4. Array and Matrix

Array

An array generalizes a vector to two or more dimensions: it’s a multi-dimensional collection of elements of the same type. Create an array with

\[
a = \text{array(data=NA, dim=length(data), dimnames=NULL)}
\]

where \(\text{dim}\) is a vector giving the largest index in each dimension and \(\text{dimnames}\) is a list of \(\text{length(dim)}\) vectors containing names (or NULL) for each dimension. The array’s values are filled from the data vector in “column major” order, in which the first subscript moves the fastest and the last subscript moves the slowest.

Access to dimensions and their names are via \(\text{dim(a)}\) and \(\text{dimnames(a)}\). e.g.

\[
a = \text{array(data=-1:24, dim=c(3,4,2))}
\]
\[
\text{dimnames(a)} = \text{list(c("slow", "medium", "fast"), c("cold", "tepido", "warm", "hot"),}
\]
\[
c("Monday", "Tuesday")}
\]

The elements in an array are stored in a vector, which allows changing dimensions! e.g.

\[
\text{dim(a)} = \text{c(4, 6)} \quad \# \text{4 by 6}
\]
\[
\text{dim(a)} = \text{NULL} \quad \# \text{vector}
\]
\[
\text{dim(a)} = \text{c(2, 3, 2, 2)} \quad \# \text{2 by 3 by 2 by 2}
\]
\[
\text{dim(a)} = \text{c(3, 4, 2)} \quad \# \text{back to start}
\]

Indexing

- Access a single element from the array perspective by giving indices for all dimensions, separated by commas, in square brackets. e.g. \(a[2, 3, 1]\)
- Access a single element from the vector perspective by using a single index. e.g. \(a[8]\)
- Access a regular subset of an array by giving a vector of values for each index (or nothing to get all the values for that index). e.g. \(a[1, 3:4, 2]\)
- Access an irregular subset of an array with an index array having \(\text{length(dim(a))}\) columns and one row for each desired value. e.g. To get values in positions \((1,1,1)\) and \((2,2,2)\), use

\[
\text{index} = \text{matrix(data=c(1,1,1, 2,2,2), nrow=2, ncol=\text{length(dim(a))}, byrow=TRUE)}
\]
\[
a[\text{index}]
\]
- Access an irregular subset of an array satisfying a logical condition: \(a[\text{logical.condition}]\) is a vector of values in \(a\) corresponding to TRUE values in the array \(\text{logical.condition}\). e.g.

\[
(a \text{ %% 2}) == 0 \quad \# \text{Which values are even?}
\]
\[
a[(a \text{ %% 2}) == 0] \quad \# \text{Get even values.}
\]
\[
a[(a \text{ %% 2}) == 0] = -a[(a \text{ %% 2}) == 0] \quad \# \text{Set even values: multiply by -1.}
\]
Matrix

A matrix is a two-dimensional array.

Create a matrix from vector data with `matrix(data=NA, nrow=1, ncol=1, byrow=FALSE, dimnames=NULL)`, where `byrow` tells whether to fill the matrix from data by row (or by column, the default), and `dimnames` is NULL or a list of two vectors containing row and column names. e.g.

```r
m = matrix(data=1:12, nrow=3, ncol=4, byrow=TRUE)
kids = matrix(data=c(c(1,2,6,7,9,11), c(1,5,100,100,100,100)),
               nrow=2, ncol=6, byrow=TRUE,
               dimnames=list(c("Age","#Toys"),
                             c("Teresa","Margaret","Monica","Andrew","Mary","Philip")))
```

Two other ways to create matrices are by combining columns with `cbind(...) or rbind(...)`, getting data from vector, matrix, or data frame arguments in . . .:

- `cbind(...) combines columns into a matrix; e.g. cbind(m, 101:103)`
- `rbind(...) combines rows into a matrix; e.g. rbind(m, 101:104)`

For a matrix `m` (this paragraph helps with Connect Four),

- `row(m)` is a matrix of row numbers of elements of `m` (depends on `m`’s dimensions, not data)
- `col(m)` is a matrix of column numbers of elements of `m`; e.g.

```r
row(m) == col(m)
m[row(m) == col(m)] # main diagonal
r = 2; c = 3
m[row(m) - col(m) == r - c] # diagonal through (r, c)
m[ ______________________ ] # reverse diagonal through (r, c)
```

For matrices `A` and `B` and vectors `b` and `x`,

- `A * B` is an element-wise product
- `A %*% B` is the usual matrix product, and `A %*% x` is the usual matrix-vector product
- `solve(a=A, b=b)` gives the solution `x` to the system of linear equations, `A\vec{x} = \vec{b}`; e.g.

```r
A = matrix(data=1:4, nrow=2, ncol=2)
b = c(7, 10)
(x = solve(a=A, b=b))
A %*% x # check: is it b?
```

The Matrix package has more: http://cran.r-project.org/web/packages/Matrix/Matrix.pdf.