Matrix Operations and Functions

Thomas J. Leeper

May 19, 2015

1 R Basics

Getting help

- `?` and `??`
- StackOverflow: `http://stackoverflow.com/questions/tagged/r`

Differences between R and Stata

- Multiple datasets at once
- Store lots of other kinds of objects simultaneously
- Core set of functionality, with lots of functionality available in add-on packages
- Open source! You can look under the hood at everything
- R is a full-fledged programming language, whereas Stata is a data analysis package
- R is primarily a "functional" programming language, which means that actions are performed by specifying an input to a function and storing the output of that function in a new object
- This allows for non-destructive analysis (often no need to go back and redo all steps if you make a mistake)

R Console vs RStudio

- Two popular ways to use R: console/GUI and RStudio
- R console or GUI is a basic interface

- RStudio is an "integrated development environment" meant to make it easier to do some things
- None of these have point and click menus
- We'll work in the console

Basic operations of the console

- Type commands and press enter (like in Stata)
- Line continuation
- Whitespace
- Previous commands
- Command completion
- Clearing previous results
- Limited menus

- Objects, naming, classes

```
x <- 1
x = 1
logical()
integer()
numeric()
character()
factor()
class(1L)
```

- basic math

```
2 + 2
1 - 3
4 * 5
6 / 3
2 ^ 2

# precedence
2 + 3 * 4
3 * 4 + 2
(2 + 3) * 4
2 + (3 * 4)
```

- vectors

```
c(1,2,3)
1:3
c(1,3:4)

c(TRUE, FALSE)

c(1L, 2L)

# missing values
NA

c(1,NA)
c(NA,TRUE)

NULL

c(1,NULL,3)
```

- coercion

```
as.numeric(TRUE)
as.integer(TRUE)
as.logical(1)
as.logical(-1)
as.logical(3)
as.integer(1.5)
```

- logicals and boolean operations

```
TRUE | TRUE
TRUE | FALSE
FALSE | FALSE
TRUE & TRUE
TRUE & FALSE
FALSE & FALSE

!TRUE
!FALSE

TRUE | 1L
FALSE | 1L
TRUE | 0L

TRUE + TRUE
TRUE - FALSE

TRUE - (2*TRUE)


# equality
1 == 1
1 == 1L
TRUE == 1L
2 == 3
```

```
2 == TRUE

1 != 1
1 != 0
TRUE != 3

4 %in% 1:5
3:5 %in% c(2,4,6)
!3:5 %in% c(2,4,6)
```

- vector indexing (positional, logical, named)

```
x <- letters
x[1]
x[2]
x[1:3]
x[c(1,3,5)]

# seq
seq_along(x)
seq(1,5)

# rep
rep(1, 3)
rep(1:2, times = 3)
rep(1:2, each = 3)

# indexing beyond range
c(1,3,5)[2]
c(1,3,5)[4]

# negative indexing
c(1,3,5)[-1]

# vectorized operations
(1:3) + 1
# but remember precedence:
1:3 * 2
1:(3 * 2)
1:3[2]
(1:3)[2]

# recycling
1:4 + 1:2
1:3 + 1:2


# named indexing
x <- 1:3
names(x)
names(x) <- c('one','two','three')
names(x)
x
x[1]
x['one']
x <- c(a = 1, b = 3, c = d)
```

```
x['b']
```

- A `data.frame` is a collection of equal-length vectors.

```
# built-in data.frames
mtcars
iris

# data I/O
install.packages('rio')
library('rio')
```

Questions?

2 Matrices and OLS Estimation

If we have the following matrix:

$$\begin{bmatrix} -1 & 3 \\ 2 & -4 \end{bmatrix} \tag{1}$$

We represent that in R as a vector of values passed to the `matrix()` function, which regulates the dimensions of the matrix:

```
matrix(c(-1, 2, 3, -4), nrow = 2)

# other examples
matrix(1:6)
matrix(1:6, nrow=2)
matrix(1:6, ncol=3)
matrix(1:6, nrow=2, ncol=3)
matrix(1:6, nrow=2, ncol=3, byrow=TRUE)

a <- matrix(1:10, nrow=2)
length(a)
nrow(a)
ncol(a)
dim(a)

rbind(1:3,4:6,7:9)
cbind(1:3,4:6,7:9)

# transpose
t(rbind(1:3,4:6,7:9))
```

Manually calculate the transpose of a matrix!

Some special matrices:

```
# identity matrix
diag(3)
# be careful with ‘diag()‘. it does many different things

# row vector
matrix(1:5, nrow = 1)

# column vector
matrix(1:5, ncol = 1)
```

Matrix indexing:

```
b <- rbind(1:3,4:6,7:9)
b[1,]   #’ first row
b[,1]   #’ first column
b[1,1]  #’ element in first row and first column

# vector (positional) indexing
b[1:2,]
b[1:2,2:3]

# negative indexing
b[-1,2:3]

# logical indexing
b[c(TRUE,TRUE,FALSE),]
b[,c(TRUE,FALSE,TRUE)]

# ‘diag‘ for extraction
diag(b)
upper.tri(b)    #’ upper triangle
b[upper.tri(b)]
lower.tri(b)    #’ lower triangle
b[lower.tri(b)]
```

Matrix operations:

```
# scalar addition/subtraction
a <- matrix(1:6, nrow=2)
a + 1
a - 1
a + 1:2
a - 1:2

# scalar multiplication/division
a * 2
a / 2
a * 1:2
a / 1:2

# additivity property:  (X + Y)’ = X’ + Y’
A <- matrix(1:6, nrow = 3)
B <- matrix(7:12, nrow = 3)
```

```
A + B
t(A + B)
t(A) + t(B)
```

2.1 OLS in matrix notation

We start with a matrix representation of our regression equation:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{e}$$

We need to minimize the sum of squared residuals, $\mathbf{e}$, so we flip our equation $\mathbf{y} = \mathbf{Xb} + \mathbf{e}$ to be $\mathbf{e} = \mathbf{y} - \mathbf{Xb}$. This gives us the residuals from one estimation of our coefficients $\mathbf{b}$. We don't want to minimize the residuals, though, instead we want to minimize the sum of squared residuals. Conveniently, matrix notation is a very easy way of representing that because $\mathbf{e}$ is a vector and the sum of a squared vector is just the dot product (i.e., vector times itself; or a column vector times a row vector representation).

$$\begin{aligned} S(\mathbf{b}) &= \mathbf{e}'\mathbf{e} \\ &= (\mathbf{y} - \mathbf{Xb})'(\mathbf{y} - \mathbf{Xb}) \\ &= (\mathbf{y}' - \mathbf{b}'\mathbf{X}')(\mathbf{y} - \mathbf{Xb}) \\ &= \mathbf{y}'\mathbf{y} - \mathbf{y}'\mathbf{Xb} - \mathbf{b}'\mathbf{X}'\mathbf{y} + \mathbf{b}'\mathbf{X}'\mathbf{Xb} \\ &= \mathbf{y}'\mathbf{y} - 2\mathbf{y}'\mathbf{Xb} + \mathbf{b}'\mathbf{X}'\mathbf{Xb} \end{aligned}$$

Last step above is because $\mathbf{b}'\mathbf{X}'\mathbf{y}$ is 1x1, thus equal to its own transpose. How do we minimize an equation? To minimize, we set differentiate $S(\mathbf{b})$, set to 0, and solve for b.

$$
\begin{aligned}
\frac{\partial S(b)}{\partial b} &= 0 - 2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{X}\mathbf{b} \\
0 &= -2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{X}\mathbf{b} \\
2\mathbf{X}'\mathbf{y} &= 2\mathbf{X}'\mathbf{X}\mathbf{b} \\
\mathbf{X}'\mathbf{y} &= \mathbf{X}'\mathbf{X}\mathbf{b} \\
(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} &= \mathbf{b}
\end{aligned}
$$

If we assume $\mathbf{y}$ is a linear function of $\mathbf{Xb}$, then $\mathbf{b}$ is an unbiased estimator of β (i.e., $E(\mathbf{b}) = \beta$). Then variance of our coefficient estimates is:

$$
\begin{aligned}
V(\mathbf{b}) &= \left[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\right] V(\mathbf{y}) \left[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\right] \\
&= \left[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\right] \sigma^2\mathbf{I}_n \left[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\right] \\
&= \sigma^2(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} \\
&= \sigma^2(\mathbf{X}'\mathbf{X})^{-1}
\end{aligned}
$$

Note assumptions of constant error variance. Other types of standard errors (e.g., clustered, heteroskedasticity-robust) would replace $V(y)$ with something else.

What do we need to know to be able to calculate this?

- Creating a matrix to represent our data
- Scalar multiplication
- Matrix transpose
- Matrix multiplication
- Matrix inversion

- Matrix indexing in R and extraction of a matrix diagonal

2.2 Data as a design matrix

A design matrix is 1 observation per row, one variable per column. In order to represent complex terms (power terms, interactions, transformations, indicator/dummy variables), we have to transform our original data.frame into a a purely numeric matrix, with no missing data, one column for each term in the model, and a column of 1's to estimate the intercept. There can also be no linear dependence between columns.

```
mtcars

X <- as.matrix(mtcars[,c('vs', 'am')])
X

# interactions
X[,1] * X[,2]
X <- cbind(X, X[,1] * X[,2])

# power terms, etc.
mtcars[,'mpg'] ^ 2

# categorical variables
mtcars[,"cyl"]

mtcars[,"cyl"] == 4
mtcars[,"cyl"] == 6
mtcars[,"cyl"] == 8
as.numeric(mtcars[,"cyl"] == 4)

model.matrix(~ 0 + factor(cyl), data = mtcars)

# intercept!
cbind(1, X)
```

A major issue with constructing a design matrix is missingness. By default, Stata, R, and other packages will use *available case analysis*. We cannot have missing values if we want to calculate most statistics or, for example, estimate a regression model, so this is a reasonable approach but if we construct the design matrix manually, we also need to deal with missingness manually.

```
# missing data
NA
x <- c(1,2,NA,4)
x
```

```
# simple, single imputation
is.na(x)
x[is.na(x)]
x[is.na(x)] <- mean(x, na.rm = TRUE)

# extracting complete cases
d <- data.frame(a = 1:4, b = c(1,2,NA,4), c = c(1,NA,3,4))
complete.cases(d)
d[complete.cases(d), ]
```

2.3 Matrix multiplication

This is the dot product we saw earlier with vectors generalized to work for matrices.

Matrices are conformable (i.e., can be multiplied) if they are *m-by-n* and *n-by-p*. So number of columns of the first matrix must match number of rows of the second matrix. Otherwise, non-conformable.

The output matrix from matrix multiplication is an *m-by-p* matrix.

```
# inner product (or dot product) of two vectors
1:3 %*% 2:4
1*2 + 2*3 + 3*4

# dot product only works for conformable vectors or matrices!
1:3 %*% 1:2

mm <- function(A,B){
    m <- nrow(A)
    n <- ncol(B)
    C <- matrix(NA,nrow=m,ncol=n)
    for(i in 1:m) {
        for(j in 1:n) {
            C[i,j] <- paste('(',A[i,],'*',B[,j],')',sep='',collapse='+')
        }
    }
    print(C, quote=FALSE)
}

# conformability
A <- matrix(1:6, nrow = 3)
B <- matrix(2:5, nrow = 2)

A %*% B
B %*% A # non-conformable
mm(A,B)

# dimension of output
A <- matrix(1:30, ncol = 3)
B <- matrix(1:3, nrow = 3)
A %*% B
mm(A,B)
```

```
# multiplication
A <- matrix(rep(2, 6), ncol = 2)
B <- matrix(1:4, nrow = 2)
# DO THIS BY HAND
mm(A,B)

A <- matrix(c(5,3,2,1,7,4), ncol = 2)
B <- matrix(c(2,5,6,2,1,7,4,1), nrow = 2)
# DO THIS BY HAND
#A %*% B
#mm(A,B)
```

Multiplication of a diagonal matrix is nice because of lots of zeros. Similar for an upper or lower triangular matrix.

Multiplication by identity matrix is just itself.

```
# multiply by diagonal matrix
A <- matrix(1:12, ncol = 3)
B <- diag(1:3)
A %*% B

# multiplication by an I identity matrix
matrix(1:9, nrow = 3) %*% diag(3)
```

Some properties:

- Associative: $(\mathbf{XY})\mathbf{Z} = \mathbf{X}(\mathbf{YZ})$
- Distributive: $(\mathbf{X} + \mathbf{Y})\mathbf{Z} = (\mathbf{XZ}) + (\mathbf{YZ})$
- Scalar distributive: $s(\mathbf{XY}) = (\mathbf{X}s\mathbf{Y}) = (\mathbf{XY})s$
- Additive: $(\mathbf{X} + \mathbf{Y})' = \mathbf{X}' + \mathbf{Y}'$
- Multiplicative: $(\mathbf{XY})' = \mathbf{Y}'\mathbf{X}'$, note reversed order

2.4 Matrix inversion

Given a square matrix $\mathbf{A}$, if it is invertible, another matrix $\mathbf{B}$ exists such that:

$$\mathbf{A}^{-1} = \mathbf{B}$$

$$\mathbf{AB} = \mathbf{AA}^{-1} = \mathbf{I}$$

Also important is how matrix multiplication and matrix inversion work together:

$$\mathbf{B}^{-1}\mathbf{A}^{-1}\mathbf{AB} = \mathbf{B}^{-1}\mathbf{IB} = \mathbf{B} = \mathbf{ABB}^{-1}\mathbf{A}^{-1}$$

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

Note how the order of the terms in the product are reversed by inversion.

Inverse of transpose is the transpose of the inverse.

An **invertible** matrix has an inverse. This is also called a **nonsingular** matrix. A matrix that does not have an inverse is **noninvertible** or **singular**.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

An example (of a 2x2 matrix):

$$\begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix}^{-1} = \frac{1}{(1 * 2) - (1 * -1)} \begin{bmatrix} 2 & -(-1) \\ -(1) & 1 \end{bmatrix}$$

Do this inversion by hand. First find the determinant, then follow the formula from above:

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}^{-1} = \frac{1}{(2 * 5) - (3 * 4)} \begin{bmatrix} 5 & -3 \\ -4 & 2 \end{bmatrix} = \begin{bmatrix} -2.5 & 1.5 \\ 2.0 & -1.0 \end{bmatrix}$$

Formula for larger matrices is more complex. For a 3x3 matrix:

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} (ei - fh) & -(bi - ch) & (bf - ce) \\ -(di - fg) & (ai - cg) & -(af - cd) \\ (dh - eg) & -(ah - bg) & (ae - bd) \end{bmatrix}$$

The determinant for a 2x2 is $ad - bc$. For a 3x3 it is more complex. It requires extracting the **minors** of the matrix and multiplying the first-row value of the matrix times the determinant of each minor, summed using **cofactors** (alternating +1 and -1 matrix). For our purposes, `det` determines the determinant of a matrix

Special case: determinant of a triangular matrix is just the product of the diagonal entries. A triangular matrix is only singular if any of the diagonals are zero.

Not all square matrices are invertible. (Non-square matrices do not have inverses.) Is not invertible if its determinant is zero. When does this occur? It is if-and-only-if if either the columns (or rows) are linear functions of each other (i.e., if the matrix is *rank-deficient*).

```
A <- matrix(1:9, nrow = 3)
det(A)
solve(A)

A <- matrix(c(1:3,2,1,3,4,6,8), nrow = 3)
det(A)
solve(A)

1/det(A)
```

Try inverting matrix `A` by hand using the formula.

In practice, we rarely invert matrices. Instead, we use a decomposition or factorization. Common decompositions are Cholesky, LU, and QR. The **QR decomposition** is used by most regression solving algorithms. It decomposes **X** into two matrices, **QR**, where **Q** is an m-by-n *orthonormal* matrix and **R** is an m-by-m upper-triangular matrix.

An *orthnormal* matrix has the nice property: $\mathbf{M}'\mathbf{M} = \mathbf{I}$. (Note: the matrix times its transpose is **I**, whereas the general result is that a matrix times its inverse is the identity matrix. So, an orthonormal matrix has an inverse equal to its transpose.)

$$\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$
$$(\mathbf{QR})'\mathbf{QR}\beta = (\mathbf{QR})'\mathbf{y}$$
$$\mathbf{R}'\mathbf{Q}'\mathbf{QR}\beta = \mathbf{R}'\mathbf{Q}'\mathbf{y}$$
$$\mathbf{R}'\mathbf{R}\beta = \mathbf{R}'\mathbf{Q}'\mathbf{y}$$
$$\beta = \mathbf{R}^{-1}\mathbf{Q}'\mathbf{y}$$

So we just have to invert the $\mathbf{R}$ upper-triangular matrix, which is easier because we can use a process called `back substitution` (the details of which are unimportant). The inverse $\mathbf{R}^{-1}$ is also upper-triangular, which is (as we know from earlier) also easier to multiply.

We won't go over the details of the decomposition, but know that while we talk about estimating OLS using the full matrix notation, in practice we estimate it using a QR decomposition.

```
QR <- qr(X)
R <- qr.R(QR)
solve(R) # inverse of the R matrix
Q <- qr.Q(QR)
```

Questions?

3 Formulae

Regression notation is R relies on an object class called "formula". Formulas are similar to regression notation in Stata but they provide more options for expressing complex regression models.

```
mtcars

mpg ~ hp
mpg ~ hp + wt
mpg ~ hp + wt + am + wt:am
```

```
mpg ~ hp + wt*am

# we can save formula for use
f1 <- mpg ~ hp + wt*am
class(f1)
lm(mpg ~ hp + wt*am, data = mtcars)
lm(f1, data = mtcars)

# intercept-only model
mpg ~ 1
lm(mpg ~ 1, data = mtcars)
sd(mtcars$mpg)

# dropping an intercept
mpg ~ 0 + hp


# I() notation
lm(mpg ~ hp + I(hp^2), data = mtcars)


# factor() notation to create categorical/dummy variables
lm(mpg ~ hp + factor(cyl), data = mtcars)

# `model.matrix`
model.matrix(~ 0 + factor(cyl), data = mtcars)
```