Matrix Basics I

Matrix Algebra for Data Analysis

Gaston **S**anchez
gastonsanchez.com

Readme

License:

This document is licensed under a

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

You are free to:

Share — copy and redistribute the material

Adapt — rebuild and transform the material

Under the following conditions:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made.

NonCommercial — You may not use this work for commercial purposes.

ShareAlike — If you remix, transform, or build upon this work, you must distribute your contributions under the same license to this one.

Matrix Algebra?

Why?

Matrix algebra is **fundamental** for a good understanding of Multivariate Data Analysis Methods.

- Multivariate data is commonly represented in **tabular format** (rows and columns)
- Mathematically, a data table can be treated as a **matrix**
- Matrix algebra provides the **analytical machinery and tools** to manipulate and exploit data

Content of the Course

Set of slides

1. Matrix Basics
2. Orthogonal matrices and projections
3. Rank and Inverse
4. Matrix Decompositions
5. Quadratic and bilinear forms

Goals of this Course

Fundamentals

Aim is for you to learn more about matrix algebra and how to use the R language/environment to express matrix computations

Tools

Equip you with the tools needed for

- doing data-matrix operations
- programming multivariate methods
- becoming a data analysis padawan

Considerations

Caveat

This course will not teach you everything you need to know about matrix algebra. It will just get you started.

In other words

The idea is to introduce you to a broad range of topics that are of value for multivariate data analysis, but not necessarily go into great depth.

Matrix Basics

Content of the present slides

1. Matrices in R
2. Matrix transpose
3. Shapes of matrices
4. Matrix addition and multiplication
5. Traces and determinants

Matrices in R

Matrices in R

Basic functions in R for matrix objects

Function	Description
`matrix()`	create a matrix
`dim()`	dimension of a matrix
`nrow()`	number of rows
`ncol()`	number of columns
`as.matrix()`	convert into matrix
`is.matrix()`	test if the argument is a matrix

Bear in mind that R can do some things that matrix algebra cannot: row-column naming, handling `NA`'s, and recycling.

Matrix Recap

```r
# matrix
A = matrix(1:12, nrow=4, ncol=3)

# add row names
rownames(A) = c("a", "b", "c", "d")

# add column names
colnames(A) = c("one", "two", "three")

A
```

```
##   one two three
## a   1   5     9
## b   2   6    10
## c   3   7    11
## d   4   8    12
```

```r
# test class
is.matrix(A)
```

```
## [1] TRUE
```

```r
# dimensions
dim(A)
```

```
## [1] 4 3
```

```r
# number of rows
nrow(A)
```

```
## [1] 4
```

```r
# number of columns
ncol(A)
```

```
## [1] 3
```

```r
# first row
A[1,]
```

```
##   one   two three
##     1     5     9
```


Matrix Recap (con't)

Recycling a vector into matrix

```
# vector
b = 1:3

# dimension
dim(b)

## NULL

# test class matrix?
is.matrix(b)

## [1] FALSE

# recycling
(B = matrix(b, nrow=4, ncol=3))

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    3    1
## [3,]    3    1    2
## [4,]    1    2    3
```

Missing values

```
# test class matrix?
is.matrix(B)

## [1] TRUE

# missing values
B[1, 1] = NA
B[4, 3] = NA
B

##      [,1] [,2] [,3]
## [1,]   NA    2    3
## [2,]    2    3    1
## [3,]    3    1    2
## [4,]    1    2   NA
```

Matrix Transpose

The transpose of a $n \times p$ matrix $\mathbf{X}$ is the $p \times n$ matrix $\mathbf{X}'$
In R the transpose is given by the function `t()`

```
# matrix X
X = matrix(1:12, 4, 3)
X

##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
# transpose of X
t(X)

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

Matrices and Vectors in R

Good to know

It is important to distinguish vectors and matrices, especially in R.

In matrix algebra we use the convention that vectors are column vectors (i.e. they are $n \times 1$ matrices).

In R, a vector with n elements is not the same as an $n \times 1$ matrix, because an R matrix has the dimensions attribute, and an R vector does not.

Matrices and Vectors in R (con't)

Good to know

Vectors in R behave more like row vectors.

However, depending on the type of functions you apply to vectors, sometimes R will handle vectors like if they were column vectors.

Also, in R numbers are actually vectors with a single element.

From scalar to matrix

Scalar

```
# scalar
x = 1

# dim
dim(x)

## NULL


# test if matrix
is.matrix(x)

## [1] FALSE
```

Scalar into 1×1 matrix

```
# convert to matrix
xx = matrix(x, 1, 1)
xx

##      [,1]
## [1,]    1


dim(xx)

## [1] 1 1
```

Shapes of Matrices

Rectangular matrix

The general *shape* of a matrix is a **rectangular** $n \times p$ matrix (n number of rows, p number of columns)

```
# rectangular matrix
Rectangular = matrix(runif(15), nrow = 3, ncol = 5)
Rectangular

##        [,1]   [,2]   [,3]    [,4]   [,5]
## [1,] 0.2655 0.9082 0.9447 0.06179 0.6870
## [2,] 0.3721 0.2017 0.6608 0.20597 0.3841
## [3,] 0.5729 0.8984 0.6291 0.17656 0.7698

# dimensions
dim(Rectangular)

## [1] 3 5
```

Rectangular Tall matrix

A **rectangular** matrix with $n > p$ is commonly known as a **tall** matrix

```
# tall matrix
Tall = matrix(runif(21), nrow = 7, ncol = 3)
Tall

##        [,1]   [,2]    [,3]
## [1,] 0.1849 0.8334 0.40528
## [2,] 0.7024 0.4680 0.85355
## [3,] 0.5733 0.5500 0.97640
## [4,] 0.1681 0.5527 0.22583
## [5,] 0.9438 0.2389 0.44481
## [6,] 0.9435 0.7605 0.07498
## [7,] 0.1292 0.1808 0.66190
```

Rectangular Wide matrix

A **rectangular** matrix with $n < p$ is commonly known as a **wide** matrix

```
# wide matrix
Wide = matrix(runif(21), nrow = 3, ncol = 7)
Wide

##        [,1]   [,2]   [,3]  [,4]   [,5]   [,6]   [,7]
## [1,] 0.1680 0.3277 0.1246 0.631 0.5340 0.8297 0.8975
## [2,] 0.8075 0.6021 0.2946 0.512 0.5572 0.1114 0.2797
## [3,] 0.3849 0.6044 0.5776 0.505 0.8679 0.7037 0.2282
```

Square matrix

A matrix $\mathbf{X}$ is **square** if the number of rows is equal to the number of columns (i.e. $n = p$)

```
# square matrix
Square = matrix(runif(9), nrow = 3, ncol = 3)
Square

##           [,1]   [,2]   [,3]
## [1,] 0.585800 0.2774 0.7244
## [2,] 0.008946 0.8136 0.9061
## [3,] 0.293740 0.2604 0.9490

# same number of rows and columns
dim(Square)

## [1] 3 3
```

Symmetric matrix

A square matrix $\mathbf{X}$ is **symmetric** if $x_{ij} = x_{ji}$ for all i and j, that is if $\mathbf{X} = \mathbf{X}'$. A square matrix is *asymmetric* if it is not symmetric.

```
# square matrix
Symmetric = matrix(c(1, 2, 3, 2, 1, 4, 3, 4, 1), 3, 3)
Symmetric

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    1    4
## [3,]    3    4    1

# transpose
t(Symmetric)

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    1    4
## [3,]    3    4    1
```

Diagonal matrix

A square matrix $\mathbf{X}$ is **diagonal** if $x_{ij} = 0$ for all $i \neq j$. Thus a diagonal matrix is symmetric.

In R we can create diagonal matrices with `diag()`

```
# diagonal matrix
Diagonal = diag(runif(3))
Diagonal

##         [,1]   [,2]  [,3]
## [1,] 0.07314 0.0000 0.000
## [2,] 0.00000 0.7547 0.000
## [3,] 0.00000 0.0000 0.286
```

We can also use `diag()` to extrac the diagonal from a square matrix

```
# diagonal matrix
diag(Square)

## [1] 0.5858 0.8136 0.9490
```

Identity matrix

A diagonal matrix is the **identity matrix** if all diagonal elements are equal to one.

We can also use `diag()` to create identity matrices:

```
# identity matrix
Identity = diag(1, 3, 3)
Identity

##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

Upper Triangular matrix

A matrix is **upper triangular** if $x_{ij} = 0$ for all $i > j$

```
# upper triangular
Upper_Triang = matrix(c(1, 0, 0, 2, 4, 0, 3, 5, 6), 3, 3)
Upper_Triang

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    0    4    5
## [3,]    0    0    6
```

Lower Triangular matrix

A matrix is **lower triangular** if $x_{ij} = 0$ for all $i < j$

```
# lower triangular
Lower_Triang = matrix(c(1, 0, 0, 2, 3, 0, 4, 5, 6), 3, 3, byrow = TRUE)
Lower_Triang
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    2    3    0
## [3,]    4    5    6
```

Upper and Lower Triangular parts

Upper Triangular part

Given a matrix, we can extract its **upper triangular** part with the help of `lower.tri()`

```
# square matrix
m = matrix(1:9, 3, 3)
# lower triangular part
# (in logical form)
lower.tri(m)

##       [,1]  [,2]  [,3]
## [1,] FALSE FALSE FALSE
## [2,]  TRUE FALSE FALSE
## [3,]  TRUE  TRUE FALSE
```

```
# extract upper triangular part
m[lower.tri(m)] <- 0
m

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    0    5    8
## [3,]    0    0    9
```

Upper and Lower Triangular parts (con't)

Lower Triangular part

Given a matrix, we can extract its **lower triangular** part with the help of `upper.tri()`

```
# square matrix
m = matrix(1:9, 3, 3)
# upper triangular part
# (in logical form)
upper.tri(m)

##       [,1]  [,2]  [,3]
## [1,] FALSE  TRUE  TRUE
## [2,] FALSE FALSE  TRUE
## [3,] FALSE FALSE FALSE
```

```
# extract lower triangular part
m[upper.tri(m)] <- 0
m

##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    2    5    0
## [3,]    3    6    9
```

Basic Matrix Operations

Basic Matrix Operations

Let's start with the basic matrix operations in R:

- addition
- scalar multiplication
- matrix-matrix multiplication
- matrix-vector multiplication

Matrix Addition

`A + B`

Matrix addition of two matrices $\mathbf{A} + \mathbf{B}$ is defined when $\mathbf{A}$ and $\mathbf{B}$ have the same dimensions:

```
# matrix A (2,3)
A = matrix(1:6, 2, 3)

# matrix B (2, 3)
B = matrix(7:9, 2, 3)

A + B
```

```
##      [,1] [,2] [,3]
## [1,]    8   12   13
## [2,]   10   11   15
```

Scalar Multiplication

`0.5 * X`

We can multiply a matrix by a scalar using the usual product operator `*`, moreover it doesn't matter if we pre-multiply or post-multiply:

```r
# matrix X (3,4)
X = matrix(1:3, 3, 4)

# (pre)multiply X by 0.5
(1/2) * X

##      [,1] [,2] [,3] [,4]
## [1,]  0.5  0.5  0.5  0.5
## [2,]  1.0  1.0  1.0  1.0
## [3,]  1.5  1.5  1.5  1.5
```

```r
# matrix X (3,4)
X = matrix(1:3, 3, 4)

# (post)multiply X by 0.5
X * 0.5

##      [,1] [,2] [,3] [,4]
## [1,]  0.5  0.5  0.5  0.5
## [2,]  1.0  1.0  1.0  1.0
## [3,]  1.5  1.5  1.5  1.5
```

Matrix-Matrix Multiplication

A %*% B

The matrix product operator in R is %*%. We can multiply matrices $\mathbf{A}$ and $\mathbf{B}$ if the number of columns of $\mathbf{A}$ is equal to the number of rows of $\mathbf{B}$

```
# matrix A (2,3)
(A = matrix(1:6, 2, 3))

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

```
# matrix B (3, 2)
(B = matrix(7:9, 3, 2))

##      [,1] [,2]
## [1,]    7    7
## [2,]    8    8
## [3,]    9    9
```

```
# product AB (2, 2)
AB = A %*% B
AB

##      [,1] [,2]
## [1,]   76   76
## [2,]  100  100
```

```
# product BA (2, 2)
BA = B %*% A
BA

##      [,1] [,2] [,3]
## [1,]   21   49   77
## [2,]   24   56   88
## [3,]   27   63   99
```

Matrix-Matrix Multiplication (con't)

`A %*% (B %*% C)`

Matrix multiplication is **associative**: $\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}$
(Obvisouly, the dimensions must conform to the matrix product)

```
# associative
A %*% (B %*% AB)

##       [,1]  [,2]
## [1,] 13376 13376
## [2,] 17600 17600
```

```
# associative
(A %*% B) %*% AB

##       [,1]  [,2]
## [1,] 13376 13376
## [2,] 17600 17600
```

Matrix-Matrix Multiplication (con't)

`A %*% (B + C)`

Matrix multiplication is **distributive over addition**:

$$\mathbf{A}(\mathbf{B}+\mathbf{C}) = (\mathbf{AB}) + (\mathbf{AC})$$

$$(\mathbf{A}+\mathbf{B})\mathbf{C} = (\mathbf{AC}) + (\mathbf{BC})$$

```r
# distributive
D = t(A)
A %*% (B + D)

##      [,1] [,2]
## [1,]  111  120
## [2,]  144  156
```

```r
# distributive
(A %*% B) + (A %*% D)

##      [,1] [,2]
## [1,]  111  120
## [2,]  144  156
```

Cross Products

`t(X) %*% X`, `X %*% t(X)`

A very common type of products in multivariate data analysis are $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}\mathbf{X}'$, sometimes known as **crossproducts**.

```
# X'X
t(X) %*% X

##      [,1] [,2] [,3] [,4]
## [1,]   14   14   14   14
## [2,]   14   14   14   14
## [3,]   14   14   14   14
## [4,]   14   14   14   14
```

```
# XX'
X %*% t(X)

##      [,1] [,2] [,3]
## [1,]    4    8   12
## [2,]    8   16   24
## [3,]   12   24   36
```

Cross Products (con't)

In R we have the functions `crossprod()` and `tcrossprod()` which are formally equivalent to:

- `crossprod(X,X)` $\equiv$ `t(X) %*% X)`
- `tcrossprod(X,X)` $\equiv$ `X %*% t(X)`

```
# X'X
crossprod(X, X)

##      [,1] [,2] [,3] [,4]
## [1,]   14   14   14   14
## [2,]   14   14   14   14
## [3,]   14   14   14   14
## [4,]   14   14   14   14
```

```
# XX'
tcrossprod(X, X)

##      [,1] [,2] [,3]
## [1,]    4    8   12
## [2,]    8   16   24
## [3,]   12   24   36
```

However, `crossprod()` and `tcrossprod()` are usually slightly faster than using `t()` and `%*%`

Matrix-Vector Multiplication

We can post-multiply an $n \times p$ matrix $\mathbf{X}$ with a vector $\mathbf{b}$ with p elements. This means making **linear combinations** (weighted sums) of the columns of $\mathbf{X}$:

```
# matrix X (4,3)
(X = matrix(1:12, 3, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12


# vector b (length 4)
(b = seq(0.25, 1, by = 0.25))

## [1] 0.25 0.50 0.75 1.00
```

```
# product: Xb
X %*% b

##      [,1]
## [1,] 17.5
## [2,] 20.0
## [3,] 22.5
```


Vector-Matrix Multiplication

We can pre-multiply a vector $\mathbf{a}$ (with n elements) with an $n \times p$ matrix $\mathbf{X}$. This means making **linear combinations** (weighted sums) of the rows of $\mathbf{X}$:

```
# matrix X (4,3)
(X = matrix(1:12, 3, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12


# vector a (length 3)
(a = 1:3)

## [1] 1 2 3
```

```
# product: a'X
a %*% X

##      [,1] [,2] [,3] [,4]
## [1,]   14   32   50   68


# product: a'X
t(a) %*% X

##      [,1] [,2] [,3] [,4]
## [1,]   14   32   50   68
```

Matrix and Vector Multiplications

Notice that when use the product operator %*% R is smart enough to use the convention that vectors are $n \times 1$ matrices.

Notice also that if we ask for a vector-matrix multiplication, we can use both formulas:

1. `a %*% X`
2. `t(a) %*% X`

(R will reformat the n vector as an $n \times 1$ matrix first)