R_APPENDIX-B_MicroeconometricsR

April 16, 2021

[2]:
```
# B2. Objects in R

# B2.1. Vectors

a <- c(1,2,3)
b <- c("one", "two", "three")

a

b
```

1. 1 2. 2 3. 3

1. 'one' 2. 'two' 3. 'three'

[4]:
```
d <- c(a,b)

d
```

1. '1' 2. '2' 3. '3' 4. 'one' 5. 'two' 6. 'three'

[5]:
```
b <- c(4,5,6)
a+b
```

1. 5 2. 7 3. 9

[6]:
```
a*b
```

1. 4 2. 10 3. 18

[7]:
```
a/b
```

1. 0.25 2. 0.4 3. 0.5

[8]:
```
a + 2
```

1. 3 2. 4 3. 5

[9]:
```
a*2
```

1. 2 2. 4 3. 6

[10]:
```
a/2
```

1. 0.5 2. 1 3. 1.5

[11]:
```
b <- c(4,5)
a+b

# Important warning that R still do the operation regardless length!
```

```
Warning message in a + b:
"longer object length is not a multiple of shorter object length"
```

1. 5 2. 7 3. 7

[12]:
```
a*b

# Important warning that R still do the operation regardless length!
```

```
Warning message in a * b:
"longer object length is not a multiple of shorter object length"
```

1. 4 2. 10 3. 12

[14]:
```
# B2.2. Matrices

a <- c(1,2,3)
b <- c(4,5,6)
A <- cbind(a,b)
B <- rbind(a,b)

A

is.matrix(A)

B

is.matrix(B)
```

a	b
1	4
2	5
3	6

TRUE

a	1	2	3
b	4	5	6

TRUE

[23]:
```
t(A)
```

```
t(B)
```

a	1	2	3
b	4	5	6

a	b
1	4
2	5
3	6

[19]:
```
C <- A + 2
C
A + C  # cell-by-cell operations
```

a	b
3	6
4	7
5	8

a	b
4	10
6	12
8	14

[20]:
```
D <- B*2
D
B*D  # cell-by-cell operations
```

a	2	4	6
b	8	10	12

a	2	8	18
b	32	50	72

[21]:
```
A^2    # cell-by-cell operations
```

a	b
1	16
4	25
9	36

[22]:
```
A + t(B)
```

a	b
2	8
4	10
6	12

[24]:
```
# standard matrix multiplication
```

```
A%*%B  # (3x2)*(2x3)
```

17	22	27
22	29	36
27	36	45

[25]:
```
# B2.3. Lists

a_list <- list(a,b)
a_list
```

1. (a) 1 (b) 2 (c) 3
2. (a) 4 (b) 5 (c) 6

[26]:
```
b_list <- list(a_list,A)
b_list
```

1. (a) i. 1 ii. 2 iii. 3

   (b) i. 4 ii. 5 iii. 6
2.

a	b
1	4
2	5
3	6

[27]:
```
c_list <- list(A,B)
c_list
```

1.

a	b
1	4
2	5
3	6

2.

a	1	2	3
b	4	5	6

[28]:
```
c_list <- c(c("one","two"),c_list)
c_list

# operations of function c() looks similar to list(), but it is very much␣
 ↪different
```

1. 'one'
2. 'two'
3.

a	b
1	4
2	5
3	6

4.

a	1	2	3
b	4	5	6

[30]:
```
# B3 Interacting with Objects

# B3.1. Transforming Objects

B
as.vector(B)
```

a	1	2	3
b	4	5	6

1. 1 2. 4 3. 2 4. 5 5. 3 6. 6

[32]:
```
a
b
c(a,b)
matrix(c(a,b),nrow=3)
```

1. 1 2. 2 3. 3

1. 4 2. 5 3. 6

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6

1	4
2	5
3	6

[34]:
```
cbind(a,b)
as.matrix(cbind(a,b))
```

a	b
1	4
2	5
3	6

a	b
1	4
2	5
3	6

[36]:
```
B
as.list(B)
```

a	1	2	3
b	4	5	6

1. 1
2. 4
3. 2

4. 5
5. 3
6. 6

[37]:
```
a_list
unlist(a_list)
```

1. (a) 1 (b) 2 (c) 3
2. (a) 4 (b) 5 (c) 6

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6

[38]:
```
A
as.character(A)
```

a	b
1	4
2	5
3	6

1. '1' 2. '2' 3. '3' 4. '4' 5. '5' 6. '6'

[39]:
```
B
as.factor(B)
```

a	1	2	3
b	4	5	6

1. 1 2. 4 3. 2 4. 5 5. 3 6. 6

Levels: 1. '1' 2. '2' 3. '3' 4. '4' 5. '5' 6. '6'

[41]:
```
a
as.vector(as.numeric(as.character(as.factor(a)))) == a
```

1. 1 2. 2 3. 3

1. TRUE 2. TRUE 3. TRUE

[42]:
```
# B3.2. Logical Expressions

a
b
a == b
```

1. 1 2. 2 3. 3

1. 4 2. 5 3. 6

1. FALSE 2. FALSE 3. FALSE

[43]:
```
A
B
A == t(B)
```

a	b
1	4
2	5
3	6

a	1	2	3
b	4	5	6

a	b
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE

[45]:
```
a_list
a_list[[1]]
a_list[[2]]
a_list[[1]] == a_list[[2]]
```

1. (a) 1 (b) 2 (c) 3
2. (a) 4 (b) 5 (c) 6

1. 1 2. 2 3. 3

1. 4 2. 5 3. 6

1. FALSE 2. FALSE 3. FALSE

[47]:
```
a
b

a > b
```

1. 1 2. 2 3. 3

1. 4 2. 5 3. 6

1. FALSE 2. FALSE 3. FALSE

[48]:
```
b > 5
```

1. FALSE 2. FALSE 3. TRUE

[49]:
```
b >= 5
```

1. FALSE 2. TRUE 3. TRUE

[50]:
```
b <= 4
```

1. TRUE 2. FALSE 3. FALSE

[51]:
```
a != b
```

1. TRUE 2. TRUE 3. TRUE

[52]:
```
(b > 4) & a == 3
```

1. FALSE 2. FALSE 3. TRUE

[54]:
```
(b > 4) && a == 3    # && asks whether all the element is the same
```

FALSE

[55]:
```
(b > 4) | a == 3
```

1. FALSE 2. TRUE 3. TRUE

[56]:
```
(b > 4) || a == 3   # || asks whether all the element is the same
```

FALSE

[57]:
```
# B3.3. Retrieving Information from a Position

a
a[1]
```

1. 1 2. 2 3. 3

1

[58]:
```
b
b[3]
```

1. 4 2. 5 3. 6

6

[59]:
```
A
A[5]
```

a	b
1	4
2	5
3	6

5

[61]:
```
A
A[2,2]
```

a	b
1	4
2	5
3	6

b: 5

[62]:
```
a
a[1:2]
```

1. 1 2. 2 3. 3

1. 1 2. 2

[63]:
```
a[-3]
```

1. 1 2. 2

[64]:
```
A
A[1,]   # take row 1, for all column
```

a	b
1	4
2	5
3	6

a 1 **b** 4

[65]:
```
A
A[c(1,5)]
```

a	b
1	4
2	5
3	6

1. 1 2. 5

[67]:
```
a
length(a)
a[2:length(a)]
```

1. 1 2. 2 3. 3

3

1. 2 2. 3

[68]:
```
A
D <- cbind(A, 2*A)
D
```

a	b
1	4
2	5
3	6

a	b	a	b
1	4	2	8
2	5	4	10
3	6	6	12

[69]:
```
D
dim(D)            # dimension: 3x4
dim(D)[2]         # take the column as the 'position', that is 4
D[,3:dim(D)[2]]   # take all row for each column starting from 3 to 4 (dim(D)[2])
```

a	b	a	b
1	4	2	8
2	5	4	10
3	6	6	12

1. 3 2. 4

4

a	b
2	8
4	10
6	12

[75]:
```
a_list
a_list[2]

a_list[2][2]   # no list, since a_list[2] just consist of one list. there is no␣
 ↪2nd list in this particular a_list[2] list


a_list[[2]]    # taking what is inside the second list from a_list. resulting in␣
 ↪a vector-like object

a_list[[2]][2] # first, taking what is inside the second list from a_list. then,␣
 ↪take the second element from it
```

1. (a) 1 (b) 2 (c) 3
2. (a) 4 (b) 5 (c) 6

1. (a) 4 (b) 5 (c) 6

1. NULL

1. 4 2. 5 3. 6

5

[78]:
```
a_list

names(a_list) <- c("first","second")
```

```
names(a_list)

a_list$first    # $ retrieves a named item in a list

names(a_list)[2]
```

$first 1. 1 2. 2 3. 3

$second 1. 4 2. 5 3. 6

1. 'first' 2. 'second'

1. 1 2. 2 3. 3

'second'

[82]:
```
a_list
B

b_list <- list(a_list,B=B)

b_list

b_list$B
```

$first 1. 1 2. 2 3. 3

$second 1. 4 2. 5 3. 6

a	1	2	3
b	4	5	6

[[1]] **$first** 1. 1 2. 2 3. 3

$second 1. 4 2. 5 3. 6

$B

a	1	2	3
b	4	5	6

a	1	2	3
b	4	5	6

[83]:
```
# B3.4 Retrieving the Position from the Information

a

which(a > 2)
```

1. 1 2. 2 3. 3

3

[84]:
```
A
```

```
which(A > 2)              # it retrieves the position in the matrix A that is␣
 ↪higher than 2
                          # in our case, (3,1), (1,2), (2,2), and (3,2) are␣
 ↪indeed larger than 2
                          # (3,1) == 3, (1,2) == 4, (2,2) == 5, (3,2) == 6
```

a	b
1	4
2	5
3	6

1. 3 2. 4 3. 5 4. 6

[88]:
```
a_list

a_list[[2]]

which(a_list[[2]] > 2)      # it retrieves the position in the vector a_list that␣
 ↪is higher than 2.
                            # in our case, elements 1,2, and 3 are indeed larger␣
 ↪than 2
```

$first 1. 1 2. 2 3. 3

$second 1. 4 2. 5 3. 6

1. 4 2. 5 3. 6

1. 1 2. 2 3. 3

[105]:
```
b

a > 2

b[a > 2]    # when a is greater than 2? it is in position/index 3, then take␣
 ↪element 3. we have b[3]. b[3] returns 6
```

1. 4 2. 5 3. 6

1. FALSE 2. FALSE 3. TRUE

6

[109]:
```
B

A > 2

B[3]

B[A > 2]  # when A is greater than 2? it is in position/index 3,4,5,6 (start for␣
 ↪each column, compute position for all row).
```

```
          # Then, it returns 2,5,3,6
```

a	1	2	3
b	4	5	6

a	b
FALSE	TRUE
FALSE	TRUE
TRUE	TRUE

2

1. 2 2. 5 3. 3 4. 6

[111]:
```
A

colnames(A)

A[, colnames(A)=="b"]    # retrieves every row for which the column name is "b".␣
 ↪then you have a vector c(4,5,6)
```

a	b
1	4
2	5
3	6

1. 'a' 2. 'b'

1. 4 2. 5 3. 6

[113]:
```
# on match() and %in%

d <- c("one","two","three","four")
c("one","four","five") %in% d

a

match(a,c(1,2))
```

1. TRUE 2. TRUE 3. FALSE

1. 1 2. 2 3. 3

1. 1 2. 2 3. <NA>

[116]:
```
a_list

match(a_list,c(4,5,6))

match(a_list,list(c(4,5,6)))

match("1",c(3,5,8,1,99))      # returns the position/index
```

$first 1. 1 2. 2 3. 3

$second 1. 4 2. 5 3. 6

1. <NA> 2. <NA>

1. <NA> 2. 1

4

[117]:
```
d

grep("two",d)    # returns the position/index
```

1. 'one' 2. 'two' 3. 'three' 4. 'four'

2

[129]:
```
# B4. Statistics

# B4.1. Data
set.seed(123456789)

a <- c(1:1000)
b <- c("one","two","NA",4,5:1000)
e <- rnorm(1000)
c <- 2 - 3*a + e
x <- as.data.frame(cbind(a,b,c))    # making it as a data frame consisting of␣
↪1000 rows and 3 columns (variables)
                                    # still has "one", "two", and missing␣
↪variables

x <- as.data.frame(cbind(a,b,c), stringsAsFactors = FALSE)
```

[134]:
```
x$a <- as.numeric(x$a)

# x$a       # goes back to getting a(c:1000). without numeric, you get it in␣
↪string

x$c <- as.numeric(x$c)

# x$c

write.csv(x, "x.csv")
```

[137]:
```
# B4.2 Missing Values

2+NA

2*NA
```

```
2 + c(3,4,5,6,7,NA)
```

<NA>

<NA>

1. 5 2. 6 3. 7 4. 8 5. 9 6. <NA>

[150]:
```
# B4.3 Summary Statistics

mean(x$a)

sd(x$a)

quantile(x$a, c(1:5)/5)

# rowMeans(x[,c(1,3)])

colMeans(x[,c(1,3)])    # important! it takes a mean for all row in a column.
                        # in this case, we indicate we take a mean for column a
 ↪and c


x$d <- NA
x[2:1000,]$d <- c(2:1000/10)
mean(x$d)
mean(x$d, na.rm=TRUE) # mean when ignoring NA
```

500.5

288.819436095749

20\% 200.8 **40\%** 400.6 **60\%** 600.4 **80\%** 800.2 **100\%** 1000

a 500.5 **c** -1499.48782712157

<NA>

50.1

[166]:
```
# B4.4 Regression

x <- read.csv("x.csv",as.is=TRUE)
# summary(x)
lm1 <- lm(c ~ a, data=x)
# summary(lm1)

summary(lm1)[[4]]    # SUPER IMPORTANT!!! HEHE

lm1$coefficients    # take just coefficients
```

```
glm1 <- glm(c > -1500 ~ a, family = binomial(link=probit), data=x)

glm1$coefficients
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2.097396	0.0641312421	32.70475	4.934713e-160
a	-3.000170	0.0001109953	-27029.69905	0.000000e+00

(Intercept) 2.0973962102833 **a** -3.00017027638733

```
Warning message:
"glm.fit: algorithm did not converge"Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"
```

(Intercept) 3799.86765848464 **a** -7.59214317379549

[170]:
```
# B5. Control

# B5.1. Loops

# B5.2. Looping in R

# don't do it this way!!! WRONG EXAMPLE !!!
start_time <- Sys.time()
A <- NULL
for (i in 1:10000) {
    A <- rbind(A,c(i,i+1,i+2))
}
# A
Sys.time() - start_time

# A[400,]
```

```
Time difference of 0.2747722 secs
```

1. 400 2. 401 3. 402

[175]:
```
# A faster Way
start_time <- Sys.time()
A <- matrix(NA,10000,3)
for (i in 1:10000) {
    A[i,] <- c(i,i+1,i+2)
}
# A
Sys.time() - start_time

A[400,]

sum(A)
```

```
Time difference of 0.01392508 secs
```

1. 400 2. 401 3. 402

150045000

[179]:
```r
# An even faster way!!! (sometimes)
start_time <- Sys.time()
A <- t(matrix(sapply(1:10000,
                     function(x) c(x,x+1,x+2)), nrow=3))

Sys.time() - start_time
# sapply is similar of doing "for loop" for matrix

A[400,]
sum(A)
```

```
Time difference of 0.0149281 secs
```

1. 400 2. 401 3. 402

150045000

[183]:
```r
# B5.3 IF ELSE

a <- c(1,2,3,4,5)
b <- ifelse(a==3,82,a)
a
b

A <- "Chris"
if (A=="Chris") {
    print("Hey Chris")
} else {
    print(paste("Hey",A))
}
```

1. 1 2. 2 3. 3 4. 4 5. 5

1. 1 2. 2 3. 82 4. 4 5. 5

```
[1] "Hey Chris"
```

[191]:
```r
# B.6 Optimization

# B6.1 Function
y <- x[,c(2,4)]    # from dataset x.csv, (1000x2) taking only a and c

apply(y,2,mean)

colMeans(y)       # in this case, colMeans(y) is equivalent to apply(y,2,mean)
```

```
b <- c(1:dim(y)[1])

summary(sapply(b,function(x) sum(y[x,])),digits=2)

summary(rowSums(y), digits = 2)
```

a 500.5 **c** -1499.48782712157

a 500.5 **c** -1499.48782712157

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2000.0 -1500.0 -1000.0 -1000.0  -500.0     0.5
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2000.0 -1500.0 -1000.0 -1000.0  -500.0     0.5
```

[197]:
```
my_mean <- function(x) {
    if (is.numeric(x)) {
        return(mean(x, na.rm=TRUE))
    }
    else return("Not Numeric!")
}
# x$b
my_mean(x$b)

my_mean(x$a)

my_mean(x$c)
```

'Not Numeric!'

500.5

-1499.48782712157

[203]:
```
lm_iv <- function(y_in, X_in, Z_in = X_in, Reps = 100, min_in = 0.05, max_in = 0.
 ↪95) {
    # takes in the y variable, x explanatory varibales
    # and the z variables if available.
    # defaults: Z_in = X_in,
    # Reps = 100, min_in = 0.05, max_in = 0.95

    # Set up
    set.seed(123456789)
    index_na <- is.na(rowSums(cbind(y_in,X_in,Z_in)))  # cbind y, X, Z. then do␣
 ↪rowSums. then check NA. if yes, return ==1
    yt <- as.matrix(y_in[index_na==0])
    Xt <- as.matrix(cbind(1,X_in))
    Xt <- Xt[index_na==0]
```

```
    Zt <- as.matrix(cbind(1,Z_in))
    Zt <- Zt[index_na==0]
    N_temp <- length(yt)

    # turns the inputs into matrices
    # removes observations with any missing values
    # add column of 1s to X and Z

    # Bootstrap
    r <- c(1:Reps)
    bs_temp <- sapply(r, function(x) {
        ibs <- round(runif(N_temp, min = 1, max = N_temp))
        solve( t(Zt[ibs,])%*%Xt[ibs,] )%*%t(Zt[ibs,])%*%yt[ibs]
    } )

    # Present Results
    res_temp <- matrix(NA,dim(Xt))
    res_temp[,1] <- rowMeans(bs_temp)
    for (j in 1:dim(Xt)[2]) {
        res_temp[j,2] <- sd(bs_temp[j,])
        res_temp[j,3] <- quantile(bs_temp[j,],min_in)
        res_temp[j,4] <- quantile(bs_temp[j,],max_in)
    }

    colnames(res_temp) <- c("coef","sd",as.character(min_in),as.
 ↪character(max_in))
    return(res_temp)
}

# lmiv1  <- lm_iv(x$c, x$a)
# lmiv1
# IT DOES NOT WORK!!!! NOT YET MITIGATED
```

```
Error in Zt[ibs, ]: incorrect number of dimensions
Traceback:

1. lm_iv(x$c, x$a)
2. sapply(r, function(x) {
 .     ibs <- round(runif(N_temp, min = 1, max = N_temp))
 .     solve(t(Zt[ibs, ]) %*% Xt[ibs, ]) %*% t(Zt[ibs, ]) %*% yt[ibs]
 . })   # at line 23-26 of file <text>
3. lapply(X = X, FUN = FUN, ...)
4. FUN(X[[i]], ...)
5. solve(t(Zt[ibs, ]) %*% Xt[ibs, ])   # at line 25 of file <text>
6. t(Zt[ibs, ])   # at line 25 of file <text>
```

[208]:
```r
# B6.2 optim()

f_ols <- function(beta, y_in, X_in) {
    X_in <- as.matrix(cbind(1,X_in))
    if (length(beta)==dim(X_in)[2]) {
        return(mean((y_in - X_in%*%beta)^2, na.rm = TRUE))
    }
    else {
        return("The number of parameters does not match.")
    }
}

lm_ols <- optim(par=c(2,-3),fn=f_ols,y_in=x$c,X_in=x$a)
lm_ols

lm(x$c ~ x$a)

lm_ols1 <- optim(par=c(1,1),fn=f_ols,y_in=x$c,X_in=x$a)
lm_ols1
```

$par 1. 2.09728736536897 2. -3.00017009107694

$value 1.02460919703226

$counts **function** 67 **gradient** <NA>

$convergence 0

$message NULL

```
Call:
lm(formula = x$c ~ x$a)

Coefficients:
(Intercept)          x$a
      2.097       -3.000
```

$par 1. 1.67590292801324 2. -2.99936153875897

$value 1.07939341375334

$counts **function** 49 **gradient** <NA>

$convergence 0

$message NULL

[ ]: