

OLCF and NICS/RDAV Tutorial: Introduction and Basics of R*

George Ostrouchov^{†‡}, Pragnesh Patel[‡], and Drew Schmidt[‡]

June 28, 2012

Contents

1	Introduction	1
1.1	How to Run R	1
2	R Packages	1
3	Basics of R	2
3.1	Basic Objects and Data Types	2
3.2	A Super Calculator	3
3.3	Probability Distributions	3
3.4	Matrices, Arrays, Data Frames, and Factors	4
4	Functions, Logic, Loops, and the *ply Family	5
4.1	Functions	5
4.2	Logic	5
4.3	Loops	6
4.4	*ply	6
4.5	Exercises	7
5	Formula Methods	7
6	Input and Output of Data	8
7	Where to Learn More?	8
8	Suggested Solutions to Exercises	9
8.1	Section 4 Solutions	9

1 Introduction

R is a language and environment for data analysis and graphics. It is by far the most popular dialect of the S language, a domain specific language for interactive Statistical analysis of data. The development of S started at Bell Labs in the 1970s. R has grown wildly beyond its origins both in scope and popularity.

R syntax has a superficial similarity to C but it is a mix of functional and object oriented programming languages with roots in Scheme (a dialect of Lisp). The R language is interpreted so it is much slower than compiled languages but many of its functions are compiled components that run at full speed. Although R has flow control, it prefers vector operations and its simplest data type is a vector.

*Copyright © 2012 George Ostrouchov, Pragnesh Patel, and Drew Schmidt. All Rights Reserved.

[†]Oak Ridge National Laboratory, Computer Science and Mathematics Division, Scientific Data Group

[‡]National Institute for Computational Sciences, Remote Analysis and Visualization Center

1.1 How to Run R

There are several GUIs available for R but most people use R with interactive commands and with scripts. This can be done in several ways using your favorite text editor (emacs, aquamacs, vi, etc.) and some will even do R-specific color coding and run specified sections of code.

Recently, an integrated development environment, **RStudio**, became popular. Its advantage is that it has the same look on all platforms. It has both a console for interactive use of R and a script editor for a scripting capability. This can be downloaded from rstudio.org and installed on any platform. We will use **RStudio** for this tutorial.

If you have not downloaded and installed **RStudio** before this session, you can do it now from rstudio.org. It will only take a minute. Start **RStudio** once it is installed.

2 R Packages

One of R's greatest strengths is its package repository system. R has the Comprehensive R Archive Network (CRAN), which is analogous to CPAN for Perl and CTAN for L^AT_EX. Briefly, it is a central repository of R packages. The address of the CRAN is <http://cran.r-project.org/>. However, there are other repositories such as R-Forge <http://www.rforge.net/> and Bioconductor <http://bioconductor.org/>.

There are currently just under 4000 packages on the CRAN (most of them good), and the number of packages available on the CRAN is growing near exponentially¹.

Thankfully, interacting with the CRAN requires very minimal effort on the part of the R user. Generally one never needs to actually visit the CRAN website, but the CRAN taskview page, located at <http://cran.r-project.org/web/views/>, is an excellent resource for finding packages.

Now it is time to start **RStudio** and follow along with some commands in the console pane. The code in each gray box that follows is available in the `rscripts/file.r` named below the box. If you edit that file in **RStudio**, C⌘Enter in the editor will evaluate a single line in your console window. Changing your working directory to the tutorial directory `rscripts` on your system can be done from the **Tools** menu.

```
1 library()
2 library(lib.loc=.Library)
3 search()
4 searchpaths()
5
6 mirror <- "http://mirrors.nics.utk.edu/cran/"
7 options(repos = structure(c(CRAN = mirror)))
8 install.packages("ggplot2")
9
10 library("ggplot2")
11 library(help="ggplot2")
12 detach("ggplot2")
13 install.packages("rpart")
```

`rscripts/packages.r`

3 Basics of R

In an interactive R session, you create objects, which are kept in your workspace until you delete them. Most R commands are functions, including quitting R with `q()`. Upon quitting, your workspace is lost unless you saved

¹See K Hornik, *Are there Too Many R Packages?*, AUSTRIAN JOURNAL OF STATISTICS, Volume 41 (2012), Number 1, 59–66, <http://www.stat.tugraz.at/AJS/ausg121/121Hornik.pdf>

it. The function `q()` will ask you if you want to save it.

Workspace is your R session memory image.

Working Directory is the default directory to read and write any files. All path specifications are relative to this directory (forward slashes are used even on Windows).

3.1 Basic Objects and Data Types

Next we describe a subset of objects and data types in R in a somewhat simplified way. The full details are in the The R language definition manual².

Vector objects are *logical*, *integer*, *double*, *complex*, *character*, and *raw*, where all elements are of the same type. **Matrices** and higher dimensional **arrays** are vectors with dimension attributes. **Factors** are vectors with elements from a (small) finite set of values. In addition, there are a few **special values** that vector elements can have

`Inf` and `-Inf` are the result of, for example, `1/0` and `-1/0`

`NaN` is the result of, for example, `0/0`

`NA` is a symbol to indicate that the value is missing

List objects are “generic vectors” that can contain a mix of any kinds of objects. *Data frames* are “matrix-like” list objects, where each element has the same number of elements or rows.

Expression objects are R statements and they can be manipulated as any other objects

Function objects can also be manipulated as any other object in R. They have a formal argument list, a body, and an environment.

`NULL` is a special object used to indicate that an object is missing.

3.2 A Super Calculator

```
1 2 + 2
2 a <- 2 + 3
3 a
4 A
5 b <- c(1, 2, 3, 4, 5)
6 b
7 b <- 1:10
8 b
9 b - 5
10 1:100
11 sum(b)
12 mean(b)
13 sd(b)
14 summary(b)
15 b[5] <- 50
16 b
17 b[11] <- 15
18 b
19 mean(b)
20 b[15] <- 20
21 b
```

²R Development Core Team, *The R language definition*, <http://cran.r-project.org/doc/manuals/R-lang.html>

```

22 mean(b)
23
24 # how do we deal with NAs? Let's get some help!
25 ?mean
26 mean(b, na.rm=TRUE)
27 example(mean)
28 mean
29 methods(mean)
30 mean.default
31
32 # more general help
33 ?cluster
34 ??cluster

```

rscripts/calc.r

You can get help from the Help tab in RStudio but you can also get it from the Console whether you are using RStudio or not.

3.3 Probability Distributions

Having a statistical heritage, there are tools for hundreds of distributions available for modeling and testing.

```

1 ?distributions # click Task View at bottom
2 ?rlnorm
3
4 x<-rlnorm(1000) # skewed to the right
5 hist(x)
6 rug(x)
7 mean(x)
8 median(x)
9
10 ## Let's study how the trimmed mean behaves
11 trim <- seq(0, 0.5, 0.005)
12 trim
13 trimmed_mean <- sapply(trim, function(z) mean(x, trim=z))
14 plot(trim, trimmed_mean)

```

rscripts/dist.r

R and its packages are full of functions that make sense in estimation both mathematically and intuitively.

3.4 Matrices, Arrays, Data Frames, and Factors

A matrix or array is a vector with a dimension attribute. Its first index moves the fastest, like in FORTRAN. We can think of data frames as heterogeneous matrices although they are lists. They can be used with some matrix operations that make sense. We get a first look at factors here. These are extremely useful in analysis when non-numeric (categorical) data are present and have special methods throughout R and its packages.

```

1 x <- matrix(rpois(6, 3), ncol=2)
2 x
3 class(x)
4 typeof(x)
5 attributes(x)
6 attr(x, "dim")
7 dim(x)
8 t(x)
9 x %*% t(x)

```

```

10 z <- array(1:24, dim=c(2,3,4))
11 z
12 z[, , 3]
13 x
14 y <- data.frame(x)
15 y
16 class(y)
17 typeof(y)
18 attributes(y)
19 attr(y, "dim")
20 dim(y)
21 t(y)
22 y %*% t(y)
23 library(rpart) # Recursive partitioning and regression trees
24 data(car.test.frame) # CU car test data
25 class(car.test.frame)
26 typeof(car.test.frame)
27 attributes(car.test.frame)
28 names(car.test.frame)
29 car.test.frame$Price
30 class(car.test.frame$Price)
31 class(car.test.frame$Country)
32 class(car.test.frame$Type)
33 car.test.frame$Type
34 attributes(car.test.frame$Type)
35 str(car.test.frame$Type)
36 as.numeric(car.test.frame$Type)
37 plot(car.test.frame$Price)
38 plot(car.test.frame$Type)

```

rscripts/matrix.r

4 Functions, Logic, Loops, and the *ply Family

4.1 Functions

In general, functions can take arbitrary numbers of arguments and can output arbitrarily complex output structures.

```

1 f <- function(a, b){
2   return(a - b)
3 }
4
5 f(a=1, b=2)
6 f(1, 2)
7 f(b=1, a=2)
8 f(b=1, 2)
9 f(1)
10 f(matrix(1:4, ncol=2), matrix(4:1, nrow=2))
11
12 g <- function(a, b){
13   return(list(a+b, a-b, a*b, a/b))
14 }
15
16 g(5,2)

```

```
17 g(1,0)
18 g(f(2,6), 2)
```

rscripts/functions.r

4.2 Logic

Technically R does not have boolean logic, since it has TRUE, FALSE, and NA. Beyond this, logic and conditionals are fairly straightforward.

```
1 x <- 1
2 y <- -1
3 z <- NA
4 X <- c(x, y)
5 Y <- c(X, z)
6
7 x && y == 1
8 x || y == 1
9 x > 0
10 y <= -1
11 X > 0
12
13 all(X > 0)
14 any(X > 0)
15 is.na(Y)
16 any(is.na(Y))
17 which(is.na(Y))
18 Y[3]
19
20 if (x == 1) print("success") else print("failure")
21 if (y >= 0) print("success") else print("failure")
```

rscripts/logic.r

4.3 Loops

```
1 # Example 1: for loop
2 for (i in 1:5) print("Hello")
3
4 # Example 2: for loop
5 fib <- function(n){
6   phi <- (1 + sqrt(5))/2
7   return((phi^n - (-1/phi)^n)/sqrt(5))
8 }
9
10 fibonacci <- numeric(10)
11 for (i in 1:10) fibonacci[i] <- fib(i)
12 }
13
14 # Example 3: while loop
15 i <- 0
16 while (i < 5){
17   print("Hello")
18   i <- i+1
19 }
20
```

```

21 # Example 4: while loop
22 while(TRUE) print("Press ctrl+c to end this madness")
23
24 # Example 5: while loop to get all Fibonacci numbers below 100
25 flag <- 0
26 i <- 1
27 fiblist <- list()
28 while (flag==0){
29   fiblist[[i]] <- fib(i)
30   i <- i+1
31   if (fiblist[[i-1]] > 100) flag <- 1
32 }
33 fiblist[[length(fiblist)]] <- NULL
34 fiblist

```

rscripts/loops.r

4.4 *ply

The *ply family of functions such as `apply()`, `lapply()`, `sapply()`, `tapply()`, and `vapply()` are loop-like functions that can offer enhanced readability of code, as well as even (sometimes) offering performance advantages over using loops.

```

1 # sorting columns of a matrix
2 x <- cbind(x1 = 2:1, x2 = c(3:1, 5:7))
3 x
4
5 for (i in 1:2){
6   x[,i] <- sort(x[,i])
7 }
8 x
9
10 x <- cbind(x1 = 2:1, x2 = c(3:1, 5:7))
11 apply(X=x, MARGIN=2, FUN=sort)
12 apply(X=x, MARGIN=1, FUN=sort)
13 apply(X=x, MARGIN=3, FUN=sort)
14
15 # some other plys
16 lapply(X=rep("Hello", 5), FUN=identity)
17 sapply(X=rep("Hello", 5), FUN=identity)
18 sapply(X=rep("Hello", 5), FUN=identity, simplify=FALSE)
19 sapply(X=rep("Hello", 5), FUN=identity, USE.NAMES=FALSE)
20
21 lapply(X=1:10, FUN=fib)
22 sapply(X=1:10, FUN=fib)
23
24 # sapply can be useful for subsetting lists
25 x <- list(1:3, 4:6, 7:9)
26 x
27 sapply(X=x, FUN="[", FUN.VALUE=2)
28
29 # tapply
30 ind <- list(c(1, 2, 2), c("A", "A", "B"))
31 table(ind)
32 tapply(X=1:3, INDEX=ind, FUN=sum)

```

4.5 Exercises

1. Create a function `f` that will take an integer `n` print "success" if the number is divisible by 5, and print "failure" otherwise. In R, `%%` is the modulus operator, with `a%%b` returning the remainder of the division of `a` divided into `b`. So `10%%5` returns 0, `11%%5` returns 1, `12%%5` returns 2, etc.
2. Use a for loop to evaluate the function `f` you just developed on each of the numbers in the vector `1:20`. Do the same with `sapply`.

5 Formula Methods

Much of data analysis involves the repeated selection of variables and their functions and interactions for models. Because this is so common, R has the powerful concept of a "formula" that allows shorthand specification and manipulation of what variables to include in an analysis and in what way. Typically, the formula method is used with a data frame, where the variables are its columns.

```

1 library(rpart)
2 install.packages("rpart.plot")
3 library(rpart.plot)
4 data(car.test.frame)
5 summary(car.test.frame)
6 fit <- rpart(Mileage ~ Country + Reliability + Price + Type +
  Weight + Disp. + HP, data=car.test.frame)
7 rpart.plot(fit)
8 rsq.rpart(fit)
9 vars <- names(car.test.frame)
10 vars
11 model <- as.formula(paste(vars[3], paste(vars[-3], collapse="
  + "), sep=" ~ "))
12 model
13 fit <- rpart(model, data=car.test.frame)
14 rpart.plot(fit)
15 model <- as.formula(paste(vars[2], paste(vars[-2], collapse="
  + "), sep=" ~ "))

```

rscripts/formula.r

6 Input and Output of Data

As you have seen, R has many integrated data sets. This is useful for teaching R and for playing with R but it doesn't get your data analyzed. Base R has facilities for reading ASCII files and there are several packages for reading files produced by other software such as Excel, SAS, SPSS, Octave, NetCDF, HDF5, and many others. The [R Data Import/Export](#) manual is an excellent initial source on what R packages are relevant to what data format. Because of so many different formats available, we only briefly cover ASCII data input and output here and refer users needing other format information to the [R Data Import/Export](#) manual.

```

1 car.test.frame
2 save(car.test.frame, file="Rcardata")
3 rm(car.test.frame)
4 ls()
5 load("Rcardata")
6 ?write.table
7 write.table(car.test.frame, file="cardata.txt")

```



```

8 ?read.table
9 cardata <- read.table("cardata.txt", header=TRUE)
10 all(cardata == car.test.frame)
11 all(cardata == car.test.frame, na.rm=TRUE)
12 all(is.na(cardata) == is.na(car.test.frame))
13
14 ?scan

```

rscripts/io.r

7 Where to Learn More?

The r-project.org web site under the Manuals link provides a good place to start. The Books link now lists over 100 books. There are numerous other web sites that provide a wealth of information on various R topics. Some of these follow.

Cookbook for R: <http://wiki.stdout.org/rcookbook/>

R Graph Gallery: <http://addictedtor.free.fr/graphiques/>

The R Journal: <http://journal.r-project.org/>

R Mailing Lists: <http://www.r-project.org/mail.html>

The R Wiki: <http://rwiki.sciviews.org>

8 Suggested Solutions to Exercises

8.1 Section 4 Solutions

1.

```

1 f <- function(n){
2   if (n %% 5 == 0) print("success") else print("failure")
3 }

```

2.

```

1 # loop
2 for (i in 1:20){
3   f(i)
4 }
5
6 # sapply
7 sapply(X=1:20, FUN=f)

```