Chapter 5

R programming

Instructor: Li, Han

Contents

- control flow: for, while, if-else, switch
- user defined function
- break
- stop(), warning() and message()
- global vs local environment
- R vectorization

R has the standard control structures you'd expect to see in a modern programming language.

- statement is a single R statement or a compound statement separated by semicolons or newline.
- cond is an expression that is judged to be true or false.
- expr is a statement that evaluates a number or character string.
- seq is a sequence of numbers or character strings.

Repetition and looping

Looping constructs repetitively execute a statement or series of statements until a condition is false. These include structures of

- for
- while

for

The for loop executes a statement repetitively until the last element in the sequence seq. The syntax is

for (var in seq) statement

Example 1 (for loop)

```
for (i in 1:10) {
    print("Hello")
}
```

while

A while loop executes a statement repetitively until the condition is no longer true. The syntax is

while (cond) statement

Example 2 (while loop)

```
i <- 10
while (i > 0) {
    print("Hello")
    i <- i - 1
}
```

In the above example, if you add 1 on each loop, R would never stop saying “Hello”. This is why while loops can be more dangerous than other looping constructs. In such case, you need to "stop" the loop forcefully.

Looping in R can be inefficient and time consuming when you are processing the rows or columns of large datasets. Whenever possible, it's better to use R's built-in numerical and character functions in conjunction with the apply family of functions.

Example 3 (caculate the column means)

```
x <- rnorm(100)
y <- matrix(x, 10,10)
z <- rep(0,ncol(y))
for (i in 1:ncol(y)) {
    z[i]=mean(y[,i])
}

w <- apply(y, 2, mean)
```

Conditional execution

In conditional execution, a statement or statements are only executed if a specified condition is met. These constructs include

- ifelse
- if-else
- switch

ifelse

The ifelse construct is a compact and vectorized version of the if-else construct. The syntax is

```
ifelse(cond, statement1, statement2)
```

Example 4 (ifelse)

```
score <- 80
ifelse(score > 60, print("Passed"), print("Failed"))
outcome <- ifelse(score > 60, "Passed", "Failed")
```

if-else

For the above example, we could also use the syntax

```
if(cond){ statement1} else{statement2}
```

Example 5 (if-else)

```
score <- 80
if (score > 60) {print("Passed")} else {print("Failed")}
```

If we have more than one condition to evaluate, we could use multiple if-else. The syntax is

```
if (cond 1) {
    statement 1
} else if (cond 2) {
    statement 2
}  ... {
} else if (cond k) {
   statement k
} else {
   statement n
}
```

Example 6 (multiple conditions)

```
score <- 90
If (score >85) {
    print("Excellent")
} else if (score >75) {
    print("Good")
} else if (score >60) {
    print("Passed")
} else{
    print("Failed")
}
```

switch

switch chooses statements based on the value of an expression. The syntax is

switch(expr, ...)

Then it compares the expr with the predefined string that matches it, and then excutes the corresponding statements.

Example 7 (switch)

```
feelings <- c("sad", "afraid")
for (i in feelings) {
   print(
    switch(i,
    happy  = "I am glad you are happy",
    afraid = "There is nothing to fear",
    sad    = "Cheer up",
    angry  = "Calm down now"
    )
  )
}
```

functions

We could build our own functions in R. The structure of a function looks like this:

```
myfunction <- function(arg1, arg2, ... ) {
   statements
   return(object)
}
```

Objects in the function are local to the function. The object returned can be any data type, from a single number to a list.

Example 8 (calculate the center and spread of the data)

```
mystats <- function(x, parametric=TRUE, print=FALSE) {
  if (parametric) {
    center <- mean(x); spread <- sd(x)
  } else {
    center <- median(x); spread <- mad(x)
  }
  if (print & parametric) {
    cat("Mean=", center, "\n", "SD=", spread, "\n")
  } else if (print & !parametric) {
    cat("Median=", center, "\n", "MAD=", spread, "\n")
  }
  result <- list(center=center, spread=spread)
  return(result)
}
```

```
set.seed(1234)
x <- rnorm(500)
y <- mystats(x)

y <- mystats(x, parametric=FALSE, print=TRUE)
```

break

In a loop, if we want to stop it, use break.

Example 9 (break)

```
for(i in 1:10){
  print(i)
  if(i>3){break}
}
```

stop()

If you want to stop the current excution, use stop().

Example 10 (stop)

```
for(i in 1:10){
  print(i)
  if(i>3){stop("It is larger than 3.")}
  print("fine")
}
```

warning()

Example 11(warning)

```
for(i in 1:10){
  print(i)
  if(i>3){warning("It is larger than 3.")}
  print("fine")
}
```

message()

Example 12 (message)

```
for(i in 1:10){
  print(i)
  if(i>3){message("It is larger than 3.")}
  print("fine")
}
```

global vs local environment

In R, each function defines an environment that includes all its local variables that can not be accessed from outside. If the function finishes running, all the local variables disappear.

A local function can access the variables in its parent environment when running. Note that it actually copies the global variable, and it is not the global variable itself. So it can not change the value of the global variable.

Example 13 (global vs local environment)

```
w <- 12
f <- function(){
  y <-5
  h <- function(z){return(z^2)}
  w <- 10
  print(w)
  return(h(y))
  }
f()
w
y
```

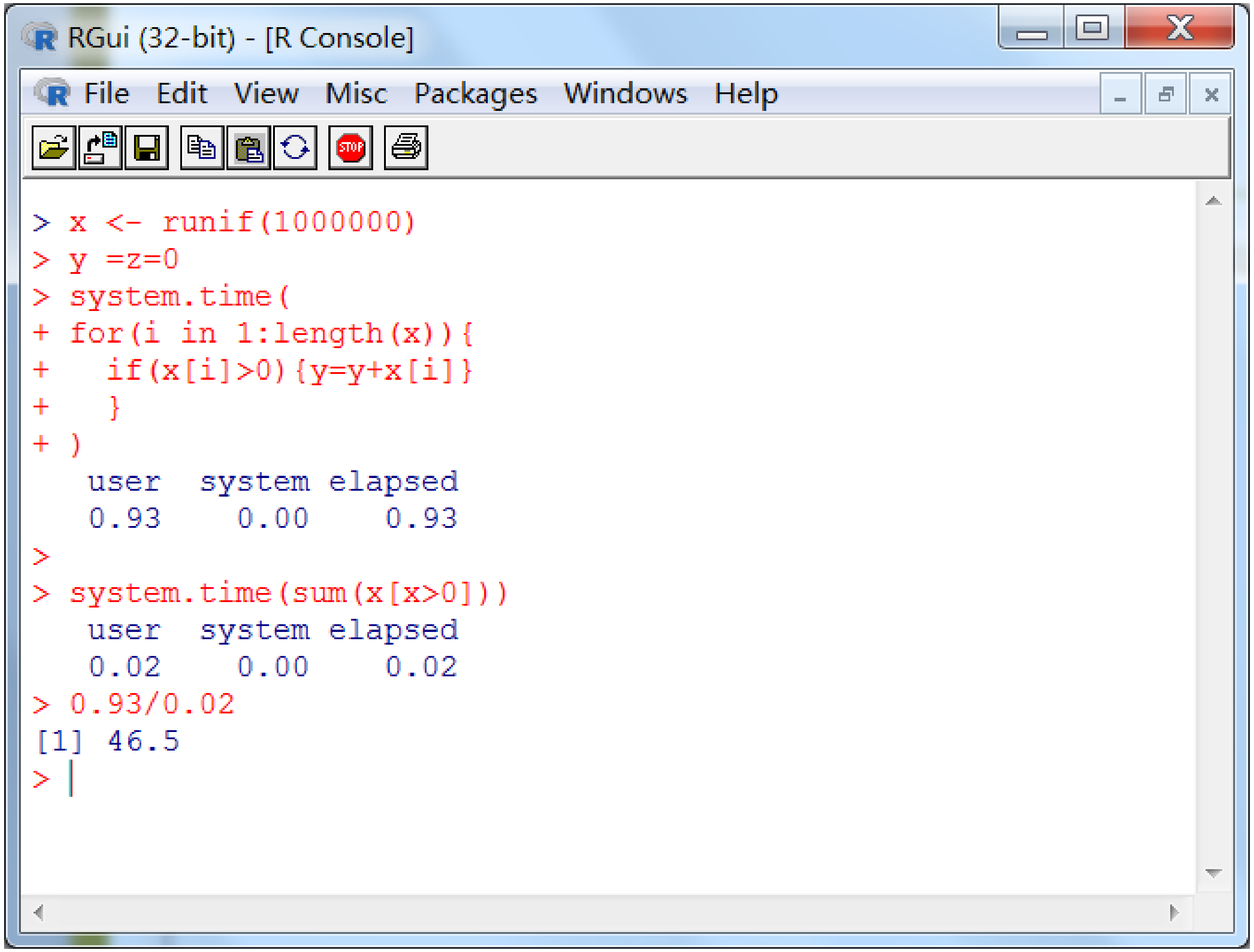
R vectorization

Compare with the following two examples that summarize the sum of those x>0.

Example 14 (use R vectorization if possible)

```
x <- runif(1000000)
y =z=0
system.time(
for(i in 1:length(x)){
  if(x[i]>0){y=y+x[i]}
  }
)

system.time(sum(x[x>0]))
```


RGui (32-bit) - [R Console]
File Edit View Misc Packages Windows Help

```
> x <- runif(1000000)
> y =z=0
> system.time(
+ for(i in 1:length(x)){
+   if(x[i]>0){y=y+x[i]}
+   }
+ )
   user  system elapsed 
   0.93    0.00    0.93 
> 
> system.time(sum(x[x>0]))
   user  system elapsed 
   0.02    0.00    0.02 
> 0.93/0.02
[1] 46.5
> 
```

Summary

In this session, we have learned how to use the control flow structures and build functions in R.