5. 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 = array(data=NA, dim=length(data), dimnames=NULL)` where `dim` is a vector giving the largest index in each dimension and `dimnames` is a list of `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 `dim(a)` and `dimnames(a)`. e.g.

```
a = array(data=-(1:24), dim=c(3,4,2))
dimnames(a) = list(c("slow", "medium", "fast"), c("cold", "tepid", "warm", "hot"),
                   c("Monday", "Tuedsay"))
```

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

```
dim(a) = c(4, 6)       # 4 by 6
dim(a) = NULL          # vector
dim(a) = c(2, 3, 2, 2) # 2 by 3 by 2 by 2
dim(a) = c(3, 4, 2)    # 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[ , 3:4, 2]`
- Access an irregular subset of an array with an *index array* having `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

  ```
  index = matrix(data=c(1,1,1, 2,2,2), nrow=2, ncol=length(dim(a)), byrow=TRUE)
  a[index]
  ```

- Access an irregular subset of an array satisfying a logical condition: `a[logical.condition]` is a vector of values in `a` corresponding to `TRUE` values in the array `logical.condition`. e.g.

  ```
  (a %% 2) == 0                       # Which values are even?
  a[(a %% 2) == 0]                    # Get even values.
  a[(a %% 2) == 0] = -a[(a %% 2) == 0] # 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.

```
m = matrix(data=1:12, nrow=3, ncol=4, byrow=TRUE)
kids = matrix(data=c(c(1,4,6,9,11,13,15), c(5,10,20,30,30,30,30)),
              nrow=2, ncol=7, byrow=TRUE,
              dimnames=list(c("Age","#Toys"),
                            c("Anna","Teresa","Margaret","Monica","Andrew","Mary","Philip")
                           )
             )
```

Two other ways to create matrices are by combining columns with `cbind(...)` or rows with `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.

```
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[row(m) + col(m) == r + c] # 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 $\vec{x}$ to the system of linear equations, $A\vec{x} = \vec{b}$; e.g.

```
A = matrix(data=1:4, nrow=2, ncol=2)
b = c(7, 10)
(x = solve(a=A, b=b)) # solve system 1x + 3y = 7 and 2x + 4y = 10
A %*% x # check: is it b?
```

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