that the computer can solve it. The notation that we will use has been chosen specifically to relate to the kinds of problems for which you will be using linear algebra: fitting models to data. This means that the notation will be very compact.

The basic linear algebra operations of importance are:

* project a single vector onto the space defined by a set of vectors.
* take a linear combination of vectors.

In performing these operations, you will use two main functions, project( ) and mat( ), along with the ordinary multiplication \* and addition + operations. There is also a new sort of operation that provides a compact description for taking a linear combination: “matrix multiplication,” written %\*%.

By the end of this StartR session, you should feel comfortable with those two functions and the new form of multiplication %\*%.

To start, consider the sort of linear algebra problem often presented in textbooks in the form of simultaneous linear equations. For example:



Thinking in terms of vectors, this equation can be re-written as



To solve, project = onto the space defined by 1 = and 2 = . The solution, x and y, will be the number of multiples of their respective vectors needed to reach the projected vectors.

When setting this up with the R notation that you will be using, you need to create each of the vectors , 1, and 2. Here’s how:

   b = c(1,1,1)
   v1 = c(1,2,4)
   v2 = c(5,-2,0)

The projection is accomplished using the project( ) function:

   project(b ~ v1 + v2 - 1)

           [,1]
  v1 0.32894737
  v2 0.09210526

Read this as “project onto the subspace defined by 1 and 1. You might well wonder what the -1 is doing there — we’ll get to that in a bit.

The answer is given in the form of the multiplier on 1 and 2, that is, the values of x and y in the original problem. This answer is the “best” in the sense that these particular values for x and y are the ones that come the closest to , that is, the linear combination that give the projection of onto the subspace defined by 1 and 2.

If you want to see what that projection is, just multiply the coefficients by the vectors and add them up. In other words, take the linear combination

   0.32894737\*v1 + 0.09210526\*v2

  [1] 0.7894737 0.4736842 1.3157895

When there are lots of vectors involved in the linear combination, it’s easier to be able to refer to all of them by a single object name. The mat( ) function takes the vectors and packages them together into a matrix. It works just like project( ), but doesn’t involve the vector that’s being projected onto the subspace. Like this:

   A = mat( ~ v1 + v2 - 1 )
   A

       v1 v2
  [1,]  1  5
  [2,]  2 -2
  [3,]  4  0

Notice that A doesn’t have any new information; it’s just the two vectors 1 and 2 placed side by side.

Let’s do the projection again:

   z = project( b ~ v1 + v2 - 1 )
   z

           [,1]
  v1 0.32894737
  v2 0.09210526

To get the linear combination of the vectors in A, you matrix-multiply the matrix A times the solution z:

   A %\*% z

            [,1]
  [1,] 0.7894737
  [2,] 0.4736842
  [3,] 1.3157895

Notice, it’s the same answer you got when you did the multiplication “by hand.”

Now to deal with the mysterious -1. The tilde (~) notation comes from the field of statistics and, in statistical modeling, it’s very, very common to include a particular vector in a model. This vector, called the “intercept,” is simply a vector of all 1s. It’s so common to include the intercept vector in statistics, that the software has been arranged to do this by default. The point of the -1 is to tell the software to override this default and not include the intercept vector. Here’s what you get without the -1:

   A = mat( ~ v1 + v2 )
   A

       (Intercept) v1 v2
  [1,]           1  1  5
  [2,]           1  2 -2
  [3,]           1  4  0

   z = project( b ~ v1 + v2 )
   z

              [,1]
  (Intercept)    1
  v1             0
  v2             0

   A %\*% z

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

Notice that the matrix has a third vector: the intercept vector. The solution consequently has three coefficients. Notice as well that the linear combination of the three vectors exactly reaches the vector . That’s because now there are three vectors that define the subspace: 1, 2, and the intercept vectors of all ones.

**EXAMPLE: The atomic bomb data.**

The data file blastdata.csv contains measurements of the radius of the fireball from an atomic bomb (in meters) versus time (in seconds). In the analysis of these data, it’s appropriate to look for a power-law relationship between radius and time. This will show up as a linear relationship between log-radius and log-time. In other words, we want to find m and b in the relationship log-radius = m log-time +b. This amounts to the projection

   bomb = read.csv("blastdata.csv")
   project( log(radius) ~ log(time), data=bomb )

  (Intercept) 6.2946893
  log(time)   0.3866425

The parameter m is the coefficient on log-time, found to be 0.3866.

**Exercises**

Exercise 1 Remember all those “find the line that goes through the points problems” from algebra class. They can be a bit simpler with the proper linear-algebra tools.

Example: “Find the line that goes through the points (2,3) and (7,-8).”

One way to interpret this is that we are looking for a relationship between x and y such that y = mx + b. In vector terms, this means that the x-coordinates of the two points, 2 and 7, made into a vector will be scaled by m, and an intercept vector will be scaled by b.

   x = c(2,7)
   y = c(3,-8)
   project( y ~ x ) # Remember: intercept included by default

              [,1]
  (Intercept)  7.4
  x           -2.2

Now you know m and b.

YOUR TASK: Using the project( ) function,

(a)

find the line that goes through the two points (9,1) and (3,7).

|  |  |
| --- | --- |
| A  | y = x + 2  |
| B  | y = -x + 10  |
| C  | y = x + 0  |
| D  | y = -x + 0  |
| E  | y = x - 2  |
|  |  |

(b)

find the line that goes through the origin (0,0) and (2,-2).

|  |  |
| --- | --- |
| A  | y = x + 2  |
| B  | y = -x + 10  |
| C  | y = x + 0  |
| D  | y = -x + 0  |
| E  | y = x - 2  |
|  |  |

(c)

find the line that goes through (1,3) and (7,9)

|  |  |
| --- | --- |
| A  | y = x + 2  |
| B  | y = -x + 10  |
| C  | y = x + 0  |
| D  | y = -x + 0  |
| E  | y = x - 2  |
|  |  |

Exercise 2

(a)

Find x, y, and z that solve the following:



(Hint: Remember to suppress the intercept.)

What’s the value of x?:

 -0.2353  0.1617  0.4264  1.3235  1.5739

(b)

Find x, y, and z that solve the following:



What’s the value of x?

 -0.2353  0.1617  0.4264  1.3235  1.5739

Exercise 3 Using project( ), solve these sets of simultaneous linear equations for x, y, and z:

1.

Two equations in two unknowns:



|  |  |
| --- | --- |
| A  | x = 3 and y = -1  |
| B  | x = 1 and y = 3  |
| C  | x = 3 and y = 3  |
|  |  |

2.

Three equations in three unknowns:



|  |  |
| --- | --- |
| A  | x = 3.1644, y = -0.8767, z = 0.8082  |
| B  | x = -0.8767,y = 0.8082, z = 3.1644  |
| C  | x = 0.8082, y = 3.1644, z = -0.8767  |
|  |  |

3.

Four equations in four unknowns:



|  |  |
| --- | --- |
| A  | x = 5.500, y = -7.356, z = 3.6918, w = 1.1096  |
| B  | x = 1.1096, y = 3.6918, z = -7.356, w = 5.500  |
| C  | x = 5.500, y = -7.356, z = 1.1096, w = 3.6918  |
| D  | x = 1.1096, y = -7.356, z = 5.500, w = 3.6918  |
|  |  |

4.

Three equations in four unknowns:



|  |  |
| --- | --- |
| A  | There is no solution.  |
| B  | There is a solution.  |
|  |  |