Matrix algebra using R

Dr. Noah Mutai

05 January 2023

Table of Contents

Matrices

A matrix is a collection of data elements of the same type arranged in a two-dimensional rectangle.

To create a matrix we must indicate the elements, as well as the number of rows (nrow) and columns (ncol).

To declare a matrix in R use the function mat () and a name. As a rule of thumb, matrix names are capitalized. However, R takes even lower case.

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

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

To check the components of the mat () function use;

```
# ?matrix
```

By default, any matrix is created column-wise. To change that we set an additional argumnet byrow = TRUE.

```
A <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
A

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

It is not necessary to specify both the number of rows and columns We can only indicate one of them. The number of elements must be a multiple of the number of rows or columns

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

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

B <- matrix(1:9, ncol = 3)
B

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

To get the class use the fucntion class()

```
class(A)

## [1] "matrix" "array"
```

Other functions include

dim() dimesnion of matrix nrow total rows ncol total columns

example;

```
dim(A)

## [1] 3 3

nrow(A)

## [1] 3

ncol(A)

## [1] 3
```

To check is somethin is a matrix use;

```
is.matrix(A)

## [1] TRUE
```

rbind() AND cbind()

rbind() and cbind() allow us to bind vectors in order to create a matrix. The vectors must have the same length

Example;

Declare three vectors.

```
a <- c(1,2,3,4)
b <- c(10,11,12,13)
c <- c(20,30,40,50)
```

If we use rbind(), our vectors will be rows

```
e <- rbind(a, b, c)
e

##   [,1] [,2] [,3] [,4]
## a    1    2    3    4
## b   10   11   12   13
## c   20   30   40   50
```

The result is a matrix.

```
class(e)

## [1] "matrix" "array"
```

The order does not matter.

```
e <- rbind(c, a, b)
e

##   [,1] [,2] [,3] [,4]
## c   20   30   40   50
```

```
## a    1    2    3    4
## b   10   11   12   13
```

Vectors can be repeated.

```
e <- rbind(a, b, c, a)
e

##   [,1] [,2] [,3] [,4]
## a    1    2    3    4
## b   10   11   12   13
## c   20   30   40   50
## a    1    2    3    4
```

It is not necessary to create the vectors first. We can enter them directly in the rbind() function.

```
e <- rbind(c(1,2,3), c(7,8,9), c(2,3,4))
e

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

If we use cbind() the vectors will be columns.

```
e <- cbind(a, b, c)
e

##      a  b  c
## [1,] 1 10 20
## [2,] 2 11 30
## [3,] 3 12 40
## [4,] 4 13 50
```

Naming the rows and columns of a matrix

Naming with dimnames()

```
e <- matrix(c(1,2,3,4,5,6), nrow = 2,
            dimnames = list(c("row1", "row2"), c("col1", "col2", "col3")))
e

##      col1 col2 col3
## row1    1    3    5
## row2    2    4    6
```

Using the functions rownames() and colnames()

```
e <- matrix(c(1,2,3,4,5,6), nrow = 2)
rownames(e) <- c("row1", "row2")
```

```
colnames(e) <- c("col1", "col2", "col3")
e

##      col1 col2 col3
## row1    1    3    5
## row2    2    4    6
```

Remove row and column names, assign NULL.

```
rownames(e) <- NULL
colnames(e) <- NULL
e

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

Indexing matrices

Indexing means accessing one or several matrix elements. Indices must be put between square brackets We must use two indices: one for the row and the other one for the column

```
e <- matrix(1:16, nrow = 4, byrow = TRUE)
e

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

Access the element on row 3, column 2

```
e[3,2]

## [1] 10
```

Access the element on row 4, column 1

```
e[4,1]

## [1] 13
```

Operations on matrices

Matrix multiplication

Scalar multiplication

```
s <- 10
s*A

##      [,1] [,2] [,3]
## [1,]   10   40   70
```

```
## [2,]   20   50   80
## [3,]   30   60   90
```

Matrix multiplication

```
A%*%B

##      [,1] [,2] [,3]
## [1,]   30   66  102
## [2,]   36   81  126
## [3,]   42   96  150
```

Matrix addition and substraction

```
A + B

##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
## [3,]    6   12   18

A - B

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

Transpose

```
t(A)

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

t(B)

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

Inverse

If it exist;

```
A <- matrix(c(2,4,6,-1,2,-1,10,11,12), ncol = 3, nrow = 3)
solve(A)

##            [,1]        [,2]        [,3]
## [1,] -0.3240741 -0.01851852  0.28703704
## [2,] -0.1666667  0.33333333 -0.16666667
## [3,]  0.1481481  0.03703704 -0.07407407
```

Determinant

```
det(A)

## [1] -108
```

Other operations

Combining matrices;

Rowwise combination.

```
rbind(A,B)

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

Columnwise combination.

```
cbind(A,B)

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

Row and column sums.

```
colSums(A)

## [1] 12  0 33

rowSums(A)

## [1] 11 17 17
```

Row and column means.

```
colMeans(A)

## [1]  4  0 11

rowMeans(A)

## [1] 3.666667 5.666667 5.666667
```

Some types of matrices.

Identity matrix

```
C <- diag(3)
C

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

Unity matrix

```
U <- matrix(1, 3, 2)
U

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

Sytem of linear equations

Use R package matlib

Install and load matlib R package.

```
# install.packages("matlib")
library(matlib)

## Warning: package 'matlib' was built under R version 4.2.2
```

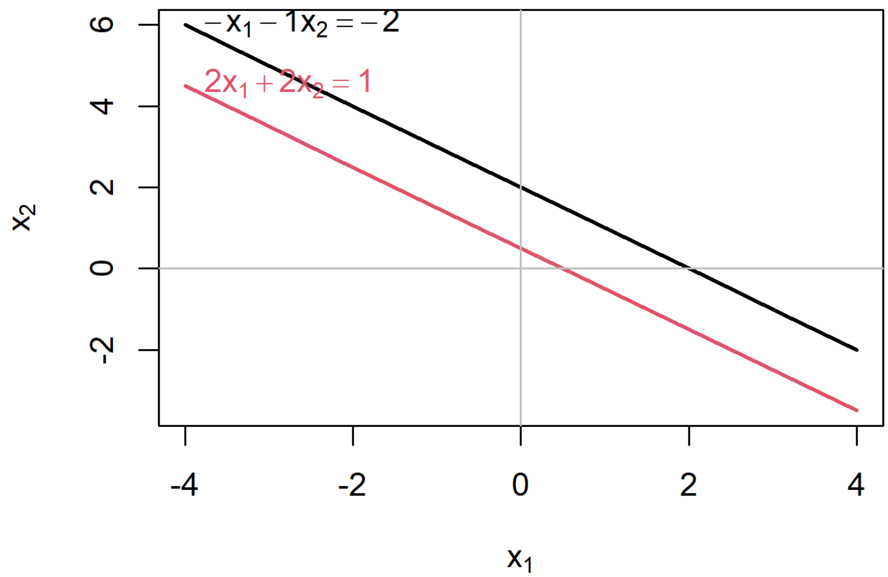
Example

```
A <- matrix(c(-1, 2, -1, 2), 2, 2)
b <- c(-2,1)
showEqn(A, b)

## -1*x1 - 1*x2  =  -2
##  2*x1 + 2*x2  =   1

plotEqn(A,b)

##  -x[1] - 1*x[2]  =  -2
## 2*x[1] + 2*x[2]  =   1
```




```r
Solve(A, b, fractions = TRUE)

## x1 + x2  =   1/2
##        0  =  -3/2
```

Example;

Solve the equation; $\begin{array}{rcrcrcr} 4x & - & 3y & + & z & = & -10 \\ 2x & + & y & + & 3z & = & 0 \\ -x & + & 2y & - & 5z & = & 17 \end{array}$

```r
A <- matrix(c(4,-3,1,2,1,3,-1,2,-5), nrow = 3, ncol = 3)
b <- c(-10,0,17)
showEqn(A,b)

##  4*x1 + 2*x2 - 1*x3  =  -10
## -3*x1 + 1*x2 + 2*x3  =    0
##  1*x1 + 3*x2 - 5*x3  =   17

Solve(A, b, fractions = TRUE)

## x1          =  -13/4
##    x2       =   -3/4
##       x3    =   -9/2

plotEqn3d(A,b)

A <- matrix(c(1,-2,4,-5,2,3,6,2,4), nrow = 3, ncol = 3)
b <- c(23,45,32)
```

```
# plotEqn3d (A,b)
Solve(A, b, fractions = TRUE)
```

```
## x1       =   -526/81
##   x2     =    490/81
##     x3  =  1613/162
```

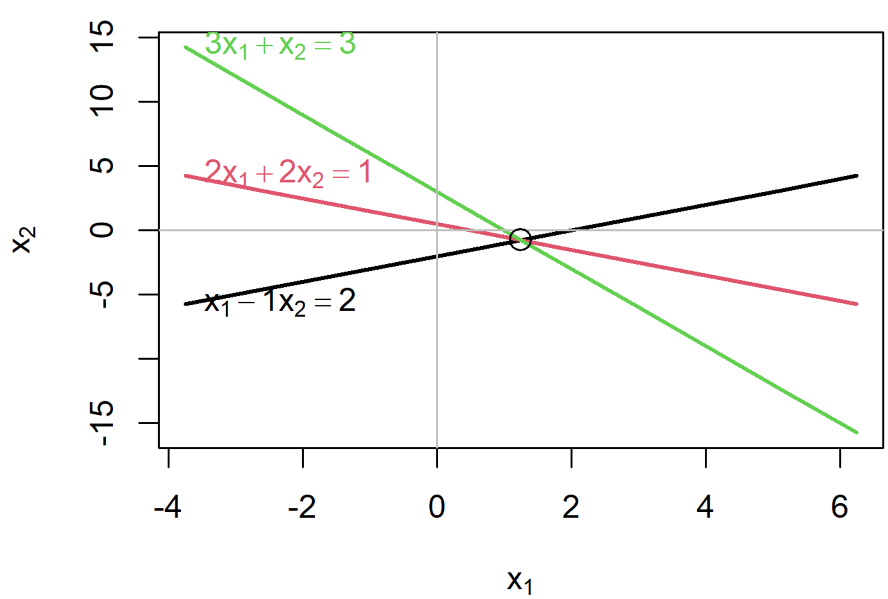
Another example;

```
A <- matrix(c(1,2,3, -1, 2, 1), 3, 2)
b <- c(2,1,3)
showEqn(A, b)
```

```
## 1*x1 - 1*x2  =  2
## 2*x1 + 2*x2  =  1
## 3*x1 + 1*x2  =  3
```

```
plotEqn(A,b)
```

```
##   x[1] - 1*x[2]  =  2
## 2*x[1] + 2*x[2]  =  1
## 3*x[1]   + x[2]  =  3
```




```
Solve(A, b, fractions=TRUE)
```

```
## x1    =   5/4
##   x2  =  -3/4
##    0  =     0
```

Applying functions to matrices

To perform operations on the matrix rows and columns we can use the apply() function

Create a matrix.

```
A <- matrix(10:25, nrow = 4)
A

##      [,1] [,2] [,3] [,4]
## [1,]   10   14   18   22
## [2,]   11   15   19   23
## [3,]   12   16   20   24
## [4,]   13   17   21   25
```

Compute the sum of the elements on each row and column, respectively

```
apply(A, 1, sum)

## [1] 64 68 72 76

apply(A, 2, sum)

## [1] 46 62 78 94
```

Compute the product of the elements on each row and column, respectively.

```
apply(A, 1, prod)

## [1]  55440  72105  92160 116025

apply(A, 2, prod)

## [1]  17160  57120 143640 303600
```

Compute the mean for each row and column, respectively

```
apply(A, 1, mean)

## [1] 16 17 18 19

apply(A, 2, mean)

## [1] 11.5 15.5 19.5 23.5
```

Compute the standard deviation for each row and column, respectively

```
apply(A, 1, sd)

## [1] 5.163978 5.163978 5.163978 5.163978

apply(A, 2, sd)

## [1] 1.290994 1.290994 1.290994 1.290994
```

Compute the cumulative sums for the data values in each row

```
apply(A, 1, cumsum)

##      [,1] [,2] [,3] [,4]
## [1,]   10   11   12   13
## [2,]   24   26   28   30
## [3,]   42   45   48   51
## [4,]   64   68   72   76
```

The cumulative sums are computed by row, BUT the matrix is built column-wise (the default way in R)

Create a mtrix row-wise with byrow=TRUE

```
B <- matrix(apply(A, 1, cumsum), nrow = 4, byrow = TRUE)
B

##      [,1] [,2] [,3] [,4]
## [1,]   10   24   42   64
## [2,]   11   26   45   68
## [3,]   12   28   48   72
## [4,]   13   30   51   76
```

compute the cumulative sums for each column

```
apply(A, 2, cumsum)

##      [,1] [,2] [,3] [,4]
## [1,]   10   14   18   22
## [2,]   21   29   37   45
## [3,]   33   45   57   69
## [4,]   46   62   78   94
```