R overview and proficiency exercises

October 6, 2006

Contents

N.B.: If you are using UNIX, ignore the disk specifications in any folder-related instructions. When I mention 'Course version of R' I mean the one supplied on CD.

1 Containers

1.1 The basic containers illustrated

```
> x <- c(1, 2, 3, 4)
> x[2]

[1] 2

> y <- c(5, 6, 7, 8)
> y[c(2, 3)]

[1] 6 7

> m <- cbind(x, y)
> m

     x y
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8

> m[, "x"]

[1] 1 2 3 4
```

1.2 symbol resolution, workspace

- assignment of a value to a symbol is accomplished by `<-` or `=`
- a workspace called `.GlobalEnv` is the dynamic repository of all creations in a session
- persistent symbols can be identified using `objects()`
- the entire search space for symbol resolution is identified using `search()`
- objects in `.GlobalEnv` are removed using `rm()`; searchlist elements are dropped using `detach()`

1.3 R: names attributes; "associative" indexing

- numeric indexing of linear structures is fine
- names of elements are also useful

```
> names(x) <- c("a", "b", "c", "d")
> x["b"]

b
2

> rownames(m) <- LETTERS[1:4]
> m

  x y
A 1 5
B 2 6
C 3 7
D 4 8

> m["A", "y"]

[1] 5
```

- Upshots:

```
> litdf <- data.frame(samp1 = c(33, 22, 12), samp2 = c(44, 111,
+     13))
> rownames(litdf) <- c("CRP", "BRCA1", "HOXA")
> litdf

      samp1 samp2
CRP      33    44
BRCA1    22   111
HOXA     12    13

> litdf["CRP", ]

    samp1 samp2
CRP    33    44

> litdf[, "samp1"]

[1] 33 22 12

> litdf["HOXA", "samp2"]

[1] 13
```

1.4 Selection rules

- explicit numeric selection
- named element selection
- logical filtering

```
> x

a b c d
1 2 3 4

> y

[1] 5 6 7 8

> keep <- c(TRUE, TRUE, FALSE, FALSE, TRUE)
> x[keep]

   a    b <NA>
   1    2   NA

> x[y > 6]

c d
3 4

> which(y > 6)

[1] 3 4
```

1.5 Other containers

```
> gender <- factor(c("M", "M", "F", "F"))
> gender

[1] M M F F
Levels: F M

> season <- ordered(c("spring", "summer", "fall", "winter"), levels = c("spring",
+     "summer", "fall", "winter"))
> season

[1] spring summer fall   winter
Levels: spring < summer < fall < winter
```

```
> df <- data.frame(m, gender, season)
> df

  x y gender season
A 1 5      M spring
B 2 6      M summer
C 3 7      F   fall
D 4 8      F winter
```

1.6 Accessing data.frame elements

```
> df

  x y gender season
A 1 5      M spring
B 2 6      M summer
C 3 7      F   fall
D 4 8      F winter

> df$gender

[1] M M F F
Levels: F M

> df["B", ]

  x y gender season
B 2 6      M summer

> z <- "season"
> df[[z]]

[1] spring summer fall   winter
Levels: spring < summer < fall < winter
```

1.7 Merging data.frames

- Default merge finds matching attributes and concatenates

```
> df

  x y gender season
A 1 5      M spring
B 2 6      M summer
C 3 7      F   fall
D 4 8      F winter
```

```
> df2

  x z
1 2 1
2 3 6
3 4 7
4 5 8

> merge(df, df2)

  x y gender season z
1 2 6      M summer 1
2 3 7      F   fall 6
3 4 8      F winter 7
```

- Conservative merge:

```
> merge(df, df2, all = TRUE)

  x  y gender season  z
1 1  5      M spring NA
2 2  6      M summer  1
3 3  7      F   fall  6
4 4  8      F winter  7
5 5 NA   <NA>   <NA>  8
```

- other parameters are available for alternate combos

1.8 Lists

- linearly or associatively ordered heterogeneous collection

```
> l1 <- list(df, x = x, fundem = mean)
> l1

[[1]]
  x y gender season
A 1 5      M spring
B 2 6      M summer
C 3 7      F   fall
D 4 8      F winter

$x
a b c d
```

```
1 2 3 4

$fundem
function (x, ...)
UseMethod("mean")
<environment: namespace:base>
```

- Split vectors to make lists

```
> dx <- c("ALL", "ALL", "AML", "AML", "ALL", "ALL", "ALL", "AML")
> ddr1 <- c(12.2, 13.1, 7.2, 6.4, 14.2, 15.3, 9.2, 10)
> split(ddr1, dx)

$ALL
[1] 12.2 13.1 14.2 15.3  9.2

$AML
[1]  7.2  6.4 10.0
```

- Access to list elements:

```
> l1[[1]]

  x y gender season
A 1 5      M spring
B 2 6      M summer
C 3 7      F   fall
D 4 8      F winter

> l1$x

a b c d
1 2 3 4

> l1$fund

function (x, ...)
UseMethod("mean")
<environment: namespace:base>
```

- Single and double bracket selection:
  - subset vectors using single brackets
    - `x[2:3]` is a 2-vector

- subset lists to sublists using single brackets
  - `l1[1:2]` is a list of 2 elements
- retrieve list elements using double brackets or $
  - `l1[[2]]` is the second element of `l1`

1.9 Generating and computing on random data

```
> table(rpois(1000, 3))

  0   1   2   3   4   5   6   7   8   9
 47 156 226 215 162 104  65  18   3   4

> table(rbinom(1000, 5, 0.5))

  0   1   2   3   4   5
 33 169 305 308 154  31

> sd(rnorm(1000, 0, 0.4))

[1] 0.4079823

> sd(rnorm(1000, 0, 0.4))

[1] 0.4057623

> set.seed(1234)
> sd(rnorm(1000, 0, 0.4))

[1] 0.3989351

> sd(rnorm(1000, 0, 0.4))

[1] 0.3924771

> set.seed(1234)
> sd(rnorm(1000, 0, 0.4))

[1] 0.3989351
```

2 Language elements

2.1 Functions; packages

- a function has a parameter list and a body
- the parameter lists identify inputs
- the bodies manipulate inputs and other entities to compute and return a final value
- symbol resolution occurs by inspecting symbols defined in the current function body, its enclosing environments, and then traversing the searchlist

```
> myfun1 <- function(x, y) {
+     x + 3 * y
+ }
> myfun1(2, 3)

[1] 11
```

- `x` and `y` are formal parameters
- `+` and `*` are primitive functions

```
> myfun1(2, c(3, 4, 5, 6))

[1] 11 14 17 20
```

2.2 RECYCLING RULE

- arithmetic combination of vector and scalar is permitted
- binary operator functions obey recycling rule
- when a binary operator has operands of different lengths, the shorter is replicated to the length of the longer (with truncation if necessary)
- the binary operation is then executed elementwise
- warning if replication leads to truncation

```
> 4 * c(2, 3, 4, 5)

[1]  8 12 16 20

> c(2, 3) * c(4, 5, 6)

[1]  8 15 12
```

2.3 Defining functions

- you can define them interactively on the fly as above
- you can write a text file containing the definition and use `source` to load that into `.GlobalEnv` for any session
- you can put the function in the `R` directory of a source package and then INSTALL the package (after documenting the function)
- you CAN NOT add a function to an INSTALLed package, don't even try

2.4 Iteration

```
> for (i in 1:2) print(i, season[i])

[1] 1
[1] 2

> for (i in 1:3) print(l1[i])

[[1]]
  x y gender season
A 1 5      M spring
B 2 6      M summer
C 3 7      F   fall
D 4 8      F winter

$x
a b c d
1 2 3 4

$fundem
function (x, ...)
UseMethod("mean")
<environment: namespace:base>

> for (s in season) print(s)

[1] 1
[1] 2
[1] 3
[1] 4

> for (s in as.character(season)) print(s)
```

```
[1] "spring"
[1] "summer"
[1] "fall"
[1] "winter"
```

Powerful alternatives to `for` loops include

- vectorized computations (e.g., `z = x+y` is equivalent to

```
z = rep(NA, length(x))
for (i in 1:length(x))
  z[i] = x[i]+y[i]
```

- use of elements of the `apply` family (see section 2.5)

2.5 Applying functions

- it is often of interest to apply a function repetitively over all the rows or columns of a matrix, or all the elements of a list or vector

```
> m

  x y
A 1 5
B 2 6
C 3 7
D 4 8

> apply(m, 1, sum)

 A  B  C  D
 6  8 10 12

> apply(m, 2, "^", 3)

   x   y
A  1 125
B  8 216
C 27 343
D 64 512
```

2.6 Apply family

- `apply(x,MARGIN,FUN,...)`, `x` is a matrix or array, `MARGIN` is 1 for rows, 2 for columns, etc., `FUN` is a function to be evaluated on the extracted vectors; the dots are there to accept additional fixed arguments to be supplied to `FUN`
- `lapply(x,FUN,...)` returns a list of values returned by `FUN` applied to elements of list `x`
- `sapply(x,FUN,...)` returns the simplest sensible structure of values returned by `FUN` applied to elements of list `x`
- `mapply(FUN,..., MoreArgs)` is an apply of multiparameter functions over multiple vectors

2.7 Summary thus far

- session concepts: `.GlobalEnv`, searchlist, history, save.image, savehistory
- basic containers: vector, matrix, array, list, data.frame
- basic constructor functions: `c`, `cbind`, `rbind`, `matrix`, `data.frame`
- basic vector or matrix element access: `[]` with numeric arguments, or `[]` with name strings
- basic list or data.frame component access: `$` or `[[]]`
- functions can be constructed on the fly or in script files
- functions can be repetitively applied with the apply family

3 Exercises: Workspace management and recovery

3.1. Create the folders `C:\WS1` and `C:\WS2`. Start the Course version of R and use `setwd` to change the working directory to `C:\WS1`. Issue the command

```
> date1 <- date()
```

Then close the session and save the workspace when queried. Start the Course version of R again. Does `date1` have a value? How can you recover the value of `date1`?

3.2. Change the working directory to `C:\WS2`, create and shut down R to save a workspace with `date2` taking the value of `date()`.

3.3. Start the Course version of R again, and show how to test whether `date1` is equal to `date2`.

4 Exercises: Saving session dialogue content; running stored programs

4.1. The `sink()` function will save information communicated by R to disk. Perform the following using a fresh R session:

```
> set.seed(1234)
> sink(file = "sink1.txt")
> rnorm(5)

[1] -1.2070657  0.2774292  1.0844412 -2.3456977  0.4291247

> sink()
> readLines("sink1.txt")
```

Verify that you obtain the numbers printed above.

4.2. Enter the following text into a TEXT file (you can use Word, but save as text) `C:\prog1.txt`. Be careful, Word may try to capitalize certain function or variable names, but R is case-sensitive.

```
set.seed(1234)
x1 <- runif(30)
x2 <- rnorm(30)
y <- x1+2*x2+rnorm(30,,3)
fit <- lm(y~x1+x2)
capture.output(summary(fit), file="C:/prog1out.txt")
```

Source the program you have written into R, and check the contents of `C:/prog1out.txt`. They should be:

```
Call:
lm(formula = Y ~ X1 + X2)

Residuals:
    Min      1Q  Median      3Q     Max
-5.3581 -2.9011 -0.4249  2.3477  7.3040

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.8954     1.5301   0.585  0.56327
X1           -0.5411     2.6124  -0.207  0.83747
X2            2.2342     0.7986   2.798  0.00938 **
```

```
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1  1

Residual standard error: 3.55 on 27 degrees of freedom
Multiple R-Squared: 0.2585,Adjusted R-squared: 0.2036
F-statistic: 4.707 on 2 and 27 DF,  p-value: 0.01763
```

5 Exercises: Vectors and predicates

Try to do the following without using R.

5.1. What is

```
> length(letters)
```

5.2. What is

```
> length(letters == LETTERS)
```

5.3. What is

```
> all(letters == tolower(LETTERS))
```

5.4. What is

```
> which(letters %in% c("a", "d"))
```

5.5. What is

```
> which(c("a", "d") %in% letters)
```

5.6. What is

```
> letters[LETTERS > "W"]
```

5.7. What is

```
> letters[!LETTERS > "C"]
```

5.8. What is

```
> sum(LETTERS > "c")
```

5.9. What is

```
> seq(1, 20, 3)
```

5.10. Give two ways of generating the even integers from 1 to 20, one using `seq`, the other using `%%`

6 Exercises: simulation, tables, matrices, arrays

6.1. What is

```
> round(mean(rnorm(1000)), 2)
```

What can be done to ensure that the same result is obtained whenever this computation is run?

6.2. What is

```
> mean(rexp(1000, 10))
```

How can you determine the parameterization in use?

6.3. Tabulation:

```
> kp <- rpois(100, 5)
> table(kp)

kp
 1  2  3  4  5  6  7  8 11
 6  8 21 22 17 10  8  7  1

> bp <- rbinom(100, 4, 0.3)
> table(bp)

bp
 0  1  2  3  4
16 47 24 10  3

> table(bp, kp)

   kp
bp  1 2 3 4 5 6 7 8 11
  0 1 0 4 5 3 1 0 2  0
  1 0 5 9 9 9 5 6 3  1
  2 3 3 5 5 3 2 2 1  0
  3 2 0 2 3 2 1 0 0  0
  4 0 0 1 0 0 1 0 1  0

> cols <- sample(c("green", "blue"), replace = TRUE, size = 100)
> table(cols, kp)
```

```
       kp
cols    1  2  3  4  5  6  7  8 11
  blue  1  3 12 13 10  3  7  3  0
  green 5  5  9  9  7  7  1  4  1
```

6.4. Why is

```
> x <- matrix(1:10, nr = 10, nc = 4)
> x
```

```
      [,1] [,2] [,3] [,4]
 [1,]    1    1    1    1
 [2,]    2    2    2    2
 [3,]    3    3    3    3
 [4,]    4    4    4    4
 [5,]    5    5    5    5
 [6,]    6    6    6    6
 [7,]    7    7    7    7
 [8,]    8    8    8    8
 [9,]    9    9    9    9
[10,]   10   10   10   10
```

the way it is?

6.5. Explain

```
> data(iris3)
> dim(iris3)
```

```
[1] 50  4  3
```

```
> dim(iris3[, , 1])
```

```
[1] 50  4
```

6.6. What is the use of

```
> expand.grid(c("M", "F"), c("trt", "control"))
```

```
  Var1    Var2
1    M     trt
2    F     trt
3    M control
4    F control
```

6.7. What are

```
> dim(cbind(x, x))
> x + 4
> x + x
> 2 * x
> x/c(2, 3)
> x + x[, -1]
> t(x) %*% x
> row(x)
> nrow(x)
> x[x[, 3] > 5, ]
```