

## Chapter 3 Introduction to R

**Xiaogang Su**  
Department of Statistics  
University of Central Florida

### Getting Started with R

One way is editing all the R commands into a file first, which can be done with any word processor (preferably, WinEdt, check out the package **RWinEdt**), and then processing the whole file at once in R. For example, create a file with pico and write R codes in this file. Save the file, and then run it in an R session using the commands as follows.

```
> source(file="filename.s")
```

To save a R session, do the following.

```
> sink("output")  
...  
> sink()
```

This will save everything in between the two `sink` commands.

### Seeking Help

To get help on any command, say, `survfit`, for example, enter

```
> help(scan)  
> ?scan
```

To get help on a specific topic, start the help facility and then use mouse to explore the help windows.

To get help from other S-PLUS or R experts, S-news is a very active news group specially designated for discussing all kinds of S-PLUS or R related questions. To subscribe, send e-mail to [s-news-request@lists.biostat.wustl.edu](mailto:s-news-request@lists.biostat.wustl.edu) with the body of the message “subscribe s-news”.

The following website contains excellent help information of R, including several useful manuals, and frequently asked questions about R: <http://stat.ethz.ch/R-alpha/>

Here are the some useful documents on *R for SAS and SPSS Users* ([Word](#), [pdf](#)).

## Preliminaries

There are seven basic types of data objects in S-PLUS: vector, matrix, array, list, factor, time series, and data frame. And they could assume values of different modes including numerical, logical, character, complex etc.

To list all the objects in the current working directory

```
> ls()
# Or
> objects()
```

Note that everything behind # in a line will be treated as comments and not be evaluated.

**Creating Objects:** There are a lot of ways to create objects in SPLUS. For creating the most common object type, namely, vectors, some useful functions are listed below.

```
# Examples about two different uses of rep
> x <- rep(1:3, 3); x
[1] 1 2 3 1 2 3 1 2 3

> x <- rep(1:3, 1:3); x
[1] 1 2 2 3 3 3

# To see the help file on rep, use
> ?rep
```

**Table 1.1:** Useful Functions for Creating Vectors in R.

<b>Function</b>	<b>Description</b>	<b>Examples</b>
<b>scan</b>	read values in any mode	<b>scan(), scan("data")</b>
<b>c</b>	combines values in any mode	<b>c(1,3,2,6), c("yes","no")</b>
<b>rep</b>	repeat values in any mode	<b>rep(NA,5), rep(c(1,2),3)</b>
<b>:</b>	numeric sequences	<b>1:5, 1:-1</b>
<b>seq</b>	numeric sequences	<b>seq(-pi,pi,.5)</b>
<b>vector</b>	initialize vectors	<b>vector('complex',5)</b>
<b>logical</b>	initialize logical vectors	<b>logical(3)</b>
<b>numeric</b>	initialize numeric vectors	<b>numeric(4)</b>
<b>character</b>	initialize character vectors	<b>character(6)</b>

If you have a longer data set, you might want to use the scan function to enter your data. Below is an example of the scan function.

```
> x1 <- scan()
1: 1 4 2 8 5 7
7: 9 7 4 5 7 4
13:
```

The variable x1 will have 12 elements. Enter a blank line at the prompt to tell S+ that you are done entering data for the variable.

**Operating on Objects:** An operation on a subject could be getting its attributes, making corrections, subsetting or subscripting, or algebraic operation, merging variables and deleting.

```
> mode(x)
> length(x)

> mat <- rep(1:4, rep(3,4)); mat
> dim(mat) <- c(3,4); mat
```

### Make Corrections

```
> mat[2,3] <- 5.6; mat
```

If you have many changes to make, you can use the `fix` command.

```
> x <- fix(x)
```

This will allow you to edit all the contents of `mat`, using Notepad.

### Subsetting

```
> mat[2:3, 3:4]
> mat[mat[,2] <= 1.5, 1:3]
```

### Merging Variables by Rows or Columns

```
# cbind and rbind
> x1 <- rnorm(100, 1, 2.5)
> x2 <- sample(x=c(1,2,3), 100, replace=T)
> x3 <- cbind(x1, x2); x3
> x4 <- rbind(x1, x2); x4
```

If you have an object, say `x7`, that you no longer need, you can delete it using `rm` commands.

```
> rm(x7)
```

Same as in any other programming languages, how to avoid unnecessary loops is always an important concern. SPLUS has a lot of powerful built-in functions that could help things out if well used. The function `apply` is one of them. Basically, it returns a vector or array by applying a specified function to sections of an array. For example,

```
> x <- apply(matrix(rnorm(5000, 4, 3), 100, 50), MARGIN=2, mean)
```

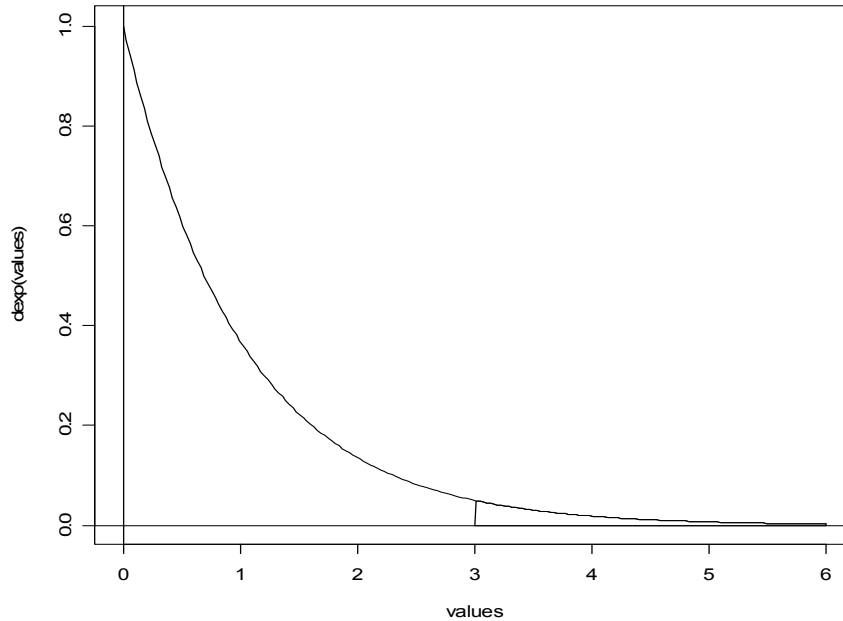
The option `MARGIN=2` tells SPLUS to compute means of `x` by columns instead of by rows. See what happen if `MARGIN = 1`. After all, a matrix is a two-dimensional array.

**Probability Distributions:** Splus had implemented almost all the common distributions. It is handy to get the density, probability, quantile and random numbers from a specific distribution in Splus.

```
# Example 1; Normal Distribution
> z <- qnorm(seq(.001, .999, len = 100), mean=2, sd=1 )
# compute a vector of quantiles
> y <- dnorm(z, mean=2, sd=1)
# density (dnorm), probability (pnorm), quantile (qnorm), or random
# sample (rnorm) for the Normal distribution with mean and sd.
> plot(z,y, type="l", ylab="Weibull Density")
```

```
# Example 2: Exponential
> values <- seq(0.0001, 6, length=200);
```

```
> bvals <- values[values>qexp(.95)]
> plot(values, dexp(values), type="l")
> polygon(c(qexp(.95), bvals, 6), c(0, dexp(bvals), 0))
> abline(h=0); abline(v=0)
```



R provides a comprehensive set of statistical tables. Functions are provided to evaluate the cumulative distribution function  $P(X \leq x)$ , the probability density function and the quantile function (given  $q$ , the smallest  $x$  such that  $P(X \leq x) > q$ ), and to simulate from the distribution.

Prefix the name given here by ‘d’ --- for the density;  
 ‘p’ --- for the CDF;  
 ‘q’ --- for the quantile function;  
 ‘r’ --- for simulation (random numbers).

<u>Distribution</u>	<u>R name</u>	<u>Additional arguments</u>
Binomial	binom	size, prob
Chi-squared	chisq	df, ncp
Exponential	exp	rate
F	f	df1, df2, ncp
Normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df, ncp
Uniform	unif	min, max

Here are some examples.

```
## 2-tailed p-value for t(13) distribution
# Suppose the observed value is 2.43
> 2*(1-pt(2.43, df = 13))
[1] 0.0303309

## The upper 1% point, i.e., the 99 percentile, for an F(2, 7)
distribution
> qf(0.99, df1 = 2, df2 = 7)
[1] 9.546578

# To generate 100 random numbers from N(0,2) distribution
> x <- rnorm(100, mean = 0, sd = 2); x
# by default, rnorm generates numbers from N(0, 1).

# to compute  $P(X=12)$ , where  $X \sim \text{Binomial}(20, .45)$ 
> pbinom(12, size = 20, prob = 0.45)
[1] 0.942
```

**Import and Export:** The scan function, which can read from either standard input or from a file, is commonly used to read data from keyboard input.

```
> x <- matrix(scan("filename"), ncol = 10, byrow = T)
```

If you have a text file with data arranged in the form of a table, you can read it into S-PLUS as a data frame using the read.table function.

```
> auto <- read.table('auto.dat', header=T)
```

When you want to export data to share with another S-PLUS user, use the data.dump or dump function:

```
> dump("matz", connection="matz.dmp")
```

To bring it back to SPLUS session, run

```
> source("matz.dmp")
```

The inverse operation to the scan function is provided by the cat and write functions. Similarly, the inverse operation to read.table is provided by write.table.

**Writing Functions in Splus:** Programming in S-PLUS consists largely of writing functions. Functions do most of the work in S-PLUS. The simplest functions arise naturally as shorthand for frequently-used combinations of S-PLUS expressions. For example, S-PLUS has no built-in function for calculating the standard deviation of a data set. It does, however, have a function for calculating the variance and another for calculating square roots. The standard deviation is simply the square root of the variance, so a standard deviation function can be created as follows:

```
> stdev <- function(x) { sqrt(var(x)) }
```

You can build more complicated functions either by adding new features incrementally onto simpler functions, or by designing whole programs from scratch. As your functions grow more complex, proper use of programming features such as conditionals and error handling becomes more important.

**Datasets in R:** Over fifty datasets are supplied with R, and others are available in packages (including the standard packages supplied with R). These datasets have to be loaded explicitly, using the function `data`. To see the list of datasets in the base system use `data()` and to load one of these use, for example,

```
> data(women)
> women
> attach(women, pos=1)
> mean(height)
> sd(height)
> hist(height)
> hist(height, nclass=10)
> stem(weight)
```

The decimal point is 1 digit(s) to the right of the |

```
11 | 57
12 | 0369
13 | 259
14 | 26
15 | 049
16 | 4
```

## Graphics in R

See the following two demos for some graphic facilities available in R.

```
demo(graphics)
demo(plotmath)

library(lattice)
demo(lattice)
demo(package = .packages(all.available = TRUE))
library(vcd)
demo(mosaic)
```

## Appendix A:

### Rattle: Gnome R Data Mining:

<http://rattle.togaware.com/>

#### Download

Please see the [Rattle User Guide](#) for a full guide to install. Make sure you have installed the GTK+ libraries for your OS (independent of R) or else you may see an error about `libatk` missing.

You can obtain a stable version of rattle from [CRAN](#):

```
> install.packages("RGtk2")
> install.packages("rattle")
```

The most current version can be obtained with the following two steps, obtaining Rattle from the Rattle repository and the RGtk2 package from the Gobi repository.

```
> install.packages("RGtk2", repos="http://www.ggobi.org/r/")
> install.packages("rattle", repos="http://rattle.togaware.com")
```

Otherwise download the Rattle source packages directly:

- GNU/Linux Package: [rattle\\_2.1.107.tar.gz](#)
- MS/Windows Package: [rattle\\_2.1.107.zip](#)

Finally, you can also download the actual Google Code repository with the command:

- `svn checkout http://rattle.googlecode.com/svn/trunk/ rattle`

Once downloaded, you simply need to:

```
> library("rattle")
> rattle()
```

And you should see the graphical user interface pop up.

**Appendix B:****R/S Cheatsheet**

Compiled by:

Barry W. Brown  
 Department of Biomathematics, Box 237  
 University of Texas M. D. Anderson Cancer Center  
 1515 Holcombe Blvd  
 Houston, TX 77030

bwb@mdaali.cancer.utexas.edu

## I. S EXPRESSIONS

- A. Literals  
 number                   1 1.1 1.1e10  
 string                   'string' or "string"  
 name  
 comment                   # string.  
 function (formals) expr  function(args){defn}
- B. Calls  
 expr infix expr  
 expr %anything% expr  
 unary expr  
 expr ( arglist )  
 expr [ arglist ]  
 expr [[ arglist ]]  
 expr \$ fname
- C. Assignment  
 expr <- expr  
 expr\_expr  
 expr -> expr  
 expr <<- expr                   Forces write to disk  
                                   from within a function
- D. Conditional  
 if ( expr ) expr  
 if ( expr ) expr else expr
- E. Iteration  
 repeat expr  
 while ( expr ) expr  
 for ( Name in expr ) expr
- F. Flow

```

break
next
return ( expr )
( expr )
{ exprlist }

```

## II. ARITHMETIC OPERATORS

```

*   Multiply
+   Add
-   Subtract
/   Divide
^   Exponentiation
%%  Remainder or modulo operator
**% Matrix multiplication operator
%/% Integer divide
%c% crossproduct           m1 %c% m2 is t(m1) **% m2
%o% Outer Product

```

## III. RELATIONAL OPERATORS

```

!=  Not-equal-to
<   Less-than
<=  Less-than-or-equal-to
==  Equal
>   Greater-than
>=  Greater-than-or-equal-to

```

## IV. LOGICAL OPERATORS

```

!   Not
|   Or (Use with arrays or matrices)

```

```

||   Shortcut Or (Don't use with arrays or matrices)
&   And (Use with arrays or matrices)
&&  Shortcut And (Don't use with arrays or matrices)

```

#### V. SUBSCRIPTS

```

[ ]   Vector subscript
[[ ]] list subscript - can only identify
      a single element
$     Named component selection from a list

```

#### V. SUBSCRIPT FORMS

```

logical           extracts or selects T component
positive numbers  extracts or selects specified indices
negative numbers  deletes specified indices
NA or out of range extends dimensions gives value NA

```

#### VI. SEQUENCE AND REPETITION

```

      seq (from, to, by, length, along)
also   :      as in 1:10

      rep(x, times, length)

```

#### VII. ARITHMETIC OPERATORS AND FUNCTIONS

```

abs(x)
acos(x)
acosh(x)
asin(x)
asinh(x)
atan(x)
atan(x, y)
atanh(x)
ceiling(x)
cos(x)
cosh(x)

```

```

exp(x)
floor(x)
gamma(x)
lgamma(x)
log(x, base=exp(1))
log10(x)
max(...)           elementwise
min(...)           elementwise
pmax(...)          parallel
pmin(...)          parallel
sin(x)
sinh(x)
sqrt(x)
tan(x)
tanh(x)
trunc(x)

```

## VIII. TYPES

Can be used in as. and is. and (length=n calls.

```

array
category           is, as only
character
complex
double
integer
list
logical
matrix
null               is, as only
numeric

```

## IX. IN AND OUT OF S

## A. Data In

```

scan(file="", what=numeric(), n, sep,
multi.line = F, flush = F, append = F)

```

```

Example:   data <-
matrix(scan("data.file"), ncol=5, byrow=T)

```

## B. Command File In

```

source(file, local = F)

```

## C. Screen Output to File

```

sink(file)
sink( )           restores output to screen

```

## D. Write and Read Objects

```

dput(x, file)

```

```
dget(file)

write(t(matrix),file,ncol=ncol(matrix),append=FALSE)

dump(list, fileout="dumpdata")
restore(file)
```

E. Make Things (Including Help) Available or Unavailable

```
assign("name", value, frame, where)

attach(file, pos=2)
detach(what=2)

library( )
library(help=section)

library(section, first=FALSE)
library.dynam(section, file)

help(name="help", offline=F)
args(name="help")
```

X. REDUCTION OPERATORS

```
all(...)
any(...)
length(x)
max(...)
mean(x, trim=0)
median(x)
min(...)
mode(x)
prod(...)
quantile(x, probs=c(0, .25, .5, .75, 1))
sum(...)
var(x, y)
cor(x, y, trim=0)
```

XI. STATISTICAL DISTRIBUTIONS

```
d(x,)          density at x
p(x,)          cumulative distn fn to x
q(p,)          inverse cdf
r(n,)          generates n random numbers from distn
```

```
+-----+
|+-----+|
```

	Distribution	Parameters	Defaults	
	beta	beta	shape1, shape2	-, -
	cauchy	Cauchy	loc, scale	0, 1
	chisq	chi-square	df	-
	exp	exponential	-	-
	f	F	df1, df2	-, -
	gamma	Gamma	shape	-
	lnorm	log-normal	mean, sd (of log)	0, 1
	logis	logistic	loc, scale	0, 1
	norm	normal	mean, sd	0, 1
	stab	stable	index, skew	-, 0
	t	Student's t	df	-
	unif	uniform	min, max	0, 1

## XII. PLOTTING

### A. Starting and Stopping Plotting

```
graphics.off()
```

### B. Device-Specification Functions

```
hp2623(ask=F, file="")
```

```
hpgl(width=10, height=7.25, ask=!auto,
auto=F, color=2, speed=400, rotated=F, file="")
```

```
postscript(file, command, horizontal=F, width,
height, rasters, pointsize=14, font=1,
preamble=ps.preamble, fonts=ps.fonts)
```

```
printer(width=80, height=64, file="", command="")
show()
```

```
tek4014(ask=F, file)
```

```
sun(ask=FALSE, color=FALSE)
```

### C. Some Plot Parameters

```
log=''          Logarithmic axes
```

```
main='title'
```

```
new=           T forces addition to current plot
```

```
sub='bottom title'
```

```
type=''       Line, points, both, none
```

```
lty=n         Line type
```

```
pch='.'      Plot character
```

```
xlab='x-axis label'
ylab='y-axis label'

xlim=c(xlo.value,xhi.value)
ylim=c(ylo.value,yhi.value)
```

## D. One-Dimension Plots

```
barplot(height)      #simple form
barplot(height, width, names, space=.2, inside=TRUE,
         beside=FALSE, horiz=FALSE, legend, angle,
         density, col, blocks=TRUE)

boxplot(..., range, width, varwidth=FALSE,
         notch=FALSE, names, plot=TRUE)

hist(x, nclass, breaks, plot=TRUE, angle,
     density, col, inside)
```

## E. Two-Dimension Plots

```
lines(x, y, type="l")
points(x, y, type="p")

matplot(x, y, type="p", lty=1:5, pch=, col=1:4)
matpoints(x, y, type="p", lty=1:5, pch=, col=1:4)
matlines(x, y, type="l", lty=1:5, pch=, col=1:4)

plot(x, y, type="p", log="")

abline(coef)
abline(a, b)
abline(reg)
abline(h=)
abline(v=)

qqplot(x, y, plot=TRUE)
qqnorm(x, datax=FALSE, plot=TRUE)
```

## F. Three-Dimension Plots

```
contour(x, y, z, v, nint=5, add=FALSE, labex)

interp(x, y, z, xo, yo, ncp=0, extrap=FALSE)

persp(z, eye=c(-6,-8,5), ar=1)
```

## G. Multiple Plots Per Page (Example)

```
par(mfrow=(nrow, ncol), oma=c(0, 0, 4, 0))
mtext(side=3, line=0, cex=2, outer=T,
      "This is an Overall Title For the Page")
```