Directly Working with Normal Equations in R

Arunsoumya Basu

Roll No. bs2029

We have seen that $\widehat{\vec{\beta}} = (\mathbf{X'X})^{-1}\mathbf{X'}\ \vec{y}$ is the least squares solution to the linear model $\vec{y} = \mathbf{X}\ \vec{\beta} + \vec{\epsilon}$. Now, for a real-life problem, computing the product $\mathbf{M} = \mathbf{X'X}$, inverting it, and multiplying the result with $\mathbf{X'}\ \vec{y}$ may seem to be a viable option, but numerical errors might creep in while finding the inverse, particularly if the matrix to be inverted is almost singular (with a determinant which is non-zero, yet very small in absolute value). This would make the end result different than the actual one by quite some margin.

Even if we try to avoid direct inversion and use the '`solve`' function in R in the following manner: `solve(M,v)`, where $\mathbf{M} = \mathbf{X'X}$ and $v = \mathbf{X'}\ \vec{y}$, the issue remains the same since this method is not very stable numerically. In most cases, $\widehat{\vec{\beta}}$ would be reasonably close to the value obtained by using the '`lm`' function in R.

To illustrate, we assume $\mathbf{X} = \begin{bmatrix} 3 & 2 & 5 \\ 7 & 1 & 3 \\ 8 & 1 & 6 \\ 1 & 4 & 2 \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} 18.9 \\ 16.0 \\ 23.1 \\ 17.1 \end{bmatrix}$.

The series of commands

```
X=matrix(c(3,7,8,1,2,1,1,4,5,3,6,2),4,3)
y=c(18.9,16.0,23.1,17.1)
lm(y ~ X-1)$coef
```

generate the output

```
      X1       X2       X3
1.017832 3.025047 1.974191
```

and the series of commands

```
M=t(X)%*%X
v=t(X)%*%y
solve(M,v)
```

generate the output

```
         [,1]
[1,] 1.017832
[2,] 3.025047
[3,] 1.974191
```

Both estimates are same, at least to 6 places after the decimal, i.e., the $\widehat{\vec{\beta}}$ s from `lm` and `solve` agree. However, there may be some cases described earlier (when the matrix $\mathbf{M}$ is nearly singular) where the `lm` function gives a more precise output. This happens because the `lm` function uses QR decomposition to solve the system of equations, which is a numerically stable method.

Further, in situations where the matrix $\mathbf{X}'\mathbf{X}$ is exactly singular, direct inversion or using `solve` fails to provide an answer. But we are guaranteed to obtain an answer since the normal equation is always consistent. In such a scenario the `lm` function gives an output, and indicates by 'NA' the coordinates of $\vec{\beta}$ that can be arbitrarily set to 0. This is demonstrated by the following example where we take

$$\mathbf{X} = \begin{bmatrix} 5 & 3 & 8 \\ 2 & 3 & 5 \\ 1 & 6 & 7 \\ 4 & 2 & 6 \end{bmatrix} \text{ and } \vec{y} = \begin{bmatrix} 34.3 \\ 18.9 \\ 23.1 \\ 26.1 \end{bmatrix}. \text{ Then } \mathbf{X}'\mathbf{X} = \begin{bmatrix} 46 & 35 & 81 \\ 35 & 58 & 93 \\ 81 & 93 & 174 \end{bmatrix}$$

It is to be noted that the first two columns of $\mathbf{X}$ add up to give its third column, so $\mathbf{X}'\mathbf{X}$ is singular (first two columns add up to give the third one). So we will be able to obtain a $\widehat{\vec{\beta}}$, although it will not be unique. In such a case, a similar series of commands

```
X=matrix(c(5,2,1,4,3,3,6,2,8,5,7,6),4,3)
y=c(34.3,18.9,23.1,26.1)
lm(y ~ X-1)$coef
```

generate the output

```
       X1        X2        X3
5.038392 3.000970        NA
```

and the series of commands

```
M=t(X)%*%X
v=t(X)%*%y
solve(M,v)
```

generate the output

```
Error in solve.default(M, v) :
  system is computationally singular: reciprocal condition number = 6.80596e-18
```

Here the output from the `lm` function indicates that the coefficient for X3 can be chosen to be 0, and then X1, X2 can be solved for uniquely. However, it is clear that the `solve` function stops once it encounters singularity in the coefficient matrix, and thus fails to produce a valid choice for $\widehat{\vec{\beta}}$.

`lm` is preferred to `solve` for another reason – in cases where the design matrix is huge, but certain structures or patterns in it are known, the `lm` function can construct the design

matrix internally if the structures are specified, saving us the tedious job of typing the whole matrix and then using `solve`. This aspect shall be explored in greater details in the upcoming sections.