Lecture 5, R Basics

MATH-7260 Linear Models

Dr. Xiang Ji @ Tulane University

January 27, 2025

Contents

Announcement

- Just stay warm.

R basics

styles

(reading assignment)

Checkout Style guide in Advanced R and the tidyverse style guide.

Arithmetic

R can do any basic mathematical computations.

symbol	use
+	addition
-	subtraction
*	multiplication
/	division
^	power

symbol	use
%%	modulus
exp()	exponent
log()	natural logarithm
sqrt()	square root
round()	rounding
floor()	flooring
ceiling()	ceiling

Objects

You can create an R object to save results of a computation or other command.

Example 1

```
x <- 3 + 5
x
```

```
## [1] 8
```

- In most languages, the direction of passing through the value into the object goes from right to left (e.g. with "="). However, R allows both directions (which is actually bad!). In this course, we encourage the use of "<-" or "=". There are people liking "=" over "<-" for the reason that "<-" sometimes break into two operators "< -".

Example 2

```
x < - 3 + 5
```

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

```
x
```

```
## [1] 8
```

- For naming conventions, stick with either "." or "_" (refer to the style guide).

Example 3

```
sum.result <- x + 5
sum.result
```

```
## [1] 13
```

- *important*: many names are already taken for built-in R functions. Make sure that you don't override them.

Example 4

```
sum(2:5)
```

```
## [1] 14
```

```
sum
```

```
## function (..., na.rm = FALSE)  .Primitive("sum")
```

```
sum <- 3 + 4 + 5
sum(5:8)
```

```
## [1] 26
```

```
sum
```

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

- R is case-sensitive. “Math.7260” is different from “math.7260”.

Locating and deleting objects:

The commands “objects()” and “ls()” will provide a list of every object that you’ve created in a session.

```
objects()
```

```
## [1] "sum"        "sum.result" "x"
```

```
ls()
```

```
## [1] "sum"        "sum.result" "x"
```

The “rm()” and “remove()” commands let you delete objects (tip: always clearn-up your workspace as the first command)

```
rm(list=ls())  # clean up workspace
```

Vectors

Many commands in R generate a vector of output, rather than a single number.

The “c()” command: creates a vector containing a list of specific elements.

Example 1

```
c(7, 3, 6, 0)
```

```
## [1] 7 3 6 0
```

```
c(73:60)
```

```
##  [1] 73 72 71 70 69 68 67 66 65 64 63 62 61 60
```

```
c(7:3, 6:0)
```

```
##  [1] 7 6 5 4 3 6 5 4 3 2 1 0
```

```
c(rep(7:3, 6), 0)
```

```
##  [1] 7 6 5 4 3 7 6 5 4 3 7 6 5 4 3 7 6 5 4 3 7 6 5 4 3 0
```

Example 2 The command “seq()” creates a sequence of numbers.

```
seq(7)
```

```
## [1] 1 2 3 4 5 6 7
```

```
seq(3, 70, by = 6)
```

```
##  [1]  3  9 15 21 27 33 39 45 51 57 63 69
```

```
seq(3, 70, length = 6)
```

```
## [1]  3.0 16.4 29.8 43.2 56.6 70.0
```

Operations on vectors

Use brackets to select element of a vector.

```
x <- 73:60
x[2]
```

```
## [1] 72
```

```r
x[2:5]
```

```
## [1] 72 71 70 69
```

```r
x[-(2:5)]
```

```
##  [1] 73 68 67 66 65 64 63 62 61 60
```

Can access by "name" (safe with column/row order changes)

```r
y <- 1:3
names(y) <- c("do", "re", "mi")
y[3]
```

```
## mi
##  3
```

```r
y["mi"]
```

```
## mi
##  3
```

R commands on vectors

command	usage
sum()	sum over elements in vector
mean()	compute average value
sort()	sort elements in a vector
min(), max()	min and max values of a vector
length()	length of a vector
summary()	returns the min, Q1, median, mean, Q3, and max values of a vector
sample(x, size, replace = FALSE, prob = NULL)	takes a random sample from a vector with or without replacement

Exercise Write a command to generate a random permutation of the numbers between 1 and 5 and save it to an object.

Matrix

matrix() command creates a matrix from the given set of values

```r
matrix.example <- matrix(rnorm(100), nrow = 10, ncol = 10, byrow = TRUE)
matrix.example
```

```
##             [,1]       [,2]       [,3]        [,4]       [,5]        [,6]
##  [1,]  0.3666982 -1.7669433  1.7641834 -0.61621887  1.4863224 -0.94479077
##  [2,] -0.9857926 -1.2483619 -0.1926700  0.15690203  0.5847165 -1.93870114
##  [3,]  1.8374926 -0.1019404  0.3921290  1.66280129 -0.1326306  1.02612476
##  [4,] -0.1215171  1.7519705  1.7286263  0.56500359  0.0752892  2.00487245
##  [5,]  0.3809785 -1.1764969 -1.3117362 -1.77008120  0.4225604  0.38768775
##  [6,] -0.8081790  0.3031515  0.2896139  1.29129656  1.7642979 -0.01015978
##  [7,] -0.8699963  1.0416742 -0.2101917 -1.21447486 -1.7583827  0.78481029
##  [8,]  0.3492486 -1.5457233  0.9106507  0.07285106 -1.4851710  0.09121639
##  [9,] -0.6137160  0.2900285  2.1979912 -1.38086049 -1.5609317  0.35626883
## [10,]  1.9593308 -2.1541732  0.1223442  1.71848814 -1.0178962 -1.81570882
##             [,7]        [,8]       [,9]       [,10]
```

```
##  [1,] -1.1364874  2.18514605  0.7735370 -0.03889042
##  [2,] -0.3107539  0.95673779 -0.2807409 -0.16059619
##  [3,] -0.1206051 -0.23887993  2.2026695 -0.79009467
##  [4,]  0.2567208 -0.44896466 -1.1802590  0.82425547
##  [5,]  1.1109844 -0.05011302 -0.2438188 -0.19231284
##  [6,] -0.2056786 -0.01680366 -1.7923752 -0.92163776
##  [7,] -0.4687571  0.80052581  1.8038154 -0.24952519
##  [8,]  0.1769619 -1.24786462  1.5204527 -1.98957849
##  [9,]  0.6492282  0.84311980 -0.6476299 -0.49847309
## [10,] -1.5319584  0.40464881  0.5350702 -0.17699616
```

R commands on vector/matrix

command	usage
sum()	sum over elements in vector/matrix
mean()	compute average value
sort()	sort all elements in a vector/matrix
min(), max()	min and max values of a vector/matrix
length()	length of a vector/matrix
summary()	returns the min, Q1, median, mean, Q3, and max values of a vector
dim()	dimension of a matrix
cbind()	combine a sequence of vector, matrix or data-frame arguments and combine by columns
rbind()	combine a sequence of vector, matrix or data-frame arguments and combine by rows
names()	get or set names of an object
colnames()	get or set column names of a matrix-like object
rownames()	get or set row names of a matrix-like object

```
sum(matrix.example)
```

```
## [1] 1.488254
```

```
mean(matrix.example)
```

```
## [1] 0.01488254
```

```
sort(matrix.example)
```

```
##   [1] -2.15417318 -1.98957849 -1.93870114 -1.81570882 -1.79237520 -1.77008120
##   [7] -1.76694327 -1.75838269 -1.56093167 -1.54572329 -1.53195838 -1.48517104
##  [13] -1.38086049 -1.31173623 -1.24836192 -1.24786462 -1.21447486 -1.18025905
##  [19] -1.17649695 -1.13648736 -1.01789623 -0.98579258 -0.94479077 -0.92163776
##  [25] -0.86999626 -0.80817903 -0.79009467 -0.64762991 -0.61621887 -0.61371604
##  [31] -0.49847309 -0.46875710 -0.44896466 -0.31075391 -0.28074086 -0.24952519
##  [37] -0.24381878 -0.23887993 -0.21019167 -0.20567863 -0.19267004 -0.19231284
##  [43] -0.17699616 -0.16059619 -0.13263062 -0.12151707 -0.12060510 -0.10194035
##  [49] -0.05011302 -0.03889042 -0.01680366 -0.01015978  0.07285106  0.07528920
##  [55]  0.09121639  0.12234418  0.15690203  0.17696188  0.25672075  0.28961388
##  [61]  0.29002852  0.30315153  0.34924861  0.35626883  0.36669820  0.38097853
##  [67]  0.38768775  0.39212899  0.40464881  0.42256040  0.53507022  0.56500359
```

```
##  [73]  0.58471649  0.64922817  0.77353698  0.78481029  0.80052581  0.82425547
##  [79]  0.84311980  0.91065069  0.95673779  1.02612476  1.04167419  1.11098445
##  [85]  1.29129656  1.48632237  1.52045266  1.66280129  1.71848814  1.72862626
##  [91]  1.75197050  1.76418343  1.76429791  1.80381544  1.83749262  1.95933084
##  [97]  2.00487245  2.18514605  2.19799120  2.20266949
```

```
summary(matrix.example)
```

```
##        V1                 V2                 V3                 V4
##  Min.   :-0.9858   Min.   :-2.1542   Min.   :-1.3117   Min.   :-1.77008
##  1st Qu.:-0.7596   1st Qu.:-1.4714   1st Qu.:-0.1139   1st Qu.:-1.06491
##  Median : 0.1139   Median :-0.6392   Median : 0.3409   Median : 0.11488
##  Mean   : 0.1495   Mean   :-0.4607   Mean   : 0.5691   Mean   : 0.04857
##  3rd Qu.: 0.3774   3rd Qu.: 0.2999   3rd Qu.: 1.5241   3rd Qu.: 1.10972
##  Max.   : 1.9593   Max.   : 1.7520   Max.   : 2.1980   Max.   : 1.71849
##        V5                  V6                   V7                 V8
##  Min.   :-1.75838   Min.   :-1.938701   Min.   :-1.5320   Min.   :-1.2479
##  1st Qu.:-1.36835   1st Qu.:-0.711133   1st Qu.:-0.4293   1st Qu.:-0.1917
##  Median :-0.02867   Median : 0.223743   Median :-0.1631   Median : 0.1939
##  Mean   :-0.16218   Mean   :-0.005838   Mean   :-0.1580   Mean   : 0.3188
##  3rd Qu.: 0.54418   3rd Qu.: 0.685530   3rd Qu.: 0.2368   3rd Qu.: 0.8325
##  Max.   : 1.76430   Max.   : 2.004872   Max.   : 1.1110   Max.   : 2.1851
##        V9                V10
##  Min.   :-1.7924   Min.   :-1.9896
##  1st Qu.:-0.5559   1st Qu.:-0.7172
##  Median : 0.1456   Median :-0.2209
##  Mean   : 0.2691   Mean   :-0.4194
##  3rd Qu.: 1.3337   3rd Qu.:-0.1647
##  Max.   : 2.2027   Max.   : 0.8243
```

Exercise Write a command to generate a random permutation of the numbers between 1 and 5 and save it to an object.

Comparison (logic operator)

symbol	use
!=	not equal
==	equal
>	greater
>=	greater or equal
<	smaller
<=	smaller or equal
is.na	is it "Not Available"/Missing
complete.cases	returns a logical vector specifying which observations/rows have no missing values
is.finite	if the value is finite
all	are all values in a logical vector true?
any	any value in a logical vector is true?

```
test.vec <- 73:68
test.vec
```

```
## [1] 73 72 71 70 69 68
```

```
test.vec < 70
```

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

```
test.vec > 70
```

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

```
test.vec[3] <- NA
test.vec
```

```
## [1] 73 72 NA 70 69 68
```

```
is.na(test.vec)
```

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

```
complete.cases(test.vec)
```

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

```
all(is.na(test.vec))
```

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

```
any(is.na(test.vec))
```

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

Now let's do a test of accuracy for doubles in R. Recall that for *Double* precision, we get approximately $\log_{10}(2^{52}) \approx 16$ decimal point for precision.

```
test.exponent <- -(7:18)
10^test.exponent == 0
```

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

```
1 - 10^test.exponent == 1
```

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

```
7360 - 10^test.exponent == 7360
```

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

```
73600 - 10^test.exponent == 73600
```

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

Other operators

%in%, match

```
test.vec
```

```
## [1] 73 72 NA 70 69 68
```

```
66 %in% test.vec
```

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

```
match(66, test.vec, nomatch = 0)
```

```
## [1] 0
```

```r
70 %in% test.vec
```

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

```r
match(70, test.vec, nomatch = 0)
```

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

```r
match(70, test.vec, nomatch = 0) > 0 # the implementation of %in%
```

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

Control flow

These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like (Algol short for "Algorithmic Language") language. They are all reserved words.

keyword	usage
if	**if**(*cond*) *expr*
if-else	**if**(*cond*) *cons.expr* **else** *alt.expr*
for	**for**(*var* **in** *seq*) *expr*
while	**while**(*cond*) *expr*
break	breaks out of a *for* loop
next	halts the processing of the current iteration and advances the looping index

Define a function

Read Function section from Advanced R by Hadley Wickham. We will visit functions in more details.

```r
DoNothing <- function() {
  return(invisible(NULL))
}
DoNothing()
```

In general, try to avoid using loops (vectorize your code) in R. If you have to loop, try using **for** loops first. Sometimes, **while** loops can be dangerous (however, a smart *compiler* should detect this).

```r
DoBadThing <- function() {
  result <- NULL
  while(TRUE) {
    result <- c(result, rnorm(100))
  }
  return(result)
}
# DoBadThing()
```

Install packages

You can install R packages from several places (reference):

- Comprehensive R Archive Network (CRAN)
  - Official R packages repository
  - Some levels of code checks (cross platform support, version control etc)
  - Most common place you will install packages

  - Pick a mirror location near you
  - `install.packages("packge_name")`
- GitHub
  - May get development version of a package
  - Almost zero level of code checks
  - Common place to develop a package before submitting to CRAN

```
install.packages("devtools")
library(devtools)
install_github("tidyverse/ggplot2")
```

Load packages

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
require(tidyverse)
```