Introductionto R: Matrices

In R, a matrix is a 2 dimensional data structure that contains atomic elements of the same type. Just like a vector, a matrix can hold numeric values, logical values, characters or other atomic data types. A matrix consists 2 or more rows and 2 or more columns. When talking about matrices, the letter m refers to number of rows and n refer to the number of columns.

Matrix Construction

To construct a matrix in R, you can use the matrix() function which takes several arguments:

```
X <- matrix(data = c(1,2,3,4,5,6),  # a vector used to construct the matrix
            nrow = 2,               # number of rows in the matrix
            ncol = 3,               # number of columns in the matrix
            byrow = FALSE)          # fill the matrix by rows or columns?

print(X)
```

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

Matrices are typically assigned to capital lettered variables to distinguish them from vectors. You can add to a matrix you've already constructed with the rbind() and cbind() functions. rbind takes a sequence of vectors, matrices or data frames (we'll cover those later) as arguments and combines them by rows, while cbind() combines sequences of data objects by columns:

```
Y <- matrix( seq(10,15,1), 3, 2)  # Create a new 3x2 matrix
print (Y)
```

```
##      [,1] [,2]
## [1,]   10   13
## [2,]   11   14
## [3,]   12   15
```

```
X <- rbind(X, c(7,8,9) )  # Adds the vector c(7,8,9) as a new row
print(X)
```

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

```
XY <- cbind(X,  Y)  # Add the new matrix to X by column
print(XY)
```

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

*Note: the arguments to rbind() should have the same number of columns and the arguments to cbind() should have the same number of rows. Since cbind() and rbind() work on vectors, you can use them to construct matrices from vectors on a row by row, or column by column basis:

```
Z <- rbind(c(1,2,3), c(4,5,6), c(7,8,9))
print(Z)
```

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

You can turn a matrix's rows into columns and columns into rows using the transpose function t():

```
X <- t(X)
print(X)
```

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

Transpose essentially flips a matrix along the main diagonal. You can also convert a matrix into a vector using the c() function:

```
c(X)  # Convert a matrix into a vector by column
```

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

```
# If you want to convert a matrix to vector by row, take the transpose first:
c(t(X))
```

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

Matrix Indexing

Similar to vectors, you can access the elements inside a matrix with indexing. You may have noticed that when we printed matrices above, the rows and columns were labeled with values in square brackets. Those labels are the index values of the rows and columns. Since matrices have two dimensions, they have two indexes, a row index and a column index, which are separated by a comma. You can use indices to grab specific values, rows or columns in a matrix:

```
X[3,2]   # Get the value at row 3 column 2
```

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

```
# Leave the row or column index blank to take the entire row or column:

X[3, ]   # Get row 3
```

```
## [1] 5 6 9
```

```
X[ ,2]   # Get column 2
```

```
## [1] 2 4 6
```

You can also take slices of rows and columns, just like you can with vectors:

```
print(X[1:2, 2:3])  # Get data points in rows 4 and 5 and columns 2 and 3
```

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

All the vector indexing operations discussed last time work for matrices:

```
print( X[c(1,2,3), c(1,3)] ) # Vector indexing
```

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

```
print( X[ -2, -2] ) # Remove row 2 and remove column 2
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    5    9
```

```
X_logical <- (X %% 2 == 0)   # Create a logical matrix identifying even numbers
print(X_logical)
```

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

```
X[X_logical]      # Use the logical matrix as an index to get even values in X
```

```
## [1] 2 4 6 8
```

```
X[X %in% c(2,6,8,9,10,15,100)]   # Get matrix values contained in a vector
```

```
## [1] 2 6 8 9
```

A matrix can also have named dimensions. You can assign dimension names when creating a matrix by passing a list of two vectors the dimnames argument:

```
Z <- matrix(c(1,2,3,4), 2, 2, dimnames = list( c("r1","r2"), c("c1","c2")) )
print(Z)
```

```
##    c1 c2
## r1  1  3
## r2  2  4
```

You can also create or reassign dimnames after creating a matrix:

```
dimnames(Z) <- list( c("first_r","second_r"), c("first_c","second_c"))

print(Z)
```

```
##          first_c second_c
## first_r        1        3
## second_r       2        4
```

When dimensions are named, you can index the matrix using the dimension names or the normal numeric indexes:

```
Z[2,2]                     # Get the value at 2,2 with numeric indexing
```

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

```
Z["second_r","second_c"]  # Get the value at 2,2 with named dimensions
```

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

Matrix Operations

Matrices in R offer many of the same conveniences as vectors. For instance, you can perform element-wise math operations on matrices of same dimensions by using the standard math symbols:

```
X <- Y <- matrix(c(1,2,-1,1,1,2,1,2,3),3,3)  #Make two new identical matrices
print(X)
```

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

```
print(Y)
```

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

```
# You can use the dim() function to check matrix dimensions:
dim(X)
```

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

```
dim(Y)
```

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

```
print(X + Y) # Element-wise addition
```

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

```
print(X * Y) # Element-wise multiplication
```

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

```
print(X / Y) # Element-wise division
```

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

For true matrix multiplication, use the %*% operator:

```
print( X %*% Y )
```

```
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    2    7   10
## [3,]    0    7   12
```

Here are a few other useful matrix operations:

```r
diag(X)  # Get elements on the main diagonal of a matrix
```

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

```r
solve(X)  # Get the inverse of a square matrix
```

```
##       [,1]  [,2]  [,3]
## [1,]  0.25  0.25 -0.25
## [2,]  2.00 -1.00  0.00
## [3,] -1.25  0.75  0.25
```

```r
eigen(X) # Get the eigenvectors and eigenvalues of a matrix
```

```
## eigen() decomposition
## $values
## [1]  4.3234043  1.3579264 -0.6813306
## 
## $vectors
##           [,1]       [,2]       [,3]
## [1,] 0.3914710  0.5815460 -0.1960489
## [2,] 0.6358665  0.6698230  0.8383244
## [3,] 0.6651498 -0.4616724 -0.5087013
```

```r
rowSums(X)    # Get the sums of the rows
```

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

```r
colSums(X)    # Get the sums of the columns
```

```
## [1] 2 4 6
```

```r
rowMeans(X)   # Get the means of the rows
```

```
## [1] 1.000000 1.666667 1.333333
```

```r
colMeans(X)   #Get the means of the columns
```

```
## [1] 0.6666667 1.3333333 2.0000000
```

```r
sum(X)  # Sum all the values in X
```

```
## [1] 12
```

```r
min(X)  # Get the min value in X
```

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

```r
max(X)  # Get the max value in X
```

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

```r
mean(X) # Get the mean of all the values in X
```

```
## [1] 1.333333
```

*Note: sum, min max and mean also work on vectors

Any time you want to perform a matrix operation, there's a good chance R has a function that does just what you need built in or available in a package. Google is your friend. R also contains an array data structure that stores elements of the same atomic type in an arbitrary number of dimensions. An array is just a vector stored with an extra attribute "dim" that specifies its dimensions:

```r
A <- array(1:64, dim = c(4,4,4))     #create a 4 x 4 x 4 array

dim(A)     #Check the dimensions of the array
```

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

```r
print(A)
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   17   21   25   29
## [2,]   18   22   26   30
## [3,]   19   23   27   31
## [4,]   20   24   28   32
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]   33   37   41   45
## [2,]   34   38   42   46
## [3,]   35   39   43   47
## [4,]   36   40   44   48
##
## , , 4
##
##      [,1] [,2] [,3] [,4]
## [1,]   49   53   57   61
## [2,]   50   54   58   62
## [3,]   51   55   59   63
## [4,]   52   56   60   64
```

Arrays generally support the same types of indexing and vector operations as vectors and matrices, just with different numbers of dimensions. Arrays are uncommon so we won't study them any further. Up till now, all the data structures we've covered hold values of the same atomic type. Real world data comes in all shapes and forms so the data structures we know aren't sufficient to tackle most real data sets.