Matrices

Statistics 135

Autumn 2005

Matrix Calculations in Regression

Linear Model:

$$Y = X\beta + \epsilon$$

where

- Responses Y: $n \times 1$ (rows $times$ cols)
- Predictors X: $n \times p$
- Errors ϵ: $n \times 1$

In this formulation n is the number of observations and p is the number of predictors. Usually the first column of X is all 1, making β_1, the first component of β, the intercept. However for what follows, this is not required.

In what follows, it will be assumed that $\mathrm{rank}(X) = p$, which will lead to unique least squares solutions of β. One way of thinking of this, is that no predictor is a linear combination of the rest.

The least squares solutions for β satisfy

$$
\begin{aligned}
X^T X\hat{\beta} &= X^T Y \qquad\qquad \text{(Normal Equations)} \\
\hat{\beta} &= (X^T X)^{-1} X^T Y
\end{aligned}
$$

The vector of fitted values satisfy

$$
\begin{aligned}
\hat{Y} &= X\hat{\beta} \\
&= X(X^T X)^{-1} X^T Y \\
&= HY
\end{aligned}
$$

The matrix H is known as the hat matrix (a $n \times n$ matrix) and is important for many regression calculations and diagnostics

The vector of residuals satisfy

$$e = Y - \hat{Y} = (I - H)Y$$

Two important variance results are

$$\mathrm{Var}(\hat{\beta}) = \sigma^2 (X^T X)^{-1}$$
$$\mathrm{Var}(e) = \sigma^2 (I - H)$$

assuming that $\mathrm{Var}(\epsilon) = \sigma^2 I$ (constant variance and uncorrelated).

The sums of squares decomposition can be calculated by

$$SSR = Y^T \left[H - \frac{1}{n} J \right] Y$$

$$SSE = Y^T (I - H) Y$$

$$SST = Y^T \left[I - \frac{1}{n} J \right] Y$$

where J is a $n \times n$ matrix of all 1's.

The vector $h = \mathrm{diag}(H)$ is known as the leverages. They can be used for the following calculations

$$\mathrm{Var}(\hat{Y}_i) = \sigma^2 h_i$$

$$\mathrm{Var}(e_i) = \sigma^2 (1 - h_i)$$

In addition, h can be used to search for potentially influential observations. For example, values of $h_i > \frac{2p}{n}$ are often considered as outliers in their X values. In addition, Cook's Distance, one common measure of influence depends on h

$$D_i = \frac{e_i^2}{pMSE}\frac{h_i}{(1-h_i)^2}$$

Other common influence measures, such as DFFITS, are also simple functions of h.

Want to use **R** to calculate these statistics. Note that **S** usually doesn't use the following approach to do regression calculations. Instead, they are usually based on the QR decomposition, which can be faster and numerically more stable.

To illustrate this calculations, we will use the Cars data set to examine the model

$$HighFuel_i = \beta_1 + \beta_2 Weight_i + \beta_3 EngSize_i + \epsilon_i$$

Creating Matrices

- `matrix` function

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
          dimnames = NULL)

    data: an optional data vector.

    nrow: the desired number of rows

    ncol: the desired number of columns

   byrow: logical. If 'FALSE' (the default) the matrix is
          filled by columns, otherwise the matrix is filled
          by rows.
```

```
dimnames: A 'dimnames' attribute for the matrix: a 'list'
          of length 2 giving the row and column names
          respectively.
```

The `matrix` function converts vectors to matrices

```
> matrix(1:6, ncol=3)                     # byrow=F is default
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> matrix(1:6, ncol=3, byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

If the dimension of the vector doesn't match `ncol` $\times$ `nrow`, the vector will repeat as many times as necessary to fill the matrix

```
> matrix(1:3, ncol=3, nrow=2)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3

> J <- matrix(1, ncol=2, nrow=2)
> J
     [,1] [,2]
[1,]    1    1
[2,]    1    1
```

- Coerce an object to a matrix

  It is possible to convert other objects into a matrix, with data frames and vectors begin the most common objects to convert. The most common approach is with the `as.matrix` function.

```
as.matrix(x)

       x: an R object.
```

If `x` is a vector, is is converted to a column vector (i.e. a matrix with one column).

When trying to do the conversion, it pick a type that is consistent with all of `x`. This can be important if `x` is a data frame containing a mixture of types.

```
> cars93[1:3,1:4]
   Manu   Model    Type MinPrice
1 Acura Integra   Small     12.9
2 Acura  Legend Midsize     29.2
3  Audi      90 Compact     25.9

> as.matrix(cars93[1:3,1:4])
  Manu    Model     Type      MinPrice
1 "Acura" "Integra" "Small"   "12.9"
2 "Acura" "Legend"  "Midsize" "29.2"
3 "Audi"  "90"      "Compact" "25.9"
```

```
> cars93[1:3,4:6]
  MinPrice MidPrice MaxPrice
1     12.9     15.9     18.8
2     29.2     33.9     38.7
3     25.9     29.1     32.3

> as.matrix(cars93[1:3,4:6])
  MinPrice MidPrice MaxPrice
1     12.9     15.9     18.8
2     29.2     33.9     38.7
3     25.9     29.1     32.3
```

Another option for coercing data frames is the `data.frame` function

```
data.matrix(frame)

   frame: a data frame whose components are logical vectors,
          factors or numeric vectors.
```

Instead of coercing the elements of the frame to the most consistent type, it converts the frame to a numeric matrix. Character strings and factors get converted to numeric codes.

```
> data.matrix(cars93[1:3,1:4])
  Manu Model Type MinPrice
1    1    49    4     12.9
2    1    54    3     29.2
3    2     9    1     25.9
```

- Binding vectors and matrices

  It is also possible to combine vectors and matrices together to make larger matrices with the `cbind` (column bind) and `rbind` (row bind) functions.

```
> A <- cbind(1:3, 4:6)
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> B <- rbind(1:3, 4:6)
> B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

```
> cbind(A, 100:102)
     [,1] [,2] [,3]
[1,]    1    4  100
[2,]    2    5  101
[3,]    3    6  102

> rbind(B, 100:102)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]  100  101  102
```

With `cbind`, vectors get treated as column vectors. Similarly, `rbind` treats vectors as row vectors.

You need to be careful that dimensions match. Vectors will get repeated or truncated to make the binding work.

```
> cbind(A, 100)
     [,1] [,2] [,3]
[1,]    1    4  100
[2,]    2    5  100
[3,]    3    6  100

> cbind(A,B)
Error in cbind(...) : number of rows of matrices must match
(see arg 2)

> rbind(B, 100:103)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]  100  101  102
Warning message:
number of columns of result
  not a multiple of vector length (arg 2) in: rbind(B, 100:103)
```

- Special matrix functions

  The most useful of these (and the only one I can think of right now) is the `diag` function. This function has three purposes, to make a diagonal matrix from a vector, create identity matrices, and extract the diagonal element from a matrix (not necessarily square).

```
> diag(1:3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3

> diag(3)       # a 3 x 3 identity matrix
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

```
> C <- cbind(A, 100:102)
> C
     [,1] [,2] [,3]
[1,]    1    4  100
[2,]    2    5  101
[3,]    3    6  102
> diag(C)
[1]   1   5 102

> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> diag(A)
[1] 1 5
```

Regression Example

```
> attach(cars93)

> HighFuel <- 100/HighMPG

> n <- length(HighFuel)

> In <- diag(n)
> J <- matrix(1, ncol=n, nrow=n)

> Y <- matrix(HighFuel, ncol=1)
> X <- cbind(Intercept=rep(1,n), Weight, EngSize)
> X[1:4,]
     Intercept Weight EngSize
[1,]         1   2705     1.8
[2,]         1   3560     3.2
[3,]         1   3375     2.8
[4,]         1   3405     2.8
```

```
> p <- dim(X)[2]  # dim gives the dimension of a matrix.
                  # The first component is the number of rows
                  # The second is the number of columns
> p
[1] 3

> betahat <- solve(t(X) %*% X, t(X) %*% Y)

  # t(X) calculates the transpose of X

> XtXinv <- solve(t(X) %*% X)

  # solve(A) gives inverse of A

> betahat2 <- XtXinv %*% t(X) %*% Y
```

```
> betahat    # usually the better approach

                  [,1]
Intercept  0.717936352
Weight     0.001045061
EngSize   -0.145369340
> betahat2
                  [,1]
Intercept  0.717936352
Weight     0.001045061
EngSize   -0.145369340

> betahat - betahat2
                   [,1]
Intercept -1.268985e-13
Weight     4.119968e-17
EngSize   -5.856426e-15

> H <- X %*% XtXinv %*% t(X)
```

```
> h <-  diag(H)

> fits <- H %*% Y
> resids <- (In - H) %*% Y

> SSR <- t(Y) %*% (H - J/n) %*% Y
> SSE <- t(Y) %*% (In - H) %*% Y
> SST <- t(Y) %*% (In - J/n) %*% Y

> SSR <- as.numeric(SSR)  # need to convert to scalar
> SSE <- as.numeric(SSE)
> SST <- as.numeric(SST)

> MSE <- SSE / (n - p)

> varbeta <- MSE * XtXinv   # estimated variance matrix
> sefits <- sqrt(MSE * h)
> seresid <- sqrt(MSE * (1 - h))
```

`b <- solve(A,x)` **vs** `b <- solve(A) %*% x`

The above **S** code are equivalent approaches to solving the system of equations

$$Ab = x$$

Usually the first approach is preferable for two reasons

- Lower computational burden

- Numerically more stable - fewer computational errors creep in

So where possible, use `solve` instead of computing inverse and multiplying.

This result holds for any programming language, **S**, **MATLAB**, c, Fortran, etc.

For example, the hat matrix can be calculated two ways in **R**.

```
H <- X %*% XtXinv %*% t(X)
```

```
H2 <- X %*% solve(t(X) %*% X, t(X))
```

The second approach is preferable as it eliminates a matrix multiplication.

In some cases you do want to calculate the inverse. One example is to get

$$\widehat{\mathrm{Var}}(\hat{\beta}) = MSE(X^T X)^{-1}$$

which was calculated by `MSE * XtXinv` in the example code.