R Data Types and Manipulation

140.776 Statistical Computing

August 21, 2011
R operates on *objects*:  
- vectors  
- matrices  
- factors  
- lists  
- data frames  
- functions
> sum(log(x[,2]-x[,1]))
Warning message:
In log(x[, 2] - x[, 1]) : NaNs produced

Which lines produced NaN?
Exercise

26 105 132 225 386 413 504 531 690 757 862 895 905
Logical vectors

- A logical vector can have values TRUE, FALSE and NA (NA will be discussed later)
- They are usually generated by conditions, e.g. comparisons involving <, <=, >, >=, ==, !=
- One can perform logical operations on them, e.g. & (and), | (or), ! (negation)

Example:

```r
> x<-c(1,2,3)
> x
[1] 1 2 3

> y<-x>2
> y
[1] FALSE FALSE TRUE

> !y
[1] TRUE TRUE FALSE
```
Missing values

- NA: an element or value is not available.
- Any operation on NA becomes NA.
- `is.na()` is used to test objects if they are NA
- `is.na(x)` is different from `x==NA`

Example:

```r
> z<-c(1:3,NA)
> z
[1] 1 2 3 NA

> is.na(z)
[1] FALSE FALSE FALSE TRUE

> z==NA
[1] NA NA NA NA

> z==NA
```

Example:

```r
> z<-c(1:3,NA)
> z
[1] 1 2 3 NA

> is.na(z)
[1] FALSE FALSE FALSE TRUE

> z==NA
```
NA: an element or value is not available.

Any operation on NA becomes NA.

is.na() is used to test objects if they are NA

is.na(x) is different from x==NA

Example:

> z<-c(1:3,NA)
> z
[1] 1 2 3 NA

> is.na(z)
[1] FALSE FALSE FALSE TRUE

> z==NA
[1] NA NA NA NA

[1] NA NA NA NA
Missing values

- NaN (not a number) is also a missing value.
- A NaN value is NA, but the converse is not true.
- Use is.nan() to test for NaN

Example:

```r
> z <- c(1:3, NA, 0/0)
> z
[1] 1 2 3 NA NaN

> is.na(z)
[1] FALSE FALSE FALSE TRUE TRUE

> is.nan(z)
[1] FALSE FALSE FALSE FALSE TRUE
```
Exercise

```r
> z <- log(x[,2] - x[,1])
> id <- is.nan(z)
> (1:1000)[id]
[1] 26 105 132 225 386 413 504 531 690 757 862 895 905
```
Character vectors

Example:

```r
> x<-c("a","b")
> x
[1] "a" "b"

> x<-c("apple","orange")
> x
[1] "apple" "orange"

> paste(c("apple","orange"),1:4)
[1] "apple 1" "orange 2" "apple 3" "orange 4"

> paste(c("apple","orange"),1:4,sep="")
[1] "apple1" "orange2" "apple3" "orange4"
```
Creating vectors

In general, vectors can be created using `c()` or `vector()`:

```r
> x <- c(1+0i, 2+4i)
> x
[1] 1+0i 2+4i

> x <- vector(mode="numeric", length=5)
> x
[1] 0 0 0 0 0
```
When different objects are mixed in a vector, *coercion* occurs so that every element in the vector is of the same class:

```r
> c(1,"a") # character
[1] "1" "a"

> c(TRUE,2) # numeric
[1] 1 2

> c("a",TRUE) # character
[1] "a" "TRUE"
```
Explicit coercion

Objects can be explicitly coerced from one class to another using the as.* functions:

```r
> x<-1:5
> class(x)
[1] "integer"

> as.numeric(x)
[1] 1 2 3 4 5

> as.logical(x)
[1] TRUE TRUE TRUE TRUE TRUE

> as.character(x)
[1] "1" "2" "3" "4" "5"

> as.complex(x)
[1] 1+0i 2+0i 3+0i 4+0i 5+0i
```
Explicit coercion

Nonsensical coercion results in NAs:

```r
> x<-c("a","b","c")
> as.numeric(x)
[1] NA NA NA
Warning message:
NAs introduced by coercion

> as.logical(x)
[1] NA NA NA NA
```
Indexing and subsetting

Subsets of the elements of a vector may be selected by using one of the following index vectors:

- logical vector
- vector of positive integral quantities
- vector of negative integral quantities
- vector of character strings

Examples:

```r
> x<-c(-2:1,NA,3)
> x
[1]  -2 -1  0  1  NA  3

> x[!is.na(x)]
[1]  -2 -1  0  1  3

> (x+1)[!is.na(x) & x>0]
[1]  2  4
```
Examples (cont):

> x
[1] -2 -1 0 1 NA 3
> x[2:3]
[1] -1 0
Indexing and subsetting

Examples (cont):

```r
> x
[1] -2 -1  0  1  NA  3

> x[rep(c(2,4),2)]
```
Examples (cont):

> x
[1]  -2  -1   0   1  NA   3

> x[rep(c(2,4),2)]
[1] -1  1 -1  1
Examples (cont):

```r
> x
[1]  -2  -1   0   1   NA  3

> x[-(2:3)]
```
Examples (cont):

> x
[1]  -2 -1  0  1  NA  3

> x[-(2:3)]
[1]  -2  1  NA  3
Indexing and subsetting

Examples (cont):

```r
> x
[1] -2 -1  0  1 NA  3

> names(x)<-c("A","B","C","D","E","F")
> x
  A B C D E F
-2 -1  0  1 NA  3

> x[c("B","F")]
  B F
-1  3
```
> x<-rnorm(100)
> y<-1:10
> ls()

> save.image("test.rda")
> rm(list=ls(all=TRUE))
> ls()
> load("test.rda")
> ls()

> save(x,y,file="test2.rda")
> ?save
> load("ex1.rda")
> ls()
Load workspace

> z <- array(x, dim=c(6,6))
R objects can have attributes:

- mode (intrinsic attribute)
- length (intrinsic attribute)
- names, dimnames
- dimensions (e.g. matrices, arrays)
- class
- other user-defined attributes

Mode and length of an object can be accessed using `mode()` and `length()`.
Other attributes of an object can be accessed using `attributes()`.
Example:

```r
> x
A B C D E F
-2 -1 0 1 NA 3

> mode(x)
[1] "numeric"

> length(x)
[1] 6

> attributes(x)
$names
[1] "A" "B" "C" "D" "E" "F"
```
Arrays are vectors with a *dimension* attribute.

The dimension attribute is a vector of non-negative integers.

If the length of the dimension vector is \(k\), then the array is \(k\)-dimensional.

Dimension is accessed through the *dim* attribute.

Example:

```r
> m <- array(dim = c(2, 3))
> m
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```
Matrices are 2-dimensional arrays:

```
> m<-array(dim=c(2,3))
> m
[,1] [,2] [,3]
[1,]  NA  NA  NA
[2,]  NA  NA  NA

> n<-matrix(nrow=2,ncol=3)
> n
[,1] [,2] [,3]
[1,]  NA  NA  NA
[2,]  NA  NA  NA
```
Creating and indexing arrays and matrices

Arrays and matrices can be created from vectors by adding a dimension attribute:

```r
> x <- array(1:6, dim = c(2, 3))

# Should it be
> x
 [,1] [,2] [,3]
[1,] 1  3  5
[2,] 2  4  6

# Or
> x
 [,1] [,2] [,3]
[1,] 1  2  3
[2,] 4  5  6
```
Values are assigned in “column major order”, i.e., the first subscript moves fastest and the last slowest.

```r
> x<-array(1:6,dim=c(2,3))
> x
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

> x<-matrix(1:6,nrow=2,ncol=3)

> x<-matrix(1:6,nrow=2,ncol=3,byrow=TRUE)
```
Creating and indexing arrays and matrices

Elements can be indexed by subscripts in square brackets.

```r
> x <- array(1:6, dim = c(2, 3))
> x
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

> x[2, 1]
[1] 2

> x[1, 2:3]
[1] 3 5

> x[1,]
[1] 1 3 5
```
Creating and indexing arrays and matrices

Elements in a matrix can also be accessed using an index matrix.

```r
> x <- matrix(1:6, nrow=2, ncol=3)
> x
 [,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6
> xindex <- array(c(1:2, 2:1), dim=c(2,2))
> xindex
 [,1] [,2]
[1,]  1  2
[2,]  2  1
> x[xindex]
[1] 3 2
> x[xindex] <- 0
> x
 [,1] [,2] [,3]
[1,]  1  0  5
[2,]  0  4  6
```
Creating and indexing arrays and matrices

Arrays with more than 2 dimensions:

```r
> y<-1:12
> class(y)
[1] "integer"
> y
[1]  1  2  3  4  5  6  7  8  9 10 11 12
> dim(y)<-c(2,3,2)
```

(Do not use computer): \( y[2,1,2] = ? \)
Arrays with more than 2 dimensions:

```r
> class(y)
[1] "array"

> y
,, 1
 [,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6
,, 2
 [,1] [,2] [,3]
[1,]  7  9 11
[2,]  8 10 12
```
Matrices can be created by `cbind()` (column-binding) or `rbind()` (row-binding):

```r
> x<-1:3
> y<-matrix(4:9,nrow=3,ncol=2)
> cbind(x,y)
  [,1] [,2] [,3]
[1,]  1  4  7
[2,]  2  5  8
[3,]  3  6  9

> z<-4:6
> rbind(x,z)
   [,1] [,2] [,3]
  x   1   2   3
  z   4   5   6
```