STAT 210
Applied Statistics and Data Analysis
Objects and Data II: Matrices and Arrays

Joaquín Ortega

joaquin.ortegasanchez@kaust.edu.sa

Matrices

Matrices

A matrix is a rectangular arrangement of cells, each of which contains a value.

To create a matrix in `R` you can use the `matrix` function, whose syntax is

```
matrix(data, nrow, ncol, byrow=F, dimnames = NULL)
```

where `nrow` and `ncol` represent, respectively, the number of rows and columns in the matrix. Only the first argument is indispensable.

Matrices

If neither the second nor the third argument appears, the data is placed in a one-dimensional matrix, i.e., a vector. If only one of the dimensions is included, the other is determined by division.

```
matrix(1:6)
```

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

```
matrix(1:6, nrow=3)
```

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

Matrices

Observe that the elements are stored column-wise in the matrix. If we want the elements to be stored row-wise, the option `byrow` has to be set to `TRUE`:

```
matrix(1:6, nrow=3, byrow=T)
```

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

The elements of a matrix must all be of the same type. To store columns of different types in a two-dimensional array, it is necessary to use a data frame, an object that we will study later on.

Matrices

To add and subtract matrices, operations that are carried out component by component, the only necessary condition is that both matrices have the same dimensions. The usual addition and subtraction symbols are used for these operations.

```r
(A <-  matrix(1:6, nrow=3, byrow=T))
```

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

```r
(B <-  matrix(seq(0, 10, 2), 3, 2))
```

```
##      [,1] [,2]
## [1,]    0    6
## [2,]    2    8
## [3,]    4   10
```

Matrices

```r
A+B
```

```
##      [,1] [,2]
## [1,]    1    8
## [2,]    5   12
## [3,]    9   16
```

```r
A-B
```

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

Matrices

You can also add, subtract, multiply, or divide by a scalar:

```
A-5
```

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

```
B/2
```

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

Matrices

Matrix multiplication is a more complicated operation. The notation is %*%.

The rule for multiplication of matrices is that the number of columns in the first matrix must be equal to the number of rows in the second.

```
(D <- matrix ((1:8)*3,2))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3    9   15   21
## [2,]    6   12   18   24
```

```
A %*% D
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   15   33   51   69
## [2,]   33   75  117  159
## [3,]   51  117  183  249
```

Matrices

It is also possible to multiply two matrices with the same dimensions component-wise, by using the usual symbol for multiplication $*$:

```
A*B
```

```
##      [,1] [,2]
## [1,]    0   12
## [2,]    6   32
## [3,]   20   60
```

Matrices

The following table presents other common operations with matrices

Name	Operation
`dim()`	matrix dimensions
`as.matrix()`	coerces the argument as a matrix
`t()`	transposition
`diag()`	extracts the diagonal elements
`det()`	determinant
`solve()`	inverse
`eigen()`	calculates eigenvalues and eigenvectores
`dimnames()`	retrieve or set the dimnames

Table 1. Matrix Functions.

Matrices

Let's see some examples:

```
(XX <- matrix(c(2,3,4,1,5,3),ncol=3))
```

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

```
t(XX)
```

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

Matrices

```
XX %*% t(XX)
```

```
##      [,1] [,2]
## [1,]   45   25
## [2,]   25   19
```

```
t(XX) %*% XX
```

```
##      [,1] [,2] [,3]
## [1,]   13   11   19
## [2,]   11   17   23
## [3,]   19   23   34
```

Matrices

The `diag` function extracts the diagonal elements of a matrix and can also be used to build up diagonal matrices:

```
diag(XX)
```

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

```
diag(t(XX))
```

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

```
diag(t(XX) %*% XX)
```

```
## [1] 13 17 34
```

Matrices

```r
diag(1:4)
```

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

```r
diag(3)
```

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

Matrices

We now explore the use of the functions `det` and `solve`:

```
(YY <- matrix(c(12,3,8,16,21,5,7,9,12,18,
                4,3,19,5,21,8), ncol=4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   12   21   12   19
## [2,]    3    5   18    5
## [3,]    8    7    4   21
## [4,]   16    9    3    8
```

```
det(YY)
```

```
## [1] -50828
```

Matrices

`solve(A)` gives the inverse of `A`. We round the output to three decimals using the function `round`:

```
round(solve(YY),3)
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,] -0.043  0.013  0.000  0.092
## [2,]  0.091 -0.042 -0.061 -0.030
## [3,] -0.015  0.066 -0.002  0.000
## [4,] -0.011 -0.004  0.068 -0.025
```

Matrices

To verify that this is the inverse, we multiply the matrices

```
YY%*%solve(YY)
```

```
##                   [,1]          [,2]         [,3]
## [1,]  1.000000e+00  4.163336e-17 2.220446e-16
## [2,]  6.938894e-18  1.000000e+00 1.110223e-16
## [3,]  0.000000e+00 -1.387779e-17 1.000000e+00
## [4,] -2.775558e-17  0.000000e+00 0.000000e+00
##                   [,4]
## [1,]  0.000000e+00
## [2,]  2.775558e-17
## [3,] -1.110223e-16
## [4,]  1.000000e+00
```

Matrices

We round off the values (to 15 decimal places) so that the result is clearer:

```
round(YY%*%solve(YY),15)
```

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

```
round(solve(YY)%*%YY,15)
```

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

Matrices

The same instruction `solve`, used with a matrix `A` and a column vector `b` solves the linear equation `b=Ax` to get `x`:

```
b <- 1:4
(x <- solve(YY,b))
```

```
## [1]  0.35185331 -0.29416857  0.11033289
## [4]  0.08585819
```

```
YY %*% x
```

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

Matrices

Let's use the `B` matrix for some examples on the `dimnames` function

```
dimnames(B)
```

```
## NULL
```

```
dim(B)
```

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

```
subjects <- c('Patient1','Patient2','Patient3')
variables <- c('Var1', 'Var2')
dimnames(B) <- list(subjects,variables)
```

Matrices

```r
dimnames(B)
```

```
## [[1]]
## [1] "Patient1" "Patient2" "Patient3"
##
## [[2]]
## [1] "Var1" "Var2"
```

```r
B
```

```
##          Var1 Var2
## Patient1    0    6
## Patient2    2    8
## Patient3    4   10
```

Matrices

To extract data from a matrix one can use the coordinates of the entry or entries one requires:

```
A[1,2]
```

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

```
A[1,]
```

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

```
YY[2:3,1:2]
```

```
##      [,1] [,2]
## [1,]    3    5
## [2,]    8    7
```

Matrices

and it is also possible to use dimension names, if available:

```
B['Patient1','Var2']
```

```
## [1] 6
```

```
B['Patient1',]
```

```
## Var1 Var2
##    0    6
```

```
B[,'Var2']
```

```
## Patient1 Patient2 Patient3
##        6        8       10
```

Matrices

Finally, to add new rows or columns to a matrix we can use the functions `rbind` and `cbind`, respectively:

```
A
```

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

```
(A <-cbind(A,c(3,5,7)))
```

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

Matrices

```r
B
```

```
##           Var1 Var2
## Patient1    0    6
## Patient2    2    8
## Patient3    4   10
```

```r
cbind(B,c(3,5,7))
```

```
##           Var1 Var2  
## Patient1    0    6 3
## Patient2    2    8 5
## Patient3    4   10 7
```

Matrices

```r
cbind(B,Var3=c(3,5,7))
```

```
##          Var1 Var2 Var3
## Patient1    0    6    3
## Patient2    2    8    5
## Patient3    4   10    7
```

```r
rbind(B,Patient4 = c(6,12))
```

```
##          Var1 Var2
## Patient1    0    6
## Patient2    2    8
## Patient3    4   10
## Patient4    6   12
```

Arrays

Arrays

An array is an extension of a matrix to more dimensions. Arrays follow the same rules as matrices. To define an array use `array(data, dim)`:

```
(x <- array (1:24, c(3,4,2)))
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

Arrays

When printing the array, R starts with the highest dimension and goes down to the lowest dimension, printing two-dimensional matrices at each stage.

Another way of creating an array is to start with a vector and then assign the dimensions

```
(x1 <- 1:24)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
## [16] 16 17 18 19 20 21 22 23 24
```

```
dim(x1) <- c(4,3,2)
```

Arrays

```
x1
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   13   17   21
## [2,]   14   18   22
## [3,]   15   19   23
## [4,]   16   20   24
```

Arrays

Indexing follows the same rules as before:

```
x[,2,]
```

```
##      [,1] [,2]
## [1,]    4   16
## [2,]    5   17
## [3,]    6   18
```

```
x[,3,1]
```

```
## [1] 7 8 9
```

Arrays

```r
x[,,1]
```

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

Arrays: `aperm`

The function `aperm` is an extension of matrix transposition that changes the dimensions of an array. The syntax is

```
aperm(array, perm, resize=TRUE)
```

where `array` is the arrangement to permute, `perm` is the permutation vector and `resize` is an indicator of whether the data vector should be resized, if necessary.

Arrays: aperm

As an example we will permute the iris3 array that has dimensions $50 \times 4 \times 3$ to an array with dimensions $4 \times 3 \times 50$:

```
str(iris3)
```

```
##  num [1:50, 1:4, 1:3] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : NULL
##   ..$ : chr [1:4] "Sepal L." "Sepal W." "Petal L." "Petal W."
##   ..$ : chr [1:3] "Setosa" "Versicolor" "Virginica"
```

```
iris3b <- aperm(iris3, c(2,3,1))
str(iris3b)
```

```
##  num [1:4, 1:3, 1:50] 5.1 3.5 1.4 0.2 7 3.2 4.7 1.4 6.3 3.3 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:4] "Sepal L." "Sepal W." "Petal L." "Petal W."
##   ..$ : chr [1:3] "Setosa" "Versicolor" "Virginica"
##   ..$ : NULL
```