Vector, Matrices and Arrays

In this part of the class, the student is introduced to the notion of data objects. The basic kind of data object in `Splus` are the following:

- vector
- matrix
- array
- list
- factor
- time series
- data frame.

We will only be concerned with the first 5 types and we will leave the last two types for further discussion in the subsequent.

Vectors

The simplest type of data object is a vector. A vector is simply an ordered set of values. This is turn implies that there exists a convenient way to extract some parts of the vector.

The most obvious way to assign a vector is with the `c` command. For example

```
> x <- c(1,3,4,5)
> x
[1] 1 3 4 5
> length(x)
[1] 4
> mode(x)
[1] "numeric"
> names(x)
NULL
> y <- c( c(2,3), c(1,-6))
```

```
> y
[1]  2  3  1 -6
```

The `rep` function might be used instead if there is need to repeat values by specifying either a `times` or a `length` argument.

```
> rep(NA,6)
[1] NA NA NA NA NA NA
> rep(x, 3)
 [1] 1 3 4 5 1 3 4 5 1 3 4 5
> rep(x, c(1,2,2,3))
[1] 1 3 3 4 4 5 5 5
```

The last example shows that when `times` is a vector *with the same length* as of the vector of values being repeated, each value is repeated the corresponding number of times.

The sequence operator generates sequence of numeric values one unit apart.

```
> 1:13
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13
> -3:6
 [1] -3 -2 -1  0  1  2  3  4  5  6
> 1.1:5
[1] 1.1 2.1 3.1 4.1
> 4:-5
 [1]  4  3  2  1  0 -1 -2 -3 -4 -5
```

More generally, the `seq` function generates sequences of numeric values with an arbitrary increment. Look how it can be used:

```
 seq(-1,2, 0.5)
[1] -1.0 -0.5  0.0  0.5  1.0  1.5  2.0
> seq(-1,2, length=12)
 [1] -1.00000000 -0.72727273 -0.45454545 -0.18181818  0.09090909  0.36363636
 [7]  0.63636364  0.90909091  1.18181818  1.45454545  1.72727273  2.00000000
> seq(1, by=0.5, length=12)
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5
```

Matrices

Matrices are used to arrange values by rows and columns in a rectangular table. For data analysis, different variables are usually represented by different columns and different cases or subjects are represented by different rows. Matrices differ from vectors in the sense that the `dim` function applies to them.

To create a matrix from an existing vector, use the `dim` function by assigning a vector of two integers specifying the number of rows and columns.

```
 matr <- rep(1:4, rep(3,4))
> matr
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
> dim(matr) <- c(3,4)
> matr
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    1    2    3    4
[3,]    1    2    3    4
 matr2 <- seq(-2,2,length=25)
> matr2
 [1] -2.0000000 -1.8333333 -1.6666667 -1.5000000 -1.3333333 -1.1666667 -1.0000000
 [8] -0.8333333 -0.6666667 -0.5000000 -0.3333333 -0.1666667  0.0000000  0.1666667
[15]  0.3333333  0.5000000  0.6666667  0.8333333  1.0000000  1.1666667  1.3333333
[22]  1.5000000  1.6666667  1.8333333  2.0000000
> dim(matr2) <- c(5,5)
> matr2
          [,1]       [,2]       [,3]      [,4]     [,5]
[1,] -2.000000 -1.1666667 -0.3333333 0.5000000 1.333333
[2,] -1.833333 -1.0000000 -0.1666667 0.6666667 1.500000
[3,] -1.666667 -0.8333333  0.0000000 0.8333333 1.666667
[4,] -1.500000 -0.6666667  0.1666667 1.0000000 1.833333
[5,] -1.333333 -0.5000000  0.3333333 1.1666667 2.000000
```

Frequently, we need to combine several vectors or matrices into a single matrix. The functions `rbind` and `cbind` are very convenient for handling such cases.

```
> matr3 <- rbind(c(1,2,-1), c(-3,1,5))
> matr3
     [,1] [,2] [,3]
[1,]    1    2   -1
[2,]   -3    1    5
> matr4 <- cbind(c(1,2,-1), c(-3,1,5))
> matr4
     [,1] [,2]
[1,]    1   -3
[2,]    2    1
[3,]   -1    5
> matr5 <- cbind(c(1,2,-1), c(-3,3,2,0))
Warning messages:
  Number of rows of result is not a multiple of
  vector length (arg 1) in: cbind(c(1, 2,-1), c(-3, 3, 2, 0))
> matr5
     [,1] [,2]
[1,]    1   -3
[2,]    2    3
[3,]   -1    2
```

```
[4,]    1    0
```

When vectors of different lengths are combined using `cbind` or `rbind` , the shorter ones are replicated cyclically so that the matrix is filled in.

```
matr6 <- cbind(matr, matr4)
> matr6
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    1   -3
[2,]    1    2    3    4    2    1
[3,]    1    2    3    4   -1    5
> matr6 <- cbind(matr, matr3)
Error in cbind(matr, matr3): Number of rows of matrices and
lengths of names vectors must match (see arg 2)
```

An alternative way to create a matrix is by the `matrix` function which takes the arguments `nrow` and `ncol`.

```
> matr7 <- matrix(1:28, nrow=7, ncol=4)
> matr7
     [,1] [,2] [,3] [,4]
[1,]    1    8   15   22
[2,]    2    9   16   23
[3,]    3   10   17   24
[4,]    4   11   18   25
[5,]    5   12   19   26
[6,]    6   13   20   27
[7,]    7   14   21   28
> matr8 <- matrix(-5:6, ncol=3, byrow=T)
> matr8
     [,1] [,2] [,3]
[1,]   -5   -4   -3
[2,]   -2   -1    0
[3,]    1    2    3
[4,]    4    5    6
>  matrix(1:23, nrow=7, ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    1    8   15   22
[2,]    2    9   16   23
[3,]    3   10   17    1
[4,]    4   11   18    2
[5,]    5   12   19    3
[6,]    6   13   20    4
[7,]    7   14   21    5
Warning messages:
  Replacement length not a multiple of number of
  elements to replace in: data[1:ll] <-old
```

```
>  matrix(1:23, nrow=7)
     [,1] [,2] [,3] [,4]
[1,]    1    8   15   22
[2,]    2    9   16   23
[3,]    3   10   17    1
[4,]    4   11   18    2
[5,]    5   12   19    3
[6,]    6   13   20    4
[7,]    7   14   21    5
Warning messages:
Replacement length not a multiple of number of
elements to replace in: data[1:ll] <-old
```

The argument `byrow` is very useful when reading data which has been stored in a text file.

```
> matr8
     [,1] [,2] [,3]
[1,]   -5   -4   -3
[2,]   -2   -1    0
[3,]    1    2    3
[4,]    4    5    6
> length(matr8)
[1] 12
> dim(matr8)
[1] 4 3
> mode(matr8)
[1] "numeric"
> dimnames(matr8)
NULL
> dimnames(matr8) <- list(c("A","B","C","D"), c("K1","K2","K3"))
> matr8
  K1 K2 K3
A -5 -4 -3
B -2 -1  0
C  1  2  3
D  4  5  6
```

Arrays

Arrays generalize matrices by extending the function `dim` to more than two dimensions. For example, if the rows and columns of a matrix are the length and the width of a rectangular arrangement of values of equal–sized cube, then length, width and height represent the dimensions of three way array. There is no limit to the number of dimensions of an array.

```
> arr1  <- array( c(2:9,12:19,112:119), dim=c(2,4,3))
> arr1
, , 1
     [,1] [,2] [,3] [,4]
[1,]    2    4    6    8
[2,]    3    5    7    9

, , 2
     [,1] [,2] [,3] [,4]
[1,]   12   14   16   18
[2,]   13   15   17   19

, , 3
     [,1] [,2] [,3] [,4]
[1,]  112  114  116  118
[2,]  113  115  117  119
```

The first dimension (the rows) is incremented first. This is the same as placing the values column by column. The second dimension (column) is incremented second. The third dimension is incremented by filling a matrix for each level of the third dimension.

```
> length(arr1)
[1] 24
> mode(arr1)
[1] "numeric"
> dim(arr1)
[1] 2 4 3
> dimnames(arr1)
NULL
```