Lab 3: Matrix Algebra

Table of Contents

You can download the lab here.

Purpose

The purpose of today's lab is to learn how to enter matrices and perform operations on matrices in R, and to use R to expand your theoretical understanding of matrices and how they underlie statistical calculations. For the minihacks, you will be applying what you learned in lecture about using data matrices to calculate means and covariance matrices.

Today's lab will cover:

1. Review of matrix types
2. How to create a matrix
3. Testing for equality
4. Transpose
5. Addition/subtraction
6. Multiplication
7. Diagonal Matrices

Review of matrix types

Notation and Terminology:

A matrix is an object that stores information in rows and columns. For a given matrix A, we use a_{rc} to refer to the entry at row r and column c. "Order" refers to the dimensions of a matrix: the number of rows and the number of columns. Here are six types of matrices:

1. Rectangular: A matrix where the number of rows does not equal the number of columns.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix}$$

2. Square: A matrix where the number of rows equals the number of columns.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

3. Vector: A matrix where the row or column (not both) is 1.

$$A = \begin{bmatrix} 3 & 1 & 2 \end{bmatrix}$$

4. Diagonal: A square matrix where all of the elements equal zero except for those making up the principal diagonal.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

5. Identity: A diagonal matrix with 1s along the principal diagonal.

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6. Null: A matrix that consists entirely of 0s.

$$0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

How to create a matrix

We create matrices in R by using the `matrix()` function. To do this, we need to provide the `data` (i.e., the elements, or numbers in the matrix), the number of rows (`nrow`), the number of columns (`ncol`), and then tell it whether the order of elements is entered `byrow` or not. If you're entering it row-wise, you want to set `byrow = TRUE`. If you're entering it column-wise, you want to set `byrow = FALSE` or leave this argument blank (as `byrow = FALSE` by default).

Let's practice creating a few matrices:

```
a_mat <- matrix(data = c(1, 2, 3,
                         1, 2, 3),
                nrow = 2, ncol = 3, byrow = TRUE) # spacing is irrelevant
                                                  # but I find it easier to
read
a_mat # print a_mat

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

b_mat <- matrix(data = c(1, 1,
                         2, 2,
                         3, 3),
                nrow = 3, ncol = 2, byrow = TRUE)
b_mat # print b_mat

##      [,1] [,2]
## [1,]    1    1
```

```
## [2,]    2    2
## [3,]    3    3
```

```
c_mat <- matrix(data = c(1, 1,
                         2, 4,
                         3, 5),
                nrow = 3, ncol = 2, byrow = TRUE)
c_mat # print c_mat
```

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

```
d_mat <- matrix(data = c(1, 1,
                         2, 2,
                         3, 3),
                nrow = 3, ncol = 2, byrow = TRUE)
d_mat # print d_mat
```

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

Naming rows and columns

If you want the rows and columns of your matrix to have labels, you can specify these using the `dimnames` argument of the `matrix()` function. Commonly, you would want to name your columns (which generally correspond to variables) but not your rows (which generally correspond to observations).

For example, let's re-create `a_mat` from above, but give our columns the names `var1`, `var2` and `var3`.

```
a_mat_named <- matrix(data = c(1, 2, 3,
                               1, 2, 3),
                      nrow = 2, ncol = 3, byrow = TRUE,
                      dimnames = list(NULL, c("var1", "var2", "var3")))

a_mat_named
```

```
##      var1 var2 var3
## [1,]    1    2    3
## [2,]    1    2    3
```

Notice that the `dimnames` argument expects a list (hence the use of `list()`) of length 2 that gives the names of the rows and columns, respectively. Since we didn't want to add row names, we set the first element of the list to `NULL`.

`rep()` function

The `rep()` function allows you to replicate values, which can come in handy when creating matrices. For example, let's say we wanted to create the following matrix in R:

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This would be a pain to type out manually using the `matrix()` function. Instead, because we have repeating values in our matrix, we can use `rep()` to create each of the rows in our matrix must faster.

To use `rep()` you specify `x` = the value that you want to replicate and `times` = the number of times you want to replicate it.

e.g. `rep(x = 1, times = 10)` would result in 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

Now let's use `rep()` and `matrix()` to create matrix `M` from above:

```
M <- matrix(c(rep(1, times = 16),
              rep(0, times = 8),
              rep(2, times = 8),
              rep(3, times = 16),
              rep(4, times = 8),
              rep(0,times = 8)),
            nrow = 4,
            byrow = TRUE)

M
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
[,14]
## [1,]    1    1    1    1    1    1    1    1    1     1     1     1     1
1
## [2,]    0    0    0    0    0    0    0    0    2     2     2     2     2
2
## [3,]    3    3    3    3    3    3    3    3    3     3     3     3     3
3
## [4,]    4    4    4    4    4    4    4    4    0     0     0     0     0
0
##      [,15] [,16]
## [1,]     1     1
## [2,]     2     2
## [3,]     3     3
## [4,]     0     0
```

Testing for equality

Equality of dimensions

We can check whether or not the order of two matrices are the same using a combination of `dim()`, which returns the dimensions of the matrix (rows, columns) and the equality test `==`.

NOTE: `=` defines something (arguments in functions & objects) and `==` tests for equality.

We get two results for each test, which correspond to (in order): 1. Do the matrices have the same number of rows? 2. Do the matrices have the same number of columns?

First let's check `A` against the others

```
# check A against the others
dim(a_mat) == dim(b_mat)

## [1] FALSE FALSE

dim(a_mat) == dim(c_mat)

## [1] FALSE FALSE

dim(a_mat) == dim(d_mat)

## [1] FALSE FALSE
```

Recall,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 1 \\ 2 & 4 \\ 3 & 5 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}$$

As we knew (from looking at them ourselves), `A` (2 x 3) does not have the same order as `B`, `C`, and `D` (3 x 2).

Next we'll check `B` against `C` and `D`:

```
# Check B against C and D.
dim(b_mat) == dim(c_mat)
```

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

```
dim(b_mat) == dim(d_mat)
```

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

Equality of elements

The dimensions of `B`, `C`, and `D` are the same, so now let's test the equality of the elements.

```
b_mat == c_mat
```

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

```
b_mat == d_mat
```

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

We can see that `B` and `D` are identical matrices because they have the exact same elements. This is a simple trivial example, because we could already tell they were the same just by looking at them – however, testing the equality of matrices can be very useful when you have much larger matrices that you can't compare by eye.

Transpose

The transpose of a matrix is a matrix that is flipped over its principal diagonal.

Example:

$$X = \begin{bmatrix} 1 & 2 \\ 5 & 1 \\ 7 & 2 \end{bmatrix}$$

To create the transpose of X, denoted X', the following happens:

1. The first column becomes the first row.
2. The second column becomes the second row.

Question: Since the order of the above matrix, B, is 3x2, what will the order of its transpose be?

$$X' = \begin{bmatrix} 1 & 5 & 7 \\ 2 & 1 & 2 \end{bmatrix}$$

Transpose in R

To transpose a matrix in R, you can use the function `t()`. Let's take a look at the transpose of matrix `X` from above.

```
# create the matrix
X <- matrix(c(1, 2,
              5, 1,
              7, 2),
              nrow = 3, ncol = 2, byrow = TRUE)

# view the matrix
X
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    5    1
## [3,]    7    2
```

```
# take the transpose of the matrix
t(X)
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    7
## [2,]    2    1    2
```

Addition/Subtraction

Matrices of the same order can be added and subtracted. Recall from lecture the rules about addition and subtraction:

Matrix addition is **commutative**...

$$\large \mathbf{A}+\mathbf{B}=\mathbf{B}+\mathbf{A}$$

...and **associative**:
$$\large \mathbf{A} + \mathbf{B} + \mathbf{C} = (\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$$
Matrix subtraction is **distributive**

$$\large \mathbf{A} – (\mathbf{B} + \mathbf{C}) = \mathbf{A} – \mathbf{B} – \mathbf{C}$$

$$\large \mathbf{A} – (\mathbf{B} – \mathbf{C}) = \mathbf{A} – \mathbf{B} + \mathbf{C}$$

Note: matrix subtraction is "distributive" because $(B + C)$ is being multiplied by the scalar -1. So, $-(B + C)$ becomes $(-B - C)$, and since addition is associative, $A + (-B - C) = A - B - C$.

Let's test out these rules using R!

```
# define some matrices to use as examples
A <- matrix(data = c(6,1, 2,10), nrow = 2, ncol = 2, byrow = T)
B <- matrix(data = c(2,1, 1,6), nrow = 2, ncol = 2, byrow = T)
C <- matrix(data = c(4,1, 3,2), nrow = 2, ncol = 2, byrow = T)
```

Let's see if the following statements are true…

1) A + B = B + A ???

2) (A + B) + C = A + (B + C) ???

3) A - B = B - A ???

4) A - (B - C) = A - B - C ???

5) A - (B - C) = A - B + C ???

Multiplication

Two matrices are "conformable for multiplication" if they have dimensions allowing them to be multiplied. Specifically, the number of columns of the first matrix must be equal to the number of rows of the second matrix.

Example 1

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \\ 4 & 6 \end{bmatrix}, B = \begin{bmatrix} 1 & 5 & 8 & 3 \\ 4 & 2 & 6 & 4 \end{bmatrix}$$

1. What is the order of A? And what about B?
2. Is AB conformable?
3. Is BA conformable?
4. For AB, what will the order of the resulting matrix be?

$$AB = \begin{bmatrix} (1 \times 1) + (3 \times 4) & (1 \times 5) + (3 \times 2) & (1 \times 8) + (3 \times 6) & (1 \times 3) + (3 \times 4) \\ (2 \times 1) + (1 \times 4) & (2 \times 5) + (1 \times 2) & (2 \times 8) + (1 \times 6) & (2 \times 3) + (1 \times 4) \\ (4 \times 1) + (6 \times 4) & (4 \times 5) + (6 \times 2) & (4 \times 8) + (6 \times 6) & (4 \times 3) + (6 \times 4) \end{bmatrix}$$

$$AB = \begin{bmatrix} 13 & 11 & 26 & 15 \\ 6 & 12 & 22 & 10 \\ 28 & 32 & 68 & 36 \end{bmatrix}$$

Example 1 in R

```
# Define the matrices
A <- matrix(data = c(1, 3,
                     2, 1,
                     4, 6),
```

```
            nrow = 3, ncol = 2, byrow = TRUE)

B <- matrix(data = c(1, 5, 8, 3,
                     4, 2, 6, 4),
            nrow = 2, ncol = 4, byrow = TRUE)
```

To multiply matrices in R, we have to use the special matrix multiplication operator, `%*%`

```
#multiply A by B
A %*% B

##      [,1] [,2] [,3] [,4]
## [1,]   13   11   26   15
## [2,]    6   12   22   10
## [3,]   28   32   68   36
```

Question: What would happen if we tried `B %*% A`?

Example 2

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 2 \\ 4 & 1 \end{bmatrix}, B = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 2 & 4 \end{bmatrix}$$

1. What is the order of A? And what about B?
2. Is AB conformable?
3. is BA conformable?
4. Does $AB = BA$? In other words, does the matrix product possess the commutative property?

Example 2 in R

```
# Define the matrices
A <- matrix(data = c(1, 2,
                     3, 2,
                     4, 1),
            nrow = 3, ncol = 2, byrow = TRUE)

B <- matrix(data = c(2, 3, 4,
                     1, 2, 4),
            nrow = 2, ncol = 3, byrow = TRUE)

# multiply A by B
A %*% B

##      [,1] [,2] [,3]
## [1,]    4    7   12
## [2,]    8   13   20
## [3,]    9   14   20
```

```
# multiply B by A
B %*% A

##      [,1] [,2]
## [1,]   27   14
## [2,]   23   10
```

Question: What would happen if we tried `A %*% B == B %*% A`?

Note: Matrix multiplication *does* possess the associative property: $A(BC) = (AB)C$.

Identity Matrix

The **identity matrix** is a diagonal matrix with 1s along the principal diagonal. For example…

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Let's take an example matrix `C`:

$$C = \begin{bmatrix} 2 & 5 \\ 4 & 3 \end{bmatrix}$$

1. If C is multiplied by its identity matrix (I_2), what will the resulting matrix be?
2. Does $CI_2 = I_2C$?

Identity Matrix in R

The simplest way to get an identity matrix in R is to use the `diag()` function.

First we'll create a matrix `C`.

```
C <- matrix(data = c(2, 5,
                     4, 3),
            nrow = 2, ncol = 2, byrow = TRUE)
C

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

And next we can get the identity matrix for `C`, or I_2. We want the identity matrix to have the same number of rows and columns as `C`.

```
id_mat <- diag(x = 1, nrow = nrow(C), ncol = ncol(C))

# Or we could have hard coded it:
#id_mat <- diag(x = 1, nrow = 2, ncol = 2)

#diag() will default to a square matrix if you only define nrow or ncol but
not both
```

And finally, multiply them together with %*%

```
C %*% id_mat

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

And we could test if $CI_2 = I_2C$

```
C %*% id_mat == id_mat %*% C

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

Note: Multiplying by the identity matrix is a special case in which the commutative property holds true for matrix multiplication.

Diagonal Matrices

The identity matrix is a special case of a diagonal matrix. A diagonal matrix is a matrix in which the entries outside the main diagonal are all zero. In the case of the identity matrix, all of the diagonal elements are 1's. But we can create diagonal matrices that contain other values as well, again using the `diag()` function.

Creating diagonal matrices

To create a diagonal matrix with 4 rows containing 2's all along the diagonal, we would do the following:

```
diag_twos <- diag(x = 2, nrow = 4)

diag_twos

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

To create a diagonal matrix with different values along the diagonal, you can assign a vector to the `x` argument in `diag`. Notice that I no longer needed to specify the number of rows because `diag` defaults to a square matrix.

```
diag_vary <- diag(x = c(5,6,1))

diag_vary
```

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

Pre-multiplication

Pre-multiplication of a matrix X by a diagonal matrix D results in the **rows** of X being multiplied by the corresponding diagonal element in D.

Let's use the following example for X:

```
X <- matrix(c(1, 2, 3,
              4, 5, 6),
            nrow = 2, ncol = 3, byrow = TRUE)

X
```

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

First we'll create our diagonal matrix that we will pre-multiply X by.

```
A <- diag(x = c(2, 3))

A
```

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

Now let's pre-multiply. This will result in multiplying all elements in the first row by 2 and all elements in the second row by 3.

```
A %*% X
```

```
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]   12   15   18
```

When working with data, pre-multiplication of a data matrix X by another matrix results in a linear combination of your rows in X. When your rows are your participants, pre-multiplication would result in a linear combination of your participants. This may be useful, for example, if you want to calculate grand means or groups means for a particular variable.

Post-multiplication

Post-multiplication of a matrix X by a diagonal matrix D results in the **columns** of X being multiplied by the corresponding diagonal element in D.

Let's use the same matrix X as our example.

```
X

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

Now we'll create our diagonal matrix that we will post-multiply X by.

```
B <- diag(x = c(2,3,4))

B

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

Now let's post-multiply by X. This will result in multiplying the first column in X by 2, the second column in X by 3, and the third column in X by 4.

```
X %*% B

##      [,1] [,2] [,3]
## [1,]    2    6   12
## [2,]    8   15   24
```

When working with data, post-multiplication of a data matrix X by another matrix results in a linear combination of your columns in X. When your columns are your variables, post-multiplication would result in a linear combination of your variables. This may be useful, for example, if you want to create a composite score for a scale.

Simultaneous multiplication

We can also simultaneously pre- and post-multiply a matrix. For example, the code below will simultaneously multiply the first row by 2, the second row by 3, the first column by 2, the second column by 3, and the third column by 4.

```
X

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

A %*% X %*% B

##      [,1] [,2] [,3]
## [1,]    4   12   24
## [2,]   24   45   72
```

Calculating means with pre-multiplication

For this example, we will be working with a dataset from lecture. In the dataset, there are five participants and three variables. Note that for the group code, `0` is the treatment group and `1` is the control group. First, we will input the dataset into R as a matrix.

```
X = matrix(c(rep(0, times = 3,),
             rep(1, times = 2),
             5, 3, 5, 8, 5,
             45, 34, 27, 32, 71),
           byrow = FALSE,
           nrow = 5,
           dimnames = list(NULL, c("group", "var1", "var2")))
X
```

```
##      group var1 var2
## [1,]     0    5   45
## [2,]     0    3   34
## [3,]     0    5   27
## [4,]     1    8   32
## [5,]     1    5   71
```

Next, using the transformation matrix `T`, we will calculate the grand means for each variable, the group means for each group for each variable, and the difference between groups for each variable.

$$T = \begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \frac{1}{n} \\ \frac{1}{n_{G1}} & \frac{1}{n_{G1}} & \frac{1}{n_{G1}} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{n_{G2}} & \frac{1}{n_{G2}} \\ \frac{1}{n_{G1}} & \frac{1}{n_{G1}} & \frac{1}{n_{G1}} & -\frac{1}{n_{G2}} & -\frac{1}{n_{G2}} \end{bmatrix}$$

Recall, pre-multiplication creates linear combinations of the participants. So, in order to calculate a mean for `var1`, we want to add all of the observations together and divide by n. That is why each entry of the first row of the transformation matrix is $1/n$. The second row will calculate the means of the variables for the treatment group only. The `0`'s in the second row ensure that the control group will not be counted in calculating the treatment group mean. The third row will calculate the means of the variables for the control group only. The final row will calculate the difference in means between the treatment and control group on each variable.

In order to create the transformation matrix, `T`, we need to know the total n and the n of each group.

```
n_total <- nrow(X)
n_total
```

```
## [1] 5
```

```
n_treat <- sum(X[,"group"]==0)
n_treat
```

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

```
n_control = sum(X[,"group"]==1)
n_control
```

```
## [1] 2
```

```
T <- matrix(c(rep(1/n_total, times = n_total),
              rep(1/n_treat, times = n_treat),
              rep(0, times = n_control),
              rep(0, times = n_treat),
              rep(1/n_control, times = n_control),
              rep(1/n_treat, times = n_treat),
              rep(-1/n_control, times = n_control)),
            nrow = 4,
            byrow = TRUE)
T
```

```
##           [,1]      [,2]      [,3] [,4] [,5]
## [1,] 0.2000000 0.2000000 0.2000000  0.2  0.2
## [2,] 0.3333333 0.3333333 0.3333333  0.0  0.0
## [3,] 0.0000000 0.0000000 0.0000000  0.5  0.5
## [4,] 0.3333333 0.3333333 0.3333333 -0.5 -0.5
```

Now, I will calculate the means by multiplying the transformation matrix and the data matrix. Note: I am excluding the first column because it is the group code (treatment group or control group).

```
means <- T %*% X[,2:3]

dimnames(means) <- list(c("grand means","treatment means", "control group
means", "difference in means"), c("var1", "var2"))

means
```

```
##                          var1      var2
## grand means          5.200000  41.80000
## treatment means      4.333333  35.33333
## control group means  6.500000  51.50000
## difference in means -2.166667 -16.16667
```

We can double check the grand means and the group means using the psych package that you used in lab last week.

```
library(psych)
```

```
#double check grand means
describe(X)

##       vars n mean    sd median trimmed   mad min max range skew kurtosis
se
## group    1 5  0.4  0.55      0     0.4  0.00   0   1     1 0.29    -2.25
0.24
## var1     2 5  5.2  1.79      5     5.2  0.00   3   8     5 0.39    -1.34
0.80
## var2     3 5 41.8 17.60     34    41.8 10.38  27  71    44 0.74    -1.36
7.87

#double check group means
describeBy(X, group = "group")

##
##  Descriptive statistics by group
## INDICES: 0
##       vars n  mean   sd median trimmed   mad min max range  skew kurtosis
se
## group    1 3  0.00 0.00      0    0.00  0.00   0   0     0   NaN      NaN
0.00
## var1     2 3  4.33 1.15      5    4.33  0.00   3   5     2 -0.38    -2.33
0.67
## var2     3 3 35.33 9.07     34   35.33 10.38  27  45    18  0.14    -2.33
5.24
## ------------------------------------------------------------
## INDICES: 1
##       vars n mean    sd median trimmed   mad min max range skew kurtosis
se
## group    1 2  1.0  0.00    1.0     1.0  0.00   1   1     0  NaN      NaN
0.0
## var1     2 2  6.5  2.12    6.5     6.5  2.22   5   8     3    0    -2.75
1.5
## var2     3 2 51.5 27.58   51.5    51.5 28.91  32  71    39    0    -2.75
19.5
```

Minihacks

The minihacks for today are based on the following example:

You run an experiment to test the effectiveness of an memory-improvement intervention. As part of the study, participants take a difficult memory test at time 1, then participate in the intervention, and then retake the memory test one week later. Half the participants are assigned to a control condition, and half are assigned to the intervention.

Use the following code to create a matrix representing a dataset of scores, $X_{10,3}$:

```
X = matrix(c(45, 52, 54, 52, 50, 72, 43, 56, 62, 47,
             41, 51, 57, 52, 45, 83, 55, 70, 75, 57,
```

```
            0, 0, 0, 0, 0, 1, 1, 1, 1, 1),
          ncol = 3, byrow = F,
          dimnames = list(NULL, c("time_1", "time_2", "group")))
X
```

```
##       time_1 time_2 group
##  [1,]     45     41     0
##  [2,]     52     51     0
##  [3,]     54     57     0
##  [4,]     52     52     0
##  [5,]     50     45     0
##  [6,]     72     83     1
##  [7,]     43     55     1
##  [8,]     56     70     1
##  [9,]     62     75     1
## [10,]     47     57     1
```

You should now have a matrix with three columns, named `time_1` (memory score at time 1), `time_2` (memory score at time 2), and `group` (group, 0 = control, 1 = intervention).

Minihack 1: Identifying linear combinations

1. Take a look at the following code. Don't run the matrix multiplication step yet. What do you think the linear combination represented below will accomplish?

```
A <- matrix(rep(1), ncol = nrow(X))

B <- diag(1/(nrow(X)) , nrow = ncol(X))

A %*% X %*% B
```

2. Now run this code and check whether your intuition was correct. You should be able to check your answer by using another function that you've already learned.

Minihack 2: Creating your own linear combinations

Use matrix algebra to do the following:

1. Using post-multiplication, create a new vector representing the difference between `time_1` and `time2`.
2. Using pre-multiplication, calculate the average `time_1` score for the whole sample.
3. Using pre-multiplication, calculate the average `time_1` score for each group.

Minihack 3: Calculate a covariance matrix

Recall from lecture that a covariance matrix is a square, symmetric matrix that contains variances on the principal diagonal, and covariances off the principle diagonal. Recall that,

for a sample, the formula for variance is $s^2 = \frac{\sum(X_i - \overline{X})^2}{N-1}$ and the formula for covariance is $s_{XY} = \frac{\sum(X_i - \overline{X})(Y_i - \overline{Y})}{N-1}$.

1. Create a 10x3 matrix `C` where each entry in each column is the respective column mean of the data matrix `X`. Hint: `colMeans()` is a quick way to get the means of each column of a matrix.

2. Create the matrix of deviation scores `D` by subtracting `C` from `X`.

3. Calculate `D'D` (the transpose of `D` times `D`.) Store this calculation in a temporary matrix.

4. What you have now is a matrix of numerators from your variance and covariance formulas (the sum of squares on the principle diagonal and the sum of cross products off the principle diagonal). What do you need to multiple each entry in your matrix by in order to do get a covariance matrix? Hint: the answer is in the formulas.

5. Complete calculating the covariance matrix by multiplying the temporary matrix you calculated in part 3 and the scalar that you identified in part 4.

6. Use the `cov()` function to show that you calculated the covariance matrix correctly.