MAT 5030
Chapter 2:
R Environment

Kazuhiko Shinki

Wayne State University

Session Management

Working Directory

```
setwd("c:/Users/Kazuhiko/WSU/Teach/MAT5030-16W/Ch2")
  # (Useful if you read data from a file or
  #  output results to a file)
```

After this line, you only have to specify file names (e.g., ”File01.txt”) to access to the files in the working directory.

```
getwd() # show the current working directory
```

In OSX from the menu bar, you can choose ”Misc” to set, get or reset the working directly.

Terminating R

Type ”q()” or ”quit()” to quit R (at least the Windows version and Mac OSX version ask if you really want to terminate).

Session Management

Existing Objects

```
> ls() % show all objects you’ve created
[1] "A"         "Dataset1" "Dataset2" "M"          "QRT"        "RT"

> rm(A) % remove object A
> ls()
[1] "Dataset1" "Dataset2" "M"          "QRT"        "RT"

> rm(list=ls())  % remove all objects
> ls()
character(0)
```

Session Management

Save Workspace

- By GUI:
  - Windows: From the menu bar, “File” → “Save Workspace”.
  - Mac OSX: From the menu bar, “Workspace” → “Save Workspace File”.
- By Script: Type “save.image(”XXXX.RData”)”.

Note: The command saves all objects and functions you have created, but it does not save the scripts you entered (cf. ”savehistory”).

Load Workspace

- By GUI:
  - Windows: From the menu bar, “File” → “Load Workspace”.
  - Mac OSX: From the menu bar, “Workspace” → “Load Workspace File”.
- By Script: Type ”load(”XXXX.RData”)”.

Session Management

Output to a file
You may want to output large results into a text file.

```
> A <- 1:100
> sink("Output01.txt") # output into the file
> A^2  # output results to "Output01.txt"

> sink() # output on screen
> A[1:10]
 [1]  1  2  3  4  5  6  7  8  9 10

> sink("Output01.txt") # overwrite results (A^2 will be eliminated)
> A^3

> sink("Output01.txt",append=T) # append new results
> 2*A
```

Session Management

Input from a file

You may want to load your data, functions and environment from a file with one command “source”.

Suppose the source file “S1.txt” is as follows:

```
A <- 1:10
B <- diag(c(4,2))
```

```
> source("S1.txt") # load the code in "S1.txt"
> A
 [1]  1  2  3  4  5  6  7  8  9 10

> source("S1.txt", echo=T) # this command shows the code onscreen
> A <- 1:10
> B <- diag(c(4,2))
```

Session Management

History (Windows only)

The letters you entered in R are called history.

```
history(max.show=5) # show most recent 5 lines of the history
                    #  in a different window
savehistory("SH.txt") # Save history to "SH.txt"
loadhistory("SH.txt") # Load history from "SH.txt"
```

Note:
”loadhistory” only loads history, and does not run the codes in the file.

Session Management

A Summary of File Management

Import:

- Data: `'read.table'` etc.
- Run a code: `'source'`
- See a code: `'loadhistory'`
- State of the R session (R objects): `'load'`, “Load Workspace”.

Export:

- Data: `'sink'`, `'write.table'` etc.
- Save a code (as text): `'savehistory'` (Windows only).
- State of the R session (R objects): `'save.image'`, “Save Workspace”.

Session Management

Library

R has many add-on packages (sometimes called *libraries*) which include procedures for specific statistical analysis.

To use a package,

1. install the package (first time only):
   - By GUI: In the menu bar, ”packages” → ”Install packages”, then follow the instruction. R downloads the package via the Internet or using a local file.
   - By script: Type ”options(CRAN=”http://cran.r-project.org”)” then ”install.packages(”Rcmdr”)” (”Rcmdr” is the name of the package).
2. load the package (everytime you started R):
   - By GUI: In the menu bar, ”packages” → ”Load packages”.
   - By script: Type ”library(Rcmdr)”.

Session Management

Example: library e1071

The base R package does not have functions to calculate skewness and kurtosis. Several packages have such functions.

```
> X <- rnorm(100) # 100 random numbers following standard normal
> skewness(X) # no such a fucntion in the base package
Error: could not find function "skewness"

> install.packages("e1071") # install the package from the Internet
>library(e1071) # load the package in your R session

> skewness(X) # skewness of X
[1] 0.1657817
```

$$skewness = \frac{1}{n}\sum_{i=1}^{100}(x_i - \bar{x})^3/s^3$$

Session Management

To remove package, type

```
> detach("package:Rcmdr")
```

To see what packages are loaded, use ”search”:

```
> search()
 [1] ".GlobalEnv"        "package:MASS"      "package:boot"      "package:s
 [6] "package:grDevices" "package:utils"     "package:datasets"  "package:m
[11] "package:base"
```

Session Management

Dataframe: ”attach” and ”detach”

The ”attach” makes it possible to use variables in a dataframe without specifying the dataframe name.

```
> D
  Chase  Citi Amex
1  3000  4000 5000
2 20000 10000 8000

> D$Chase
[1]  3000 20000
> Chase   # R does not recognize where is "Chase".
Error: object "Chase" not found

> attach(D)
> Chase
[1]  3000 20000

> detach(D) # cancel "attach(D)"
```

Session Management

Dataframe: ”subset” and ”transform”

```
> D
          Chase  Citi Amex
Andy       3000  4000 5000
Bob       20000 10000 8000
Chris      4000  9000 6400
Dan        4000 10000 1500

> D2 <- subset(D, Amex > 6000)  # Make a subset of D
> D2
      Chase  Citi Amex
Bob   20000 10000 8000
Chris  4000  9000 6400

> transform(D2, Total = Chase + Citi + Amex) # add a column: Total
      Chase  Citi Amex Total
Bob   20000 10000 8000 38000
Chris  4000  9000 6400 19400
```

Session Management

An alternative way to add a column:

```
> attach(D2)
> Total <- Chase + Citi + Amex
> D3<- cbind(D2, Total)
> D3
      Chase  Citi Amex Total
Bob   20000 10000 8000 38000
Chris  4000  9000 6400 19400
> detach(D2)
```

Graphics

There are two tools to customize graphics:

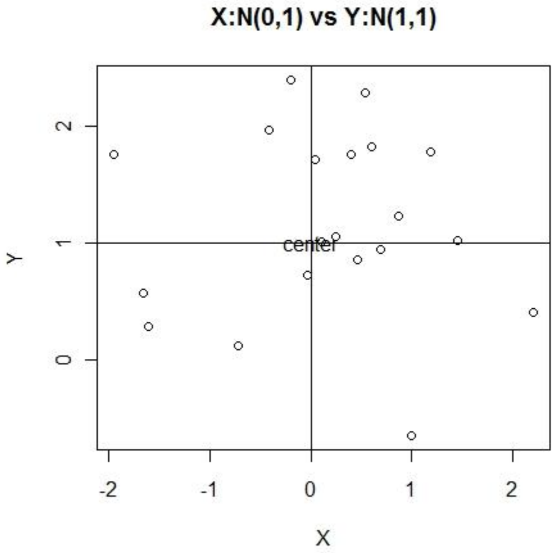
- Overlay simple objects such as axes, texts, legends one by one.
- Change the graphic parameters by using the ”par” command.

Graphics

Example 1: One by one

```
> plot(X,Y, axes = F) # plot only, no axes
> axis(1, at = c(-2,-1,0,1,2))  # label for X-axis
> axis(2, at = c(-1, 0,1,2,3))  # label for Y-axis
> abline(h = 1, v = 0)  # axes (Y=1 and X=0)
> box()  # draw an outer frame
> text(0,1,"center")  # put the text "center" at (0,1)
> title(main ="X:N(0,1) vs Y:N(1,1)") # title
```

Graphics



Graphics

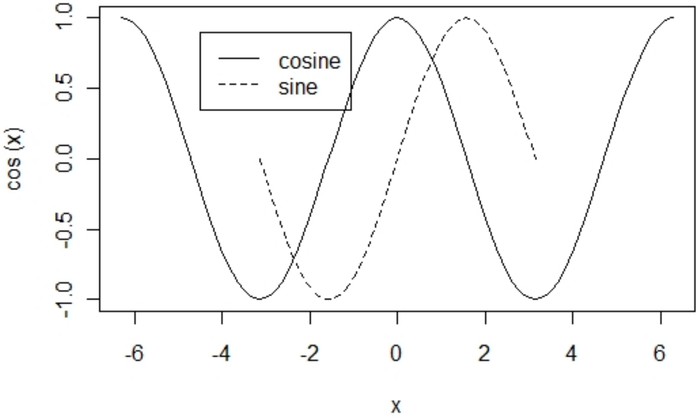
Example 2: Graphic parameters

```
>  plot(cos, -2*pi, 2*pi, lty=1) # cos curve on [-2pi, 2pi]
>  plot(sin, -pi, pi, lty=2, add=T) # sin curve on [-pi,pi]
>      # lty: line type, "add=T": overlay

> legend(-4.5,0.9, c("cosine","sine"), lty=1:2)
>      # add legend at (-4.5, 0,9)
```

Note: ”add=T” does not apply for more complicated plots such as ”plot(X,Y)”.

Graphics

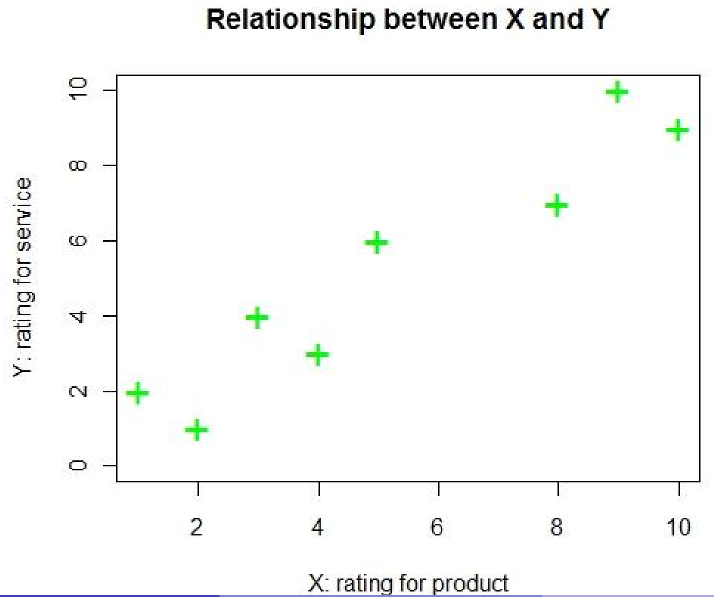


Graphics

Example 3: Graphic parameters

```r
X <- c(1,5,3,8,2,10,4,9)
Y <- c(2,6,4,7,1,9,3,10)
plot(X, Y,
        main="Relationship between X and Y", # title
        xlab ="X: rating for product",  # label for x
        ylab= "Y: rating for service",   # label for y
        ylim = c(0,10),  # range for y-axis
        pch="+",   # point type
        col="green",   # color
  cex = 2.0)   # point size (1.0 = normal)
```

Graphics


Relationship between X and Y
Y: rating for service
0
2
4
6
8
10
2
4
6
8
10
X: rating for product

Graphic parameters: Example 4

```
TempMI <- c(18,20,29,38,49,59,64,62,54,43,34,23)
TempWI <- c(9,14,25,35,46,56,61,59,50,39,28,16)

plot(TempMI, ylim=c(0,65), type="l", lty=1,
        xlab="month", ylab="Min Temp (F)")

par(new=T)  # overlay figures

plot(TempWI, ylim=c(0,65), type="l", lty=2,
        xlab="month", ylab="Min Temp (F)")

par(new=F)  # finish to overlay figures

legend(6, 20, c("Detroit, MI","Madison, WI"), lty=1:2) # legend at (6,20)
```

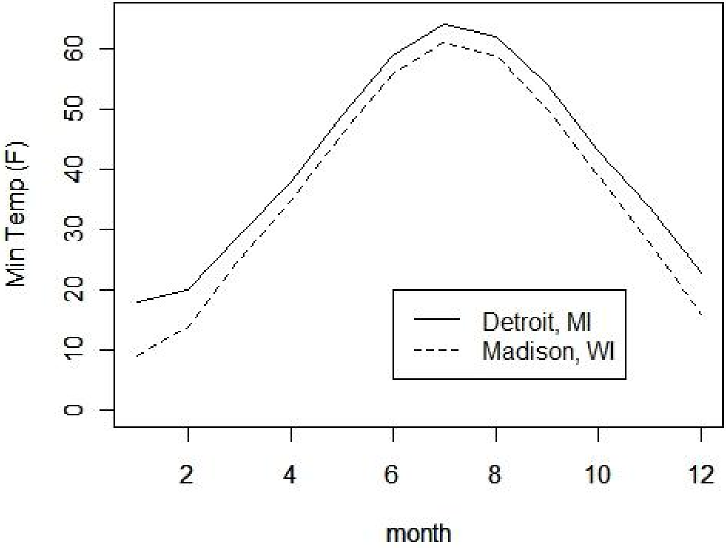
Graphics

Note:
The same range for y-axis (and x-axis) should be used to overlay two plots. To this end point, you may write:

```
> Ylim <- range(TempMI, TempWI) # range = c(Min,Max)
> Ylim
[1]  9 64
```

Then you write ”ylim=Ylim” in all ”plot()”.

Graphics


Min Temp (F)
0
10
20
30
40
50
60
2
4
6
8
10
12
month
Detroit, MI
Madison, WI

R Programing

Loop: ”while” and ”repeat”

Suppose we want to find the smallest n such that $2^n > N$, given a positive integer N. How to code?

- Code for $N = 123456789$.
- Code for general N.

R Programing

Example 1: while

```
> N <- 123456789
> n <- 1
> while (2^n <= N){ n <- n + 1} # increase n by 1 while "2^n <= N"
> n
[1] 27

> 2^27 # verify the result
[1] 134217728
```

R Programing

Example 2: repeat & if ~ break

```
> N <- 123456789
> n <- 1
> repeat{ (n <- n + 1)  # repeat until...
+ if (2^n > N) break   # 2^n > N
+ }
> n
[1] 27
```

R Programing

The “if” statements

Example 1:

When an integer ***n*** is given, print “congratulations!” on screen if and only if ***n*** is divisible by 9.

```
> n <- 17
> if (n %% 9 == 0){print("congratulations!")}
>
> n <- 18
> if (n %% 9 == 0){print("congratulations!")}
[1] "congratulations!"
```

R Programing

Example 2:

When a number X is given, let Y be 0 if X is negative and 1 if x is non-negative.

```
> X <- 0.3
> if (X < 0) Y <- 0 else Y <- 1
> Y
[1] 1

> X <- -0.5
> if (X < 0) Y <- 0 else Y <- 1
> Y
[1] 0
```

R Programing

Example 3: When a number ***X*** is given, let ***Y*** be 1 and ***Z*** be 2 if ***X*** is between -10 and 10, and ***Y*** be 0 and ***Z*** be -2 otherwise.

```
> X <- -12
> {
+         if ((X >= -10) & (X <= 10)){
+                 Y <- 1
+                 Z <- 2}
+                 else
+                 {Y <- 0
+                 Z <- -2}
+ }
> c(Y,Z)
[1]  0 -2
```

R Programing

Defining a function

We want to develop a **function** ”binarylength” which returns ***n*** when we input ***N***.

```
> binarylength <- function(N){  # define function "binarylength"
+ n <- 1
+   while (2^n <= N){ n <- n + 1}
+   n
+ }

> binarylength(123456789)
[1] 27
```

Note: ”+” appears in R when a function is defined using more than 1 line.

R Programing

We can create a function with multiple inputs and/or outputs. Suppose we want to create a function to find n such that $q^n > N$ when positive integers $q \geq 2$ and N are given. We want to output n and q^n.

```
> qadiclength <- function(q, N){
+ n <- 1
+ while (q^n <= N){ n <- n + 1}
+ c(n, q^n)
+ }

> qadiclength(3, 123456789)
[1]         17 129140163
```

R Programing

We can set default values for a function.

```
> qadiclength <- function(q = 2, N = 123456789){
+ n <- 1
+ while (q^n <= N){ n <- n + 1}
+ c(n, q^n)
+ }

> qadiclength()  # using default values
[1]        27 134217728

> qadiclength(q=3) # q has been changed to 3
[1]        17 129140163
```

R Programing

Loop: ”for”

Suppose we want to create a vector $(x_2 - x_1, x_3 - x_2, \cdots, x_n - x_{n-1})$ when $V = (x_1, \cdots, x_n)$ is given.

Unlike the examples for while and repeat, We always have to iterate calculation for a fixed number of times (i.e., $n - 1$ times).

R Programing

Example 1: Code using ”for”

```
> dV <- function(V){
+ n <- length(V)  # n = dimension of V
+ W <- numeric(n-1) # (n-1)-dim vector
+ for (i in 1:(n-1)){   # loop n is from 1 to (n-1)
+ W[i] <- V[i+1] - V[i]    # define i-th element of W
+ }
+ W
+ }

> X <- c(1,3,6,10,15)
> dV(X)
[1] 2 3 4 5
```

R Programing

Example 2: Code without using ”for”

Loop functions are slower than matrix algebra in R. (This is common among statistical languages, and uncommon among fundamental languages like C). Avoid loops if possible.

```
> dV2 <- function(V){
+ n <- length(V)
+     W <- V[(2:n)] - V[1:(n-1)]
+ W
+ }

> X <- c(1,3,6,10,15)
> dV2(X)
[1] 2 3 4 5
```

R Programing

A function to overlay plot

It is cumbersome to overlay two X-Y plots. Create a function to do it at once.

Input:

- x1: x-coordinates for the 1st data
- y1: y-coordinates for the 1st data
- x2: x-coordinates for the 2nd data
- y2" y-coordinates for the 2nd data

Output:

Save the figure as "G1.jpg".

R Programing

Function:

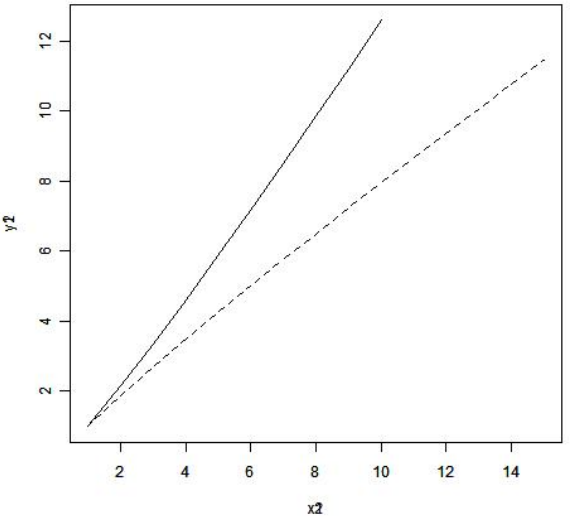
```
jpgout2 <-function(x1, y1, x2, y2, ...){
        Xlim <- range(x1, x2)
        Ylim <- range(y1, y2)
        jpeg("G1.jpg")
        plot(x1, y1, xlim= Xlim, ylim= Ylim, type = "l", lty=1, ...)
        par(new=T)
        plot(x2, y2, xlim= Xlim, ylim= Ylim, type = "l", lty=2, ...)
        par(new=F)
        dev.off()
}
```

Sample:

```
> X1 <- 1:10
> Y1 <- X1^1.1  # Y = X^1.1 on [1,10]
> X2 <- 1:15
> Y2 <- X2^0.9  # Y =X^0.9 on [1,15]

> jpgout2(X1, Y1, X2, Y2)
windows
      2
```

R Programing



Data Entry

Text files
Text file: ”Credit.txt”

Chase	Citi	Amex
3000	4000	5000
20000	10000	8000
4000	9000	6400
4000	10000	1500
5000	3000	5500

```
Data1 <- read.table("Credit.txt",sep="        ",header=TRUE)
        # sep:  specify a symbol for separation (tab, colon etc.)
        # header: TRUE if the 1st row is row names
        # "read.csv" is similar to "read.table"
```

To save a spread sheet as a text file, you can cut and paste data in a text editor or save as a tab delimited text file.

Data Entry

Clipboard

You can copy the data into the clipboard (”Ctrl + C” in Windows; ”Command + C” in Mac OSX), then input it into R by:

Windows:

```
> read.table("clipboard", header=T)
```

Mac OSX:

```
> read.table(pipe("pbpaste"), header=T)
Warning message:
In read.table(pipe("pbpaste"), header = T) :
  incomplete final line found by readTableHeader on 'pbpaste'
```

Data Entry

MS Excel

It is recommended to load data as a text or csv file, but there are a few options to load data from Microsoft Excel.

library(xlsx):

- Install and load the package “xlsx”.
  - The package “xlsxjars” is also required.
  - Java is also required. The R asks you to install the Java when you type `'library(xlsx)'`.
- Use the `'read.xlsx'` or `'read.xlsx2'` function to import the data. The latter is faster for the data with many rows.

Data Entry

MS Excel: Sample Code

```
> read.xlsx("Data1.xls",1, header=T) # 1: 1st sheet
  A  B  C
1 6 13  7
2 8  8 10
3 7 10  9
4 8  9  9
> read.xlsx2("Data1.xls",1, header=T)
> read.xlsx2("Data1.xlsx",sheetName="GHDay", header=T)
      Weight Height Color
1    7.4     23  dark
2    8.9     26 light
3    6.5     19  dark
4    7.2     24 light
5    5.9     18  dark
```

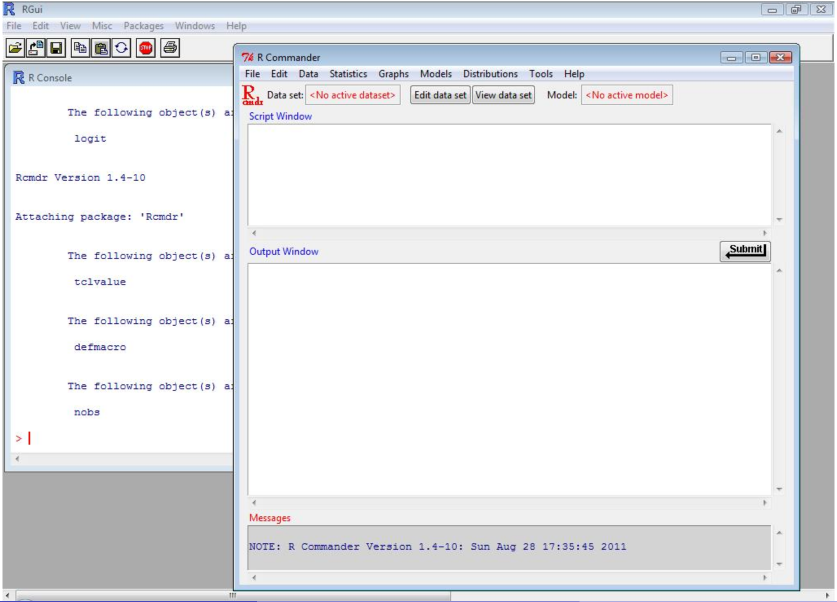
Data Entry

library(Rcmdr):

- Install and load the pacakage ”Rcmdr” (R commander).
- From the menu bar in R commander, choose ”Data” → ”Import Data” → ”from Excel, ...”.

Note: The R commander is a graphical user interface similar to more basic statistical softwares such as Minitab. You can do various statistical analysis using this interface.

Data Entry



RGui
File Edit View Misc Packages Windows Help
R Console
R Commander
File Edit Data Statistics Graphs Models Distributions Tools Help
Data set: <No active dataset>
Edit data set
View data set
Model: <No active model>
Script Window
Output Window
Submit
Messages

```
        The following object(s) a

         logit

Rcmdr Version 1.4-10

Attaching package: 'Rcmdr'

        The following object(s) a

         tclvalue

        The following object(s) a

         defmacro

        The following object(s) a

         nobs

> 
```

```
NOTE: R Commander Version 1.4-10: Sun Aug 28 17:35:45 2011
```

Data Entry

Some other formats

Some other formats such as SAS, SPSS, Minitab data files can be loaded by the ”foreign” package.

The R commander also has a interface to load such datasets.