R Programming
Lab 2
Data Type

Lecturer：Professor Pao-Ann Hsiung

Teaching Assistant: Hendrik

Embedded Systems Laboratory
National Chung Cheng University
Chiayi, Taiwan-62102

Outline

- Data Type
- Demo & Requirements
- Grading Policies
- Turn In

Data Types

Objects

- R has five basic or “atomic” classes of objects:
  - Character
  - numeric (real numbers)
  - Integer
  - Complex
  - logical (True/False)

```
> x <- c(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex
```

- The most basic object is a vector
  - A vector can only contain objects of the same class
  - BUT: The one exception is a list, which is represented as a vector but can contain objects of different classes (indeed, that’s usually why we use them)

Numbers

- Numbers in R a generally treated as numeric objects (i.e. double precision real numbers)
- If you explicitly want an integer, you need to specify the L suffix
- Ex: Entering 1 gives you a numeric object; entering 1L explicitly gives you an integer.
- There is also a special number Inf which represents infinity; e.g. 1/0; Inf can be used in ordinary calculations; e.g. 1/Inf is 0
- The value NaN represents an undefined value ("not a number") e.g. 0 / 0; NaN can also be thought of as a missing value (more on that later)

Attributes

- R objects can have attributes
  - names, dimnames
  - dimensions (e.g. matrices, arrays)
  - Class
  - Length
  - other user-defined attributes/metadata

Attributes of an object can be accessed using the attributes() function

```
x <- cbind(a = 1:3, pi = pi)
# simple matrix with dimnames attributes(x)
## strip an object's attributes:
attributes(x) <- NULL x
# now just a vector of length 6
mostattributes(x) <- list(mycomment = "really special", dim = 3:2,
dimnames = list(LETTERS[1:3], letters[1:5]), names = paste(1:6))
x # dim(), but not {dim}names
```

Vectors

- A vector is a variable containing multiple values

Creating vector

- Method 1 for creating a vector: c function
  `Wingcrd <- c(59, 55, 53.5, 55)`
- Method 2 for creating a vector:
  `Id <- 1 : 8`
- Method 3 for creating a vector

```
Id <- rep(c(1, 2, 3, 4), each = 8)
Id <- rep(1 : 4, each = 8)
a <- seq(from = 1, to = 4, by = 1)
Id <- rep(a, each = 8)
```

```
> x <- c(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex
```

- Method 4 for creating a vector
  define a vector by length ***vector ( )***

```
> x <- vector("numeric", length = 10)
> x
[1] 0 0 0 0 0 0 0 0 0 0
```

Limitation: Vector has only one dimesion

c = concatenate

Vectors (con't)

- **Operating Vectors**
  - How to access elements of a vector?
    - A <- Wingcrd[1] # the first value
      B <- Wingcrd[1 : 5] # the first five values
      C <- Wingcrd[-2] # all except the second value
      D <- Wingcrd[c(1, 3, 5)] # the first, third, and fifth values
    - Do not go outside the range of allowable values
      - E.g. Wingcrd[9] is not defined!
  - Functions for a vector
      - E.g. sum(Wingcrd), mean(Wingcrd), max(Wingcrd) etc.
  - Combining vectors into a longer one
      - E.g. c(Wingcrd, Tarsus, Head, Wt)
  - To know the length of a vector
      - E.g. **length** (Wingcrd)

Explicit Coercion

- Mixing objects

```
> y <- c(1.7, "a") ## character
> y <- c(TRUE, 2) ## numeric
> y <- c("a", TRUE) ## character
```

- When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class

Explicit Coercion

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

```
> x <- 0:6
> class(x)
[1] "integer"
> as.numeric(x)
[1] 0 1 2 3 4 5 6
> as.logical(x)
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
> as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"
```

Matrices

- Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

```
> m <- matrix(nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA

> dim(m)
[1] 2 3

> attributes(m)
$dim
[1] 2 3
```

constructed *column-wise* starting in the **"upper left"** corner and **running down** the columns

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

Matrices (cont.)

- Matrices can also be created directly from vectors by adding a dimension attribute

```
> m <- 1:10
> m
[1] 1 2 3 4 5 6 7 8 9 10
> dim(m) <- c(2, 5)
> m
[,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
```

cbind-ing and rbind-ing

Matrices can be created by column-binding or row-binding with **cbind()** and **rbind().**

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
x y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
[,1] [,2] [,3]
x 1 2 3
y 10 11 12
```

Accessing Matrices

- How to access part of a matrix?

```
A <- M[1, 1]                                # 1st row, 1st column
B <- M[2, ]          (M[2, 1 : 4])          # 2nd row
C <- M[, 1] (M[1 : 8, 1])                   # 1st column
D <- M[, 2 : 3]                 # 2nd and 3rd columns
E <- M[, c(1, 3, 4)]            # 1st, 3rd, 4th columns
F <- M[, -3]                    # all except the 3rd column
Msub <- M[1:3, 3:3]             # Create Submatrix
```

*Do not miss the comma!
*Do not go outside the range of allowable values!

Function and Operation for Matrices

- E.g.

$$D = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 2 & 1 \\ 2 & 3 & 0 \end{pmatrix}$$

Row names and column names

>rownames(D)

>colnames(D) <- c("Wingcrd", "Tarsus", "Head", "Wt")

Check whether an object is a matrix

>is.matrix(D)

Matrix transpose: t(D)
Matrix inverse: solve(D)
Matrix multiplication: D %*% solve(D)

Limitation!!
A matrix can only hold one type of data

Operator or Function	Description
A * B	Element-wise multiplication
A %*% B	Matrix multiplication
A %o% B	Outer product. AB'
crossprod(A,B) crossprod(A)	A'B and A'A respectively.
t(A)	Transpose
diag(x)	Creates diagonal matrix with elements of x in the principal diagonal
diag(A)	Returns a vector containing the elements of the principal diagonal
diag(k)	If k is a scalar, this creates a k x k identity matrix. Go figure.
solve(A, b)	Returns vector x in the equation b = Ax (i.e., $A^{-1}b$)
solve(A)	Inverse of A where A is a square matrix.
ginv(A)	Moore-Penrose Generalized Inverse of A. ginv(A) requires loading the MASS package.
y<-eigen(A)	y$val are the eigenvalues of A y$vec are the eigenvectors of A
y<-svd(A)	Single value decomposition of A. y$d = vector containing the singular values of A y$u = matrix with columns contain the left singular vectors of A y$v = matrix with columns contain the right singular vectors of A

Factors

- Represent categorical data
  - Treated specially by modelling functions : **lm() and glm()**
  - Using factors with labels is better than using integers because factors are self-describing; having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x
[1] yes yes no yes no
Levels: no yes
> table(x)
x
no yes
2 3
> unclass(x)
[1] 2 2 1 2 1
attr(,"levels")
[1] "no" "yes"
```

levels argument to ***factor()***

*****linear modelling

```
> x <- factor(c("yes", "yes", "no", "yes", "no"),
levels = c("yes", "no"))
> x
[1] yes yes no yes no
Levels: yes no
```

Missing values

- Missing values are denoted by NA or NaN for undefined mathematical operations
  - is.na() is used to test objects if they are NA
  - is.nan() is used to test for NaN
  - NA values have a class also, so there are integer NA, character NA, etc.
  - A NaN value is also NA but the converse is not true

```
> x <- c(1, 2, NA, 10, 3)
> is.na(x)
[1] FALSE FALSE TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE TRUE TRUE FALSE
> is.nan(x)
[1] FALSE FALSE TRUE FALSE FALSE
```

Data Frames

- Used to store tabular data
  - They are represented as a special type of list where every element of the list has to have the same length
  - Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
  - Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class
  - Data frames also have a special attribute called row.names
  - Data frames are usually created by calling read.table() or read.csv()
  - Can be converted to a matrix by calling data.matrix()

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
> x
foo bar
1 1 TRUE
2 2 TRUE
3 3 FALSE
4 4 FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

Data Frames

- Method for creating a data frame

```
Dfrm <- data.frame(WC = Wingcrd,
                   TS = Tarsus,
                   HD = Head,
                   W = Wt )
```

- Common usage of data frames
  - Enter the data into R
  - make changes to the data
  - store the modified data in a data frame

E.g.
```
Dfrm <- data.frame(WC = Wingcrd,
TS = Tarsus,
HD = Head,
W = Wt,
Wsq = sqrt(Wt))
```

Limitation!!
A data frame can only combine data of the same size

Accessing data frames

- How to access part of a data frame?
  - Method 1: E.g. Dfrm$W
    - Note that Dfrm$W and Wt are two different objects
    - rm(Wt) # remove the variable Wt
    - Check values of Wt and Dfrm$W
  - Method 2: the way accessing a matrix also works!
    - A <- Z[1, 1] # 1st row, 1st column
    - B <- Z[2,] (Z[2, 1 : 4]) # 2nd row
    - C <- Z[, 1] (Z[1 : 8, 1]) # 1st column

List

- Lists are a special type of vector that can contain elements of different classes. Lists are a very important data type in R and you should get to know them well.

- Lists allow combining data of different dimensions

```
X1 <- 1 : 3
X2 <- c(“a”, “b”)
X3 <- matrix(nrow = 2, ncol = 2)
 Y <- list(X1= X1, X2=X2, X3=X3)
```

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a“

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

- But data frames cannot

E.g. data.frame(X1= X1, X2=X2, X3=X3)

causes an error

Accessing Lists

- How to access part of a list
  - E.g. Y$X1
  - Similar as accessing a data frame (E.g. Dfrm$W)

- Many functions in R produce output as a list
  - E.g. M <- lm(WC ~ Wt, data = Dfrm)
  - names(M)
  - M$coefficients

Reading Data

- Functions reading data into R
  - read.table, read.csv, for reading tabular data
  - readLines, for reading lines of a text file
  - source, for reading in R code files (inverse of dump)
  - dget, for reading in R code files (inverse of dput)
  - load, for reading in saved workspaces
  - unserialize, for reading single R objects in binary form

```
data <- read.table("foo.txt")
# In this case, R will automatically
## • skip lines that begin with a #
## • figure out how many rows there are (and how much memory needs to be allocated)
## • figure what type of variable is in each column of the table.
```

Writing Data

- **write.table**, for writing tabular data to text files (i.e. CSV) or connections
- **writeLines**, for writing character data line-by-line to a file or connection
- **dump**, for dumping a textual representation of multiple R objects
- **dput**, for outputting a textual representation of an R object
- **save**, for saving an arbitrary number of R objects in binary format (possibly compressed) to a file.
- **serialize**, for converting an R object into a binary format for outputting to a connection (or file).

Demo & Requirements

Part1(Vector)

- Write your R to Calculate the following:

a) $$\sum_{i=10}^{100} \left(i^3 + 4i^2\right).$$

b) $$\sum_{i=1}^{25} \left(\frac{2^i}{i} + \frac{3^i}{i^2}\right)$$

Part2 (Matrices)

- Suppose $\mathbf{A} = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$
  - Check that $\boldsymbol{A^3} = \mathbf{0}$ where **0** is a 3 $\times$ 3 matrix with every entry equal to 0
- Create the following matrix **B** with 15 rows:

$$\mathbf{B} = \begin{bmatrix} 10 & -10 & 10 \\ 10 & -10 & 10 \\ \cdots & \cdots & \cdots \\ 10 & -10 & 10 \end{bmatrix}$$

  - Calculate the 3 $\times$ 3 matrix $\boldsymbol{B^T B}$. (Look at the help for crossprod.)

Part3(Data Frame)

- Go to e-course, download lab02.rar. It contains 2 files(traffic-data.csv and lab2_yourstudentID.R)

sectionID	sectionName	speed	occupancy	volume
1	zhong	32	12	58
2	zhong	34	9	359
3	zhong	NA	NA	NA
4	zhong	62	31	202
5	ximen	70	24	401
6	ximen	48	25	143
7	shan	47	NA	211
8	shan	31	25	NA
9	shan	42	42	344
10	ximen	25	66	32

Part3 (con't)

a) Use the *c function* to create variable "*speed", "occupancy"* and "*volume"* that contain values
b) Use the cbind() function to create a variable "dtbind" that combines *speed, occupancy* and *volume.* Then check whether "dtbind" is a matrix
c) create data frame from the variable that you have created above, store to a new variable, called "dtfrm"
d) read dataset (traffic-data.csv) use read.table() called "dtfrm1"
e) read dataset (traffic-data.csv) use read.csv() called "dtfrm2"
f) Use the names ,str, dim, head functions to check the correctness of the traffic-data.csv
g) why dimension dtfrm1 and dtfrm2 different?
h) Compute the following: how many observations were made?
i) What are the *minimum, mean, and maximum* speed, occupancy and volume without removing missing value
j) Create new dataset called 'trafficData' with removing NA
k) compute mean 'speed', mean 'volume' again (in dataset trafficData)

Result of Demo

- Show the following to TAs:
  - save your code to a file named *lab2_studentID.R*
  - *Follow lab2_yourstudentID.R*

Grading Policies

Part1 10%

Part2 10%

Part 3 80%

Turn In

Turn In

- The E-course System
  - http://ecourse.elearning.ccu.edu.tw/
- Upload Lab2_StudentID.zip into “Lab 2”

Turn In (cont'd)

- Office hour: (Lab 501B)
- TA's email: hendrik.gian@gmail.com