Basic R Programming

Author: Chiachun Chiu (jiajyun.ciou@gmail.com (mailto:jiajyun.ciou@gmail.com))

2015年03月16日

About R 

> R (http://www.r-project.org/about.html) is a **language** and **environment** for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R. …… R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

How/where to obtain and install R?

If you use linux as your default os, you can install R from the package repositories of each distribution directly. Alternatively, you can download R binary-version or source code from **CRAN** if you use M$ windows or Mac OS.

Ubuntu users

- Add **deb http://my.favorite.cran.mirror/bin/linux/ubuntu (http://my.favorite.cran.mirror/bin/linux/ubuntu) [version name of ubuntu]/** into sources.list
- sudo apt-key adv –keyserver keyserver.ubuntu.com –recv-keys E084DAB9
- sudo apt-get update
- sudo apt-get install r-base

Redhat/Fedora/CentOS users

- Add **EPEL** repository
- sudo yum install R

Other OS users - CRAN (mirror-sites in Taiwan)

- NTU (http://cran.csie.ntu.edu.tw/)
- YZU (http://ftp.yzu.edu.tw/CRAN/)

Using  as your default R-programming IDE

About RStudio (http://www.rstudio.com/)

> RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

Install the most suitable version of RStudio for your needs.

- Desktop version: Access RStudio locally.
- Server version: Access via a web browse.

Other choices

-  : Revolution R (http://www.revolutionanalytics.com/revolution-r-enterprise)
-  : Vim-R-plugin (http://www.lepem.ufc.br/jaa/vim-r-plugin.html)
- Any other text editors: gedit, emacs(+ESS), eclipse and etc.

Resources of learning R

Books & web sites

- The R Book (http://as.wiley.com/WileyCDA/WileyTitle/productCd-0470973927.html)
- Introduction to Probability and Statistics Using R (http://ipsur.org/)
- Introduction to Scientific Programming and Simulation Using R (http://www.amazon.com/Introduction-Scientific-Programming-Simulation-Chapman/dp/1420068725)
- The Art of R Programming (http://www.amazon.com/The-Art-Programming-Statistical-Software/dp/1593273843)
- R Tutorial (http://www.cyclismo.org/tutorial/R/)
- Programming in R (http://manuals.bioinformatics.ucr.edu/home/programming-in-r)
- Advanced R (http://adv-r.had.co.nz/)

Forums & communities

- R-bloggers (http://www.r-bloggers.com/)
- Taiwan R User Group (http://www.meetup.com/Taiwan-R)
- Stack Overflow (http://stackoverflow.com/questions/tagged/r)
- Ptt R_Language (https://www.ptt.cc/bbs/R_Language/index.html)

Blog

- 糗世界 (http://216.49.144.90:8080/rbioconductor-2)

The most important step when begining to learn R is using help()

help() & help.search()

```
help(help)
help.search("standard deviation")
```

? & ??

```
?mean
??hypergeometric
```

Package installation & PATH setting

Installing packages

```
# Download Pkgs from CRAN repository & install
install.packages('rmarkdown',                  # Package name
                 repo="http://cran.csie.ntu.edu.tw", # The url of CRAN-repository
                 destdir="~/Download",            # The directory where downloaded pkgs are stored
                 lib=.libPaths()[1])              # The directory where to install pkgs

# Install Pkgs from downloaded source code
install.packages('~/Download/rmarkdown_0.5.1.tar.gz',
                 repos=NULL,
                 type="source",
                 lib=.libPaths()[1])
```

Setting PATH

```
.libPaths(new)
```

Some Pkgs shoud be downloaded/installed from R-forge (https://r-forge.r-project.org/)

Set `install.packages(Pkg, repo='http://R-Forge.R-project.org')`

Using the package installed

```
library(Pkg)
require(Pkg) # Avoid to use this!
```

What is the difference between require() and library() (http://stackoverflow.com/questions/5595512/what-is-the-difference-between-require-and-library)

Bioconductor

About Bioconductor (http://www.bioconductor.org/)

> Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development.

Install Pkgs from Bioconductor

```
source("http://bioconductor.jp/biocLite.R") # Mirror site in Japan
biocLite("Package Name")
```

Basic operation

```
5+5
5-3
5*3
5/3
5^3
10%%3

# Variable declaration
x <- 5 # '<-' is assign operator in R, which is equivalent to '='
y <- function(i) mean(i)
```

Data and object types

Data types

- numeric: c(1:3, 5 ,7)
- character: c("1","2","3"); LETTERS[1:3]
- logical: TRUE; FALSE
- complex: 1, b, 3

Object types

- **vector**: the data types of all elements in a vector must be consistent!

```
x <- 1:5
y <- c(6,7,8,9,10)
z <- x - y
print(z)
```

```
## [1] -5 -5 -5 -5 -5
```

```
# Vectorized code performs better!
a <- 1:100000
system.time(mean(a))
```

```
##    user  system elapsed
##       0       0       0
```

```
total <- 0
system.time(for (i in a) {total <- total + i; total/100000})
```

```
##    user  system elapsed
##   0.045   0.000   0.045
```

- **matrix**

```
x <- matrix(rnorm(100), nr=20, nc=5)
print(x)
```

```
##             [,1]        [,2]       [,3]       [,4]       [,5]
##  [1,] -0.6754317  0.134148660  0.5927052 -0.5530251 -1.1088951
##  [2,] -1.4228888 -0.084036350  1.3801329  0.3093756 -0.6960788
##  [3,] -0.8292911 -0.004618448  0.0187798 -0.4626981 -0.8628353
##  [4,]  0.7258537 -0.844312745 -0.5712349  0.5594801  1.9541366
##  [5,] -0.2384304 -0.088841066 -0.3596803 -0.8754051  0.8005695
##  [6,] -0.1479699  0.216153107 -0.8839655 -0.1728935 -1.4268509
##  [7,]  0.0563501 -0.863779452  1.6639431 -2.3768348 -2.1984382
##  [8,]  0.3070430 -1.231048421  0.7757296 -0.8868931 -0.8926329
##  [9,]  0.4682441 -0.788113191 -0.0150292  1.6968962 -0.6793644
## [10,]  1.3287126 -0.368290858 -1.2309210  0.6210978  0.4145185
## [11,]  1.6872157 -0.744221892  0.1633968  1.1851491  0.1302009
## [12,]  1.4550518 -0.068929741 -0.4308870 -0.5612326 -0.4651322
## [13,]  0.4348402  1.078654609  0.3354110 -2.4280680  1.3842205
## [14,] -0.2141863 -1.208897862  0.3920276 -1.6319501  0.8399402
## [15,] -0.8520265  1.291777317 -0.3774625  1.7005098  0.9069191
## [16,]  0.2974654  1.131435329  0.8101829  2.9755471 -1.5567814
## [17,]  1.1463924 -0.677693268 -1.9367018 -0.4480987 -1.5951580
## [18,] -0.1996670 -0.616000483  0.9678444  0.2118683  0.6534989
## [19,]  0.3501281  0.492866434 -0.8670774 -1.2095160  0.4817640
## [20,]  1.7111155 -1.017105619  1.7264748  0.5923495 -0.2378380
```

```
x[1,3]
x[2:4,]
x[,3:5]
x %*% t(x)

# A matrix is a vector with subscripts!
x[1:3]
x[1:3,1]
```

- **array**

```
y <- array(rnorm(64), c(8,4,2))
print(y) # An array is also a vector with subscripts!
```

```
## , , 1
##
##           [,1]       [,2]       [,3]       [,4]
## [1,] -1.0742369  0.06601057  1.02202297  0.3853873
## [2,] -0.2578930  0.29666649  0.35217231 -1.5104227
## [3,]  0.6719079 -1.78392593 -0.75034698  0.5365331
## [4,] -0.1414159  0.37590162  0.52602126  1.2179640
## [5,]  0.7520959 -0.98978186 -0.04598731 -0.2733599
## [6,] -1.6396862 -0.35024146  0.70217723 -0.6721059
## [7,] -0.3051964  2.33603654 -1.76391917 -0.7244794
## [8,]  0.9442586  2.48152649 -0.28601087  0.7679506
##
## , , 2
##
##           [,1]       [,2]       [,3]       [,4]
## [1,] -0.95000902 -0.69788492 -0.3575473  0.3226971
## [2,]  1.63694952  0.02535667 -0.3954249  0.2357777
## [3,] -0.67501902 -0.76218857  0.8691595 -0.9661859
## [4,]  0.53366608  0.67272887  1.1548069  1.2198832
## [5,]  1.25008984 -0.70611021  0.3154586 -0.5865808
## [6,]  0.02947842  0.69886675 -0.2581292 -2.7711122
## [7,] -0.71954344 -0.16734232 -0.4298188 -2.8059128
## [8,]  0.80218395  0.27347526  1.4230249 -1.2853159
```

- **list**: the data types of elements in a list could be complex

```
x<-list(1:5, c("a","b","c"), matrix(rnorm(10),nr=5,nc=2))
print(x)
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "a" "b" "c"
##
## [[3]]
##           [,1]       [,2]
## [1,]  0.53909696  1.4274476
## [2,] -0.07313031 -0.4988201
## [3,] -0.15451398 -0.1749385
## [4,]  0.22246201 -1.4697027
## [5,]  0.80046207 -2.8300532
```

```
x$mylist <- x
print(x)
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "a" "b" "c"
##
## [[3]]
##            [,1]       [,2]
## [1,]  0.53909696  1.4274476
## [2,] -0.07313031 -0.4988201
## [3,] -0.15451398 -0.1749385
## [4,]  0.22246201 -1.4697027
## [5,]  0.80046207 -2.8300532
##
## $mylist
## $mylist[[1]]
## [1] 1 2 3 4 5
##
## $mylist[[2]]
## [1] "a" "b" "c"
##
## $mylist[[3]]
##            [,1]       [,2]
## [1,]  0.53909696  1.4274476
## [2,] -0.07313031 -0.4988201
## [3,] -0.15451398 -0.1749385
## [4,]  0.22246201 -1.4697027
## [5,]  0.80046207 -2.8300532
```

- **data frame**: a data frame is collection of multiple lists with the same length

```
df<-data.frame(num=1:10,
        char=LETTERS[1:10],
        logic=sample(c(TRUE,FALSE), 10, replace=TRUE))

df
```

```
##    num char logic
## 1    1    A  TRUE
## 2    2    B FALSE
## 3    3    C FALSE
## 4    4    D  TRUE
## 5    5    E FALSE
## 6    6    F  TRUE
## 7    7    G  TRUE
## 8    8    H FALSE
## 9    9    I  TRUE
## 10  10    J  TRUE
```

```
df$char
```

```
##  [1] A B C D E F G H I J
## Levels: A B C D E F G H I J
```

```
df$logic[5:7]
```

```
## [1] FALSE  TRUE  TRUE
```

- **factor**: An R **factor** might be viewed simply as **a vector with a bit more information added** (though, as seen below, it's different from this internally). That extra information consists of a record of the distinct values in that vector, called **levels**.

```
x <- c(5, 12, 32, 12)
xf <- factor(x)
print(xf)
```

```
## [1] 5  12 32 12
## Levels: 5 12 32
```

So…. a factor looks like a vector, right?

```
str(xf) # Here str stands for structure. This function shows the internal structure of any R object.
```

```
##  Factor w/ 3 levels "5","12","32": 1 2 3 2
```

```
unclass(xf)
```

```
## [1] 1 2 3 2
## attr(,"levels")
## [1] "5"  "12" "32"
```

```
length(xf)
```

```
## [1] 4
```

What??? What are you talking about?

```
x <- c(5, 12, 13, 12)
xff <- factor(x, levels=c(5, 12, 13, 88))
xff
```

```
## [1] 5  12 13 12
## Levels: 5 12 13 88
```

```
xff[2] <- 88
xff
```

```
## [1] 5  88 13 12
## Levels: 5 12 13 88
```

```
xff[2] <- 28 # You cannot sneak in an "illegal" level
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = 28): invalid factor level, NA
## generated
```

- **table**: Another common way to store information is in a table.

```
# One way table
a <- factor(c("A","A","B","A","B","B","C","A","C"))
a
```

```
## [1] A A B A B B C A C
## Levels: A B C
```

```
a.table <- table(a)
a.table
```

```
## a
## A B C
## 4 3 2
```

```
attributes(a.table)
```

```
## $dim
## [1] 3
##
## $dimnames
## $dimnames$a
## [1] "A" "B" "C"
##
##
## $class
## [1] "table"
```

```
# Two way table
a <- c("Sometimes","Sometimes","Never","Always","Always","Sometimes","Sometimes","Never")
b <- c("Maybe","Maybe","Yes","Maybe","Maybe","No","Yes","No")
twoway.table <- table(a,b)
twoway.table
```

```
##            b
## a          Maybe No Yes
##   Always       2  0   0
##   Never        0  1   1
##   Sometimes    2  1   1
```

```
# An example
sexsmoke<-matrix(c(70,120,65,140),ncol=2,byrow=TRUE)
rownames(sexsmoke)<-c("male","female")
colnames(sexsmoke)<-c("smoke","nosmoke")
sexsmoke <- as.table(sexsmoke)
sexsmoke
```

```
##        smoke nosmoke
## male      70     120
## female    65     140
```

Control structures

Conditional excutions

- **equal**: ==
- **not equal**: !=
- **greater/less than**: >, <
- **greater/less than or equal**: >=, <=

Logical operators

- **and**: &, &&
- **or**: |, ||
- **not**: !

if-else statements

```
if (cond1==TRUE) {cmd1} else {cmd2}
```

```
# Example
if (1 == 0) {
   print(1)
} else {
   print(2)
}
```

```
## [1] 2
```

ifelse statements (ternary operator in R)

```
ifelse(test, true_value, false_value)
```

```
x <- 1:10
ifelse(x<5|x>8, x, 0)
```

```
##  [1]  1  2  3  4  0  0  0  0  9 10
```

switch-case statements

```
AA <- 'foo'
switch(AA,
       foo = {print('AA is foo')},
       bar = {print('AA is bar')},
       {print('Default')}
)
```

```
## [1] "AA is foo"
```

Loops

For loop

```
for (var in vector) {
   statement
}
```

```
# Example
mydf <- iris
head(mydf)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
myve <- NULL
for (i in 1:nrow(mydf)) {
   myve <- c(myve, mean(as.numeric(mydf[i, 1:3])))
}
myve
```

```
##   [1] 3.333333 3.100000 3.066667 3.066667 3.333333 3.666667 3.133333
##   [8] 3.300000 2.900000 3.166667 3.533333 3.266667 3.066667 2.800000
##  [15] 3.666667 3.866667 3.533333 3.333333 3.733333 3.466667 3.500000
##  [22] 3.433333 3.066667 3.366667 3.366667 3.200000 3.333333 3.400000
##  [29] 3.333333 3.166667 3.166667 3.433333 3.600000 3.700000 3.166667
##  [36] 3.133333 3.433333 3.300000 2.900000 3.333333 3.266667 2.700000
##  [43] 2.966667 3.366667 3.600000 3.066667 3.500000 3.066667 3.500000
##  [50] 3.233333 4.966667 4.700000 4.966667 3.933333 4.633333 4.333333
##  [57] 4.766667 3.533333 4.700000 3.933333 3.500000 4.366667 4.066667
##  [64] 4.566667 4.033333 4.733333 4.366667 4.200000 4.300000 4.000000
##  [71] 4.633333 4.300000 4.566667 4.533333 4.533333 4.666667 4.800000
##  [78] 4.900000 4.466667 3.933333 3.900000 3.866667 4.133333 4.600000
##  [85] 4.300000 4.633333 4.833333 4.333333 4.233333 4.000000 4.166667
##  [92] 4.566667 4.133333 3.533333 4.166667 4.300000 4.266667 4.466667
##  [99] 3.533333 4.200000 5.200000 4.533333 5.333333 4.933333 5.100000
## [106] 5.733333 3.966667 5.500000 5.000000 5.633333 4.933333 4.800000
## [113] 5.100000 4.400000 4.566667 4.966667 5.000000 6.066667 5.733333
## [120] 4.400000 5.266667 4.433333 5.733333 4.633333 5.233333 5.466667
## [127] 4.600000 4.666667 4.933333 5.333333 5.433333 6.033333 4.933333
## [134] 4.733333 4.766667 5.600000 5.100000 5.000000 4.600000 5.133333
## [141] 5.133333 5.033333 4.533333 5.300000 5.233333 4.966667 4.600000
## [148] 4.900000 5.000000 4.666667
```

while loop

```
while (condition) statements
```

```
# Example
z <- 0
while (z < 5) {
   z <- z + 2
   print(z)
}
```

```
## [1] 2
## [1] 4
## [1] 6
```

apply loop

For matrix/array

```
apply(X, MARGIN, FUN, ARGS)

# Examples
apply(iris[,1:3], 1, mean)

x <- 1:10

apply(as.matrix(x), 1, function(i) {
   if (i < 5)
      i - 1
   else
      i/i
})
```

For vector/list

```
lapply(X, FUN)
sapply(X, FUN)
```

```
# Examples
mylist <- as.list(iris[1:3, 1:3])
mylist
```

```
## $Sepal.Length
## [1] 5.1 4.9 4.7
##
## $Sepal.Width
## [1] 3.5 3.0 3.2
##
## $Petal.Length
## [1] 1.4 1.4 1.3
```

```
lapply(mylist, sum) # Compute sum of each list component and return result as list
```

```
## $Sepal.Length
## [1] 14.7
##
## $Sepal.Width
## [1] 9.7
##
## $Petal.Length
## [1] 4.1
```

```
sapply(mylist, sum) # Compute sum of each list component and return result as vector
```

```
## Sepal.Length  Sepal.Width Petal.Length
##         14.7          9.7          4.1
```

More apply functions

- **tapply**
- **mapply**

---

function

```
FunctionName <- function(arg1, arg2, ...) {
   statements
   return(R_object)
}
```

```
add <- function(a, b) {
   c <- a + b
   return(c)
}
x <- 5
y <- 7
z <- add(x,y)
z
```

```
## [1] 12
```

Advanced R programming

Garbage collection

- **rm()**
- **gc()**

```
x <- as.matrix(read.table("test.csv", sep="\t")) # x is a 4500000 x 220 matrix
y <- apply(x, 1, mean)
rm(list=c("x","y"))
gc()
```

Use data.table to speed up acquirement of data

See Introduction to the data.table package in R (http://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.pdf)

> Fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete of columns by group using no copies at all, list columns and a fast file reader (fread). Offers a natural and flexible syntax, for faster development. - from CRAN (http://cran.r-project.org/web/packages/data.table/index.html)

```
library(data.table)
grpsize <- ceiling(1e7/26^2)
DF <- data.frame(
   x=rep(LETTERS, each=26*grpsize),
   y=rep(letters, each=grpsize),
   v=runif(grpsize*26^2),
   stringsAsFactors=FALSE)
system.time(ans1 <- DF[DF$x=="R" & DF$y=="h",])
```

```
##    user  system elapsed
##   1.331   0.020   1.350
```

```
DT <- as.data.table(DF)
setkey(DT, x, y)
system.time(ans2 <- DT[list("R","h")])
```

```
##    user  system elapsed
##   0.005   0.000   0.005
```

Drawing graph

ggplot2 (http://ggplot2.org/)

> ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.

ggplot2 tutorial - from Edwin Chen (http://blog.echen.me/2012/01/17/quick-introduction-to-ggplot2/)'s Blog

Version control

> Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer.
> –from git (http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control)

Installing git

- **ubuntu users**: sudo apt-get install git
- **RedHat/Fedora/CentOS users**: sudo yum install git
- **Mac users**: http://git-scm.com/download/mac (http://git-scm.com/download/mac)
- **Windows users**: http://git-scm.com/download/win (http://git-scm.com/download/win)

Git setup

```
git config --global user.name "YOUR NAME"
git config --global user.email you.email@address.org
git config --global core.ui true
git config --global core.editor vim

# For windows users
git config --global core.quotepath off
```

Git basics

```
## Initializing a repository in an existing directory
# Go to the project's directory and type
git init

# Add files you want to track
git add LICENSE
git add READ.md
got commit -m 'First commit. Add LICENSE & READ.md'

# Add new files
git add R.Rmd
git add helloworld.r
git commit -m 'Second commit. Add R.Rmd, helloworld.r'
git remote add origin
git push -u origin master

# Recover your codes to the last commit
git checkout -- filename
git reset --hard


## Cloning an existing repository
git clone https://github.com/godkin1211/Rcourses.git
git pull https://github.com/godkin1211/Rcourses.git
```

References

- 猴子都能懂的git入門 (http://backlogtool.com/git-guide/cn/intro/intro1_1.html)
- ProGit (http://git-scm.com/book/en/v2)
- Git Reference (http://gitref.org/creating/)
- ProGit (http://git-scm.com/book/en/v2)