

A Brief Introduction to R

Jun Li

Dept. of Statistics, Stanford Univ.

What is R

- A software & a dialect of the S language
- A system for statistical analysis and graphics
- Works in various OS.
- Free (under GNU General Public Licence)
- An interpreted language => easy to program.

A “Hello, World!” script

Just type

```
> print("Hello, World!")
```

Or

```
> hw <- "Hello, World!"
```

```
> print(hw)
```

How to get help

- To get the usage of a function (**“Examples”**, **“See Also”**)
 - `help("lm")`
 - `help(lm)`
 - `?lm`
- To open HTML help
 - `help.start()`
- To search a function
 - `help.search("lm")`
 - `apropos("lm")`

Basic operations of objects

- Creating:
 - `<-`
 - `=`
- Listing:
 - `ls()`
 - `ls(pat = "a")`
 - `ls(pat = "^a")`
- Deleting
 - `rm(a)`

Objects (1)

- Two intrinsic attribute: mode and length
- Four modes:
 - numeric
eg. `1`, `1.2`, `2.4e25`, `Inf`, `-Inf`, `NaN`
 - character
eg. `"this is R"`, `'this is R'`, `"hello, \n"`
 - complex
eg. `1+2i`, `1.1+2.4e4i`
 - logical (`FALSE` or `TRUE`)
- Length: the number of elements of the object

Objects(2)

- vector
- factor – a categorical variable
- array – a k -dimension “matrix”
- matrix
- data frame – a table composed with one or several vectors and/or factors all of the same length but possibly in different modes
- ts – time series
- list – a list can contain any type of object

Reading data in a file

- text (ASCII) files
- Excel, SAS, SPSS, ...
- Getting and setting working directory
 - `getwd()`, `setwd()`
- `read.table()`
- `scan()`
- `read.fwf()`

read.table() (1)

- The main way to read data in tabular form.
- `read.table(file, header = FALSE, sep = "\", ...)`
- Examples:
 - `> mydata <- read.table("fish.data")`
 - `> mydata <- read.table("c:/data/fish.data", TRUE, "\",")`
 - `> mydata <- read.table("http://www-stat.stanford.edu/~hastie/stats305/Data/abalone.305", TRUE, "\",")`

read.table() (2)

- `read.csv(...)`
- `read.csv2(...)`
- `read.delim(...)`
- `read.delim2(...)`

scan() and read.fwf()

- scan(): more flexible than read.table()
 - can specify the mode of the variables.
eg. `mydata <- scan("data.dat",
what=list("", 0))`
 - can be used to create different objects, such as vectors (default), matrices, data frames, lists, ...
- read.fwf(): Read in fixed width format
 - eg.
`No123.341.35`
`No224.531.45`
`read.fwf("fish.dat", widths = c(3,5,4))`

Saving data

- `write.table(x, file = "", append = FALSE, ...)`
- `write(x, file = "data", ncolumns = if(is.character(x)) 1 else 5, append = FALSE, sep = " ")`
- `save(x, y, "xy.RData")`
 - `load("xy.RData")`
- `save.image()`
 - `load(".RData")`

generating data (1)

- `x <- c(1, 2, 3, 4, 5)`
- `x <- 1 : 5`
- `x <- 1 : 5 - 1`
- `x <- seq(1, 5, 2)`
- `x <- rep(1, 5)`

Random sequences

- **rfunc(n, p1, p2, ...)**
 - **n**: the number of data generated
 - **p1, p2, ...**: parameters of the distribution
 - **rnorm, rexp, rgamma, rpois, rcauchy, rbeta, rt, rf, rchisq, rbinom, ...**
- Generally,
 - **rfunc**: generate random numbers
 - **dfunc**: the probability density
 - **pfunc**: the cumulative probability density (can be used to find p-value)
 - **qfunc**: the value of quantile (can be used to find critical values)

Creating objects (1)

- Vector

```
> vector("integer", 10)
```

```
> vector("logical", 10)
```

```
> vector("numeric", 10)
```

```
> vector("character", 10)
```

Or

```
> integer(10)
```

```
> logical(10)
```

```
> numeric(10)
```

```
> character(10)
```

Creating Objects (2)

- Factor

- > `factor(1:3)`

- > `factor(1:3, levels = 1:4)`

- > `factor(1:2, labels = c("Male", "Female"))`

- > `factor(1:4, exclude=3)`

Creating Objects (3)

- Matrix

```
matrix(data = NA, nrow = 1, ncol =  
1, byrow = FALSE, dimnames = NULL)
```

```
> matrix(3, 2, 3)
```

```
> matrix(1:6, 2, 3)
```

```
> matrix(1:6, 2, 3, TRUE)
```

Creating Objects (4)

- Data Frame

```
> data.frame(..., row.names = NULL, ...)
```

```
> x <- 1:4
```

```
> y <- 1:2
```

```
> z <- c("no1", "no2", "no3", "no4")
```

```
> datFr <- data.frame(x, y, z)
```

```
> datFr <- data.frame(N1=x, N2=y, N3=z)
```

Creating Objects (5)

- List

```
> x <- 1:4
```

```
> y <- 1:2
```

```
> z <- c("no1", "no2", "no3", "no4")
```

```
> xyzList <- list(x, y, z)
```

```
> xyzList <- list(N1=x, N2=y, N3=z)
```

- Time-series, expression

Converting objects (1)

- `as.something()`
 - `as.numeric()`, `as.integer()`,
`as.logical()`, `as.character()`, ...
 - `as.matrix()`, `as.data.frame()`, ...
- Generally, conversion follows intuitive rules.

Converting Objects (2)

- Converting factors into numeric values

```
- > fac <- factor(c("Male", "Female"))
```

```
> num <- as.numeric(fac)
```

```
- > fac <- factor(c(1, 2))
```

```
> num <- as.numeric(fac)
```

```
- > fac <- factor(c(1, 5))
```

```
> num <- as.numeric(fac)
```

Converting Objects (3)

- How to keep the levels as they are originally specified?
- Answer: first convert it into character, then into numeric.

```
> fac <- factor(c(1, 5))
```

```
> num <-
```

```
  as.numeric(as.character(fac))
```

Operators

- Arithmetic

`+, -, *, /, ^, %, %/, %/`

- Comparison

`<, <=, >, >=, ==, !=`

- Logical

`!, &, &&, |, ||`

Identical comparison

- `==`, `identical()`, and `all.equal()`
- Eg:
 - > `a <- c(1, 1.1-0.2)`
 - > `b <- c(1, 0.9)`
 - > `a == b`
 - > `identical(a, b)`
 - > `all.equal(a, b)`

The indexing system

- Vector

```
x[3], x[1:3], x[c(1, 3)], x[-1], x[-  
  (1:3)], x[-c(1, 3)], ...
```

```
x[x >= 5], x[x %% 2 == 0], x[x == 1],  
  x[c(FALSE, TRUE)], ...
```

- Matrix

```
x[2, 3], x[2, ], x[ ,3], x[ , -(1:2)], ...
```

```
x[2, 3, drop = FALSE), ...
```

- Array

```
x[3, 2, 4], ...
```

names

- Vector

```
names(x) <- c("no1", "no2", "no3")
```

- Matrix

```
rownames(x) <- c("r1", "r2", "r3")
```

```
colnames(x) <- c("c1", "c2", "c3")
```

```
dimnames(x) <- list(c("a", "b"), c("c", "d"), c("e",  
"f"))
```

- Data Frame

Simple functions

- `exp, log, log10, log2, sin, cos, tan, ctan, asin, acos, atan, choose, ...`
- `abs, sqrt, ...`
- `sum, max, min, which.max, which.min, range, mean, median, var, cov, cor, ...`
- `round, rev, sort, rank, ...`
- `scale, unique, na.omit, sample, ...`
- ...

Matrix computation

- `matrix()`
- `rbind()`, `cbind()`
- `+`, `-`, `*`, `/`
- `+`, `-`, `%*%`, `solve()`
- `t()`, `det()`, `diag()`, `qr()`,
`eigen()`, `svd()`, ...

Graphics

- R offers a remarkable variety of graphics.
- `demo(graphics)`, `demo(persp)`
- The result of a graphic function cannot be assigned to an object but is sent to a graphical device.
- Two kinds of graphical functions:
 - High-level plotting functions
 - Low-level plotting functions

Graphical devices

- `x11()`, `postscript()`, `pdf()`,
`png()`, `bmp()`, `jpeg()`, ...
- `dev.list()`
- `dev.cur()`
- `dev.set()`
- `dev.off()`

layout

- `layout(mat)`

```
> mat <- matrix(1:4, 2, 2)
```

```
> mat <- matrix(1:4, 2, 2, byrow =  
TRUE)
```

```
> mat <- matrix(c(1:3, 3), 2, 2)
```

High-level graphical functions

- `plot(x)`
- `plot(x, y)`
- `boxplot(x)`
- `pairs(x)`
- `hist(x)`
- `qqplot(x, y)`
- `image(x, y, z)`
- `persp(x, y, z)`
- ...

some common options

- **type**: specifies the type of plot, “p”, “l”, ...
- **xlim, ylim**: specifies the lower and upper limits of the axes.
- **xlab, ylab**: annotates the axes
- **main**: the main title
- **sub**: sub-title (written in a smaller font)

Low-level plotting commands

- `points(x, y)`
- `lines(x, y)`
- `text(x, y, labels, ...)`
- `segments(x0, y0, x1, y1)`
- `abline(a, b)`
- `abline(h=y)`
- `abline(v=x)`
- `rect(x1, y1, x2, y2)`
- `title(str)`
- ...

Graphical parameters

- Par gives us more control to the drawing.

```
> opar <- par( )
```

```
> par(...)
```

```
> plot...
```

```
> par(opar)
```

Statistical Analyses

Example:

```
> mydata <-  
  read.table("c:/data/fish.txt",  
            header = TRUE)  
  
> pairs(mydata)  
  
> mydata.lm <- lm(weight ~ length +  
  width, data = mydata)  
  
(or) mydata.lm <- lm(mydata$weight  
  ~ mydata$length + mydata$width)  
  
> summary(mydata.lm)  
  
(or) print(mydata.lm)
```

Formulae

- A formula is typically of the form $y \sim \text{model}$.

- Common models:

$a + b$

$X = X[, 1] + X[, 2] + X[, \text{ncol}(X)]$

$a : b$ only the interaction

$a * b = a + b + a : b$

$\text{poly}(a, n)$

$(a + b) ^ n$

-b: removes the effect of b

-1: a regression through the origin

1: only the intercept

Using the result

- Example(cont.)

```
> names(mydata.lm)
```

```
[1] "coefficients"  "residuals"  
    "effects"      "rank"
```

```
[5] "fitted.values" "assign"  
    "qr"           "df.residual"
```

```
[9] "xlevels"       "call"  
    "terms"        "model"
```

```
> mydata.lm$df.residual
```

Generic functions

- Functions that act specifically with respect to the class of the object.
- Examples:
 - `print`
 - `summary`
 - `df.residual`
 - `coef`
 - `residuals`
 - `logLik`
 - ...

Writing functions

- Example

```
## wrapper.R
wrapper <- function(x, y, sigma, hat = FALSE)
{
  source("llr.R");    # load function llr

  n <- length(y);
  H_hat <- rep(0, n^2);
  dim(H_hat) <- c(n, n);

  for (i in 1 : n)
  {
    H_hat[i,] <- llr(x, y, x[i], sigma, TRUE);
  }
}
```

```
df <- sum(diag(H_hat));

if (hat == FALSE)
{
  ret_val <- df;
}
else
{
  ret_val <- H_hat;
}

return(ret_val);
}
```

execute the function

```
> source("wrapper.R")  
> val <- wrapper(data$x, data$y,  
100)
```

Vectorization (1)

- Loops make R slow.
- Example

```
> x <- 1:500000
```

```
> y <- 1:500000
```

compare

```
> z <- x + y
```

with

```
> for (i in 1:500000) z[i] <- x[i]  
  + y[i]
```

Vectorization (2)

```
x <- 1:500000
Compare z <- x[x %% 2 == 0] with
z <- numeric(250000)
j <- 1
for (i in 1 : 500000)
{
  if (x[i] %% 2 == 0)
  {
    z[j] <- x[i]
    j <- j + 1
  }
}
```

Vectorization (3)

- Using `apply()`

- `apply(X, MARGIN, FUN, ...)`

- Examples:

- `x <- matrix(1:250000, 500, 500)`

- `cmean <- apply(x, 2, mean)`

- `rmean <- apply(x, 1, mean)`

Reference

[1] R for Beginners*. Emmanuel Paradis.

[2] An Introduction to R. John Verzani.

* This is the main reference for my PPT.
Most parts of my PPT are selected from
this reference and according to its order,
and I also use some of its examples.