R Data Types and Manipulation

140.776 Statistical Computing

August 21, 2011
R operates on *objects*:  
- vectors  
- matrices  
- factors  
- lists  
- data frames  
- functions
Arithmetic expressions

> x
> [,1] [,2] [,3]
> [1,]  1  3  5
> [2,]  2  4  6

> sum(x)

> rowSums(x)

> colSums(x)
Use what you have learnt so far:

```r
> load("ex1.rda")
> x
> y
> z<-matrix(x,6,6)
```

\[ z_{i1} \times y_1 + z_{i2} \times y_2 + \ldots + z_{i6} \times y_6 = ? \]
Exercise

```r
> w <- matrix(y, nrow=6, ncol=6, byrow=TRUE)
> rowSums(w*z)
[1]  -0.1160327 -0.2419110  0.2789480
    -0.3061841 -0.1621261  1.1042598
```
For matrix multiplication, you have to use %*%:

```r
> x
 [,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6
> y
 [,1] [,2] [,3]
[1,]  7  9 11
[2,]  8 10 12
> x%*%y
Error in x %*% y : non-conformable arguments
> x%*%t(y)  ## t() obtains transpose of a matrix
 [,1] [,2]
[1,] 62  71
[2,] 80  92
```
Use what you have learnt so far:

```r
> z%*%y
[1,] -0.1160327 -0.2419110  0.2789480 -0.3061841 -0.1621261  1.1042598
[2,] -0.2419110  0.2789480 -0.3061841 -0.1621261  1.1042598
[3,]  0.2789480 -0.3061841 -0.1621261  1.1042598
[4,] -0.3061841 -0.1621261  1.1042598
[5,] -0.1621261  1.1042598
[6,]  1.1042598
```
aperm(a, perm) creates a new array. If a is a $k$ dimensional array, then the new array is also $k$ dimensional, but the dimension $\text{perm}[j]$ in the old array now becomes the $j$-th dimension of the new array:

```r
> x
   [,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6
> aperm(x,c(2,1)) ## here, aperm() is equivalent to t()
   [,1] [,2]
[1,]  1  2
[2,]  3  4
[3,]  5  6
```
> x<-array(1:12,dim=c(2,3,2))
> x

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

, , 2
 [,1] [,2] [,3]
[1,] 7 9 11
[2,] 8 10 12

# Without using your computer, tell us y[2,1,2] = ?
> y<-aperm(x,c(3,1,2))
Generalized transpose of an array

```r
> y[2,1,2]
[1] 9
```
### Outer product of two arrays

$a \odot b$ creates a new array $c$, $\text{dim}(c) = \text{c}(\text{dim}(a), \text{dim}(b))$, and data vector in $c$ is obtained by forming all possible products of elements of the data vector of $a$ with those of $b$:

```r
> a
 [,1] [,2]  
[1,]   1   3  
[2,]   2   4  
> b
 [,1] [,2]  
[1,]   1   2  
> a %o% b
 , , 1, 1
 [,1] [,2]  
[1,]   1   3  
[2,]   2   4  
, , 1, 2
 [,1] [,2]  
[1,]   2   6  
[2,]   4   8
```
Linear equations and inversion

\[ x_1 + 3x_2 = 1 \]
\[ 2x_1 + 4x_2 = -1 \]
\[ x_1 = ?, \ x_2 = ? \]
\[ A \cdot x = b \]
\[ x = A^{-1} \cdot b \]
Linear equations and inversion

\[ A = \begin{bmatrix}
1 & 3 \\
2 & 4
\end{bmatrix} \]

\[ x = \begin{bmatrix}
1 \\
-1
\end{bmatrix} \]

\[ b = A \times x \]

\[ \text{solve}(A, b) \] gives \( x \)

\[ \text{solve}(A) \] inverse of \( A \)

\[ \begin{bmatrix}
-2 & 1.5 \\
1 & -0.5
\end{bmatrix} \]
Linear equations and inversion

- `x <- solve(A) %*% b` is numerically inefficient and potentially unstable.
- `solve(A, b)` is preferred.
- `t(x) %*% solve(A, x)` is better than `t(x) %*% solve(A) %*% x`
Factors

Factors are used to represent discrete classifications (categorical data). They can be thought of as integer vectors where each integer has a label:

```r
> state<-c("MD","CA","MD","MD","CA","CA")
> state
[1] "MD" "CA" "MD" "MD" "CA" "CA"
> statef<-factor(state)
> statef
[1] MD CA MD MD CA CA
Levels: CA MD
> table(statef)
statef
CA  MD
3   3
> unclass(statef)
[1] 2 1 2 2 1 1
attr(,"levels")
[1] "CA" "MD"
```
Factors are useful in statistical analysis such as linear regression, ANOVA, generalized linear regression.

Using factors with labels to represent categorical data is better than using integers because factors are self-describing.
A useful function is `tapply()` which applies a function to each group of values given by levels of a factor:

```
> income
[1] 10 12 9 13 8 17
> statef
[1] MD CA MD MD CA CA
Levels: CA MD

> tapply(income, statef, mean)
    CA     MD
   12.333 10.667
```
Factors can be ordered or unordered:

```r
> x<-c("Medium","High","Low","Low","High")
> factor(x)
[1] Medium High Low Low High
Levels: High Low Medium

## useful for linear modelling, specifies the baseline level
> factor(x, levels=c("Low", "Medium", "High"))
[1] Medium High Low Low High
Levels: Low Medium High

## levels have natural ordering which we want to use
> ordered(x, levels=c("Low", "Medium", "High"))
[1] Medium High Low Low High
Levels: Low < Medium < High
```
List is an object that contains a collection of objects known as *components*. Components of a list can have different modes or types (i.e. they could belong to different classes), or dimensions.

```r
> x <- list(course = "computing", active = TRUE, grade = c(8, 10, 9))
> x

$course
[1] "computing"

$active
[1] TRUE

$grade
[1] 8 10 9
```
Components of a list can be accessed using `[[ ]]`: 

```r
> x <- list(course = "computing", active = TRUE, grade = c(8, 10, 9))
> x[[1]]
[1] "computing"

> x[[3]]
[1] 8 10 9

> x[[3]][2]
[1] 10
```
Lists

[[ ]] and [ ] have different meanings:

```r
## [[ ]] selects a single element
## [ ] can select multiple elements
> x[[2:3]]
Error in x[[2:3]] : subscript out of bounds
> x[2:3]
$active
[1] TRUE
$grade
[1] 8 10 9

## [] returns a list, not true for [[]]
> y<-x[3]
> class(y)
[1] "list"
> z<-x[[3]]
> class(z)
[1] "numeric"
```
List components can also be accessed via names:

```r
> names(x)
[1] "course" "active" "grade"
> x$grade
[1] 8 10 9
> x$grade[2]
[1] 10

> x$course
[1] "computing"
> x[['course']]  # Using double square brackets
[1] "computing"
> x$cour
[1] "computing"
```
Two lists can be combined using `c()`:

```r
> y <- list(dept = "biostatistics")
> z <- c(x, y)
> z
$course
[1] "computing"

$active
[1] TRUE

$grade
[1]  8 10  9

$dept
[1] "biostatistics"
```
> load("student.rda")
> ls()

What is the data structure of Student.
The min score of the second student + Mary’s third score = ?
> min(Student[[2]]$grade) + Student$Mary$grade[3]
[1] 177
Data frames

Data frames are used to store tabular data

- They are lists with class “data.frame”
- Each element in the list must have the same length
- Unlike matrices, columns can store different classes of objects
- Have a special attribute called row.names
- Can be converted to a matrix by data.matrix()
> x <- data.frame(id=1:4, val=c(T,F,T,F))
> x
   id  val
 1  1   TRUE
 2  2  FALSE
 3  3   TRUE
 4  4  FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
In R, every object comes from a class. Class defines behaviors of operations:

```r
> x <- data.frame(id=1:4, val=c(T, F, T, F))
> x
  id  val
 1 1   TRUE
 2 2   FALSE
 3 3   TRUE
 4 4   FALSE
> unclass(x)
$id
[1] 1 2 3 4
$val
[1] TRUE FALSE TRUE FALSE
attr("row.names")
[1] 1 2 3 4
```