R Programming Language
R03 Control Structure

林 建 甫
C.F. Jeff Lin, MD. PhD.

台 北 大 學 統 計 系 助 理 教 授
台 北 榮 民 總 醫 院 生 物 統 計 顧 問
美 國 密 西 根 大 學 生 物 統 計 博 士

Control Structures and Functions

Semicolons & Newlines

```
# semicolons
> x <- 0; x + 5
[1] 5
# new lines
> y <- 1:10
> 1; 2
[1] 1
[1] 2
```


{...} : Grouped Expressions in R

```
> x <- 1:9
> { x <- 0
  + x + 5
  + }
  [1] 5

if (length(x) <= 10)
{
  x <- c(x,10:20);
     print(x)
}else
{
     print(x[1])
}
```


Conditional Execution Statements and Loops

Branching: if(…){…} else{…}

if (logical expression) {
  statements
} **else** {
  alternative statements
}

else branch is optional

if(…){…} else{…}

```
x <- 1:5
if (length(x) <= 10) {
   x <- c(x,10:15);print(x) }

if (length(x) < 5) print(x) else print(x[5:20])
```


if(…){…} else{…}

```
> x <- 1:5
> if (length(x) <= 10) {
+    x <- c(x,10:15);print(x) }
 [1]  1  2  3  4  5 10 11 12 13 14 15

> if (length(x) < 5) print(x) else print(x[5:20])
 [1]  5 10 11 12 13 14 15 NA NA NA NA NA NA
      NA NA NA
```


Repetitive Execution

- **for**
- **while**
- **repeat**

for{…} Loops in R

```
>for(i in 1:10) {
    x[i] <- rnorm(1)
}
```


Iterations in Vectors

Most of the commands in R perform iteration implicitly. That is, if you ask for the square root of a vector, R will work out the square root for all the members inside the vector.

```
> a<-c(1,2.3,4,5,6)
   > sqrt(a)
   [1] 1.000000 1.516575 2.000000 2.236068
     2.449490
```


Loops

- When the same or similar tasks need to be performed multiple times; for all elements of a list; for all columns of an array; etc.
  - Monte Carlo Simulation
  - Cross-Validation (delete one and etc)

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

i=1
while(i<=10) {
  print(i*i)
  i=i+sqrt(i)
}
```

for() Loops

```
# for (name in expr1) { expr2 }
> x <-rep(0,5)
> for(i in 1:5) {
   print(i)
   x[i] <- rnorm(1)
   print(x[i])
}
```

where **name** is the loop variable. **expr1** is a vector expression, (often a sequence like 1:20), and **expr 2** is often a grouped expression with its sub-expressions written in terms of the dummy name. **expr2** is repeatedly evaluated as name ranges through the values in the vector result of **expr1**.

for() Loops

```
> for(i in 1:5) {
+   print(i)
+     x[i] <- rnorm(1)
+     print(x[i])
+ }
[1] 1
[1] 0.4454815
[1] 2
[1] -0.6491777
[1] 3
[1] -1.274070
[1] 4
[1] 2.382059
[1] 5
[1] -0.3947709
```


repeat

- **repeat expr**
- **repeat {operations, test, break}**
- The break statement can be used to terminate any loop, possibly abnormally. This is the only way to terminate repeat loops.
- The next statement can be used to discontinue one particular cycle and skip to the "next".

repeat & break

```
j <-1
repeat {
  print(j)
  j <- j + j/3
  if (j > 10) break
}
```


repeat & break

```
> j <-1
> repeat {
+   print(j)
+   j <- j + j/3
+   if (j > 10) break
+ }
[1] 1
[1] 1.333333
[1] 1.777778
[1] 2.370370
[1] 3.160494
[1] 4.213992
[1] 5.618656
[1] 7.491541
[1] 9.988721
```


While()

- while (condition) expr
- The break statement can be used to terminate any loop, possibly abnormally. This is the only way to terminate repeat loops.
- The next statement can be used to discontinue one particular cycle and skip to the "next".

while in R

```
j = 1
while( j  < 10) {
  j <- j + 2
  print(j)
}
```


while in R

```
> j = 1
> while( j  < 10) {
+   j <- j + 2
+   print(j)
+ }
[1] 3
[1] 5
[1] 7
[1] 9
[1] 11
```


switch

switch(flag, dolist)

```
citizen="uk"
switch(citizen,au="Aussie",uk="Brit",
    us="Yankee",ca="Canuck")
[1] "Brit"
```


Functions

Functions

- Functions do things with data
  - "Input": function arguments (0,1,2,…)
  - "Output": function result (exactly one)

```
> xvec<-seq(1,10)
> mean(xvec)
[1] 5.5
```


Common Use Functions

- abs(x)
- exp(x)
- log(x)
- log10(x)
- sqrt(x)
- sign(x)
- gamma(x)

Common Use Functions

- round(x)
- trunc(x)
- ceiling(x)
- floor(x)
- sort(x)
- rev(x)
- rank(x)
- order(x)

Common Use Functions

- sin(x) cos(x) tan (x)
- asin(x) acos(x) atan(x)
- sinh(x) cosh(x) tanh(x)
- asinh(x) acosh(x) atanh(x)

Functions

```
standard.deviation<-function(x){
    sqrt(var(x))
    }
x<-rnorm(100,mean=0,sd=1)
var(x)
standard.deviation(x)
sd(x)
```


Functions

```
> standard.deviation<-function(x){
+ sqrt(var(x))
+ }
> x<-rnorm(100,mean=0,sd=1)
> var(x)
[1] 0.8109763
> standard.deviation(x)
[1] 0.9005422
> sd(x)
[1] 0.9005422
```


Functions

- Example:

```
add = function(a,b)
{ result = a+b
  return(result) }
```

- Operators:
- Short-cut writing for frequently used functions of one or two arguments.
- Examples: + - * / ! & | %%

General Form of Functions

```
function(arguments) {
    expression
   }

  larger <- function(x,y) {
  if(any(x < 0)) return(NA)
  y.is.bigger <- y > x
  x[y.is.bigger] <- y[y.is.bigger]
  x
}
```

If you are in doubt...

```
> help (predict)
 'predict' is a generic function for predictions
     from the results
  of various model fitting functions.
> help (predict.lm)
  'predict.lm' produces predicted values,
     obtained by evaluating the
  regression function in the frame 'newdata'
 > predict(lm(dist~speed),newdata)
```


Calling Conventions for Functions

- Arguments may be specified in the same order in which they occur in function definition, in which case the values are supplied in order.
- Arguments may be specified as name=value, when the order in which the arguments appear is irrelevant.
- Above two rules can be mixed.

```
> t.test(x1, y1, var.equal=F, conf.level=.99)
> t.test(var.equal=F, conf.level=.99, x1, y1)
```


Probability Distributions

Probability Distributions

- norm, binom, beta, cauchy, chisq, exp, f, gamma, geom, hyper, lnorm, logis, nbinom, t, unif, weibull, wilcox
- Four prefixes:
  - ‘d’ for density (PDF)
  - ‘p’ for distribution (CDF)
  - ‘q’ for quantile (percentiles)
  - ‘r’ for random generation (simulation)
- Each distribution has arguments that need to be specified
- i.e., dnorm, pnorm, qnorm, rnorm

Probability Distributions

- Cumulative distribution function $P(X \leq x)$: ‘*p*’ for the CDF
- Probability density function: ‘d’ for the density,,
- Quantile function (given q, the smallest x such that $P(X \leq x) > q$): ‘*q*’ for the quantile
- simulate from the distribution: ‘r’

Probability Distributions

Distribution	R name	additional arguments
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
chi-squared	chisq	df, ncp
exponential	exp	rate
F	f	df1, df1, ncp
gamma	gamma	shape, scale
geometric	geom	prob
hypergeometric	hyper	m, n, k
log-normal	lnorm	meanlog, sdlog

logistic logis; negative binomial nbinom; normal norm; Poisson pois; Student’s t t ; uniform unif; Weibull weibull; Wilcoxon wilcox

Missing Arguments

- R function can handle missing arguments two ways:
- either by providing a default expression in the argument list of definition, or
- by testing explicitly for missing arguments.

Missing Arguments in Functions

```
> add <- function(x,y=0){x + y}
> add(4)

> add <- function(x,y){
  if(missing(y)) x
  else x+y
}
> add(4)
```


Variable Number of Arguments

- The special argument name "…" in the function definition will match any number of arguments in the call.
- nargs() returns the number of arguments in the current call.

Variable Number of Arguments

```
> mean.of.all <- function(…) mean(c(…))
> mean.of.all(1:10,20:100,12:14)

> mean.of.means <- function(…)
{
  means <- numeric()
  for(x in list(…)) means <- c(means,mean(x))
  mean(means)
}
```


Variable Number of Arguments

```
mean.of.means <- function(…)
{
  n <- nargs()
  means <- numeric(n)
  all.x <- list(…)
  for(j in 1:n) means[j] <- mean(all.x[[j]])
  mean(means)
}
mean.of.means(1:10,10:100)
```


Output From a Function

The last line of a function should be the output.

```
myfun=function(x){
        y=3*x
        y}

myfun=function(x){
        y=3*x
        list(x=x,y=y)
        }
```

Useful Functions

```
> seq(2,12,by=2)
[1] 2 4 6 8 10 12
> seq(4,5,length=5)
[1] 4.00 4.25 4.50 4.75 5.00
> rep(4,10)
[1] 4 4 4 4 4 4 4 4 4 4

> paste("V",1:5,sep="")
[1] "V1" "V2" "V3" "V4" "V5"

> LETTERS[1:7]
[1] "A" "B" "C" "D" "E" "F" "G"
```


Mathematical Operation

Opérations usuelles : **+ - * /**
Puissances: **2^5** ou bien **2**2**5**
Divisions entières: **%/%**
Modulus: **%%** **(7%%5** gives **2)**

Fonctions standards: **abs(), sign(), log(), log10(), sqrt(), exp(), sin(), cos(), tan() gamma(), lgamma(), choose()**

Pour arrondir: **round(x,3)** arrondi à 3 chiffres après la virgule

Et aussi: **floor(2.5)** donne 2, **ceiling(2.5)** donne 3

Vector Functions

```
> vec <- c(5,4,6,11,14,19)
> sum(vec)
[1] 59
> prod(vec)
[1] 351120
> mean(vec)
[1] 9.833333
> median(vec)
[1] 8.5
> var(vec)
[1] 34.96667
> sd(vec)
[1] 5.913262
> summary(vec)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.000   5.250   8.500   9.833  13.250  19.000
```

And also: **min()** **max()** **cummin()** **cummax()** **range()**

Logic Functions

R contains : **TRUE** (ou **T**) et **FALSE** (ou **F**).
Exemple:

```
> 3 == 4
[1] FALSE
> 4 > 3
[1] TRUE

> x <- -4:3
> x > 1
[1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE
> sum(x[x>1])
[1] 5
> sum(x>1)
[1] 2
```

==	exactement égal
<	plus petit
>	plus grand
<=	plus petit ou égal
>=	plus grand ou égal
!=	différent
&	“et” (“and”)
|	“ou” (“or”)

lapply, sapply, apply

"apply" and Its Relatives

Often we want to repeat the same function on all the rows or columns of a matrix, or all the elements of a list.

We could do this in a loop, but R has a more efficient operator the apply function

Applying a Function to the Rows or Columns of a Matrix

If mat is a ***matrix*** and ***fun*** is a function (such as ***mean***, ***var***, ***lm*** ...) that takes a vector as its arguments, then

apply(mat,1,fun) applies fun to each row of mat

apply(mat,2,fun) applies fun to each column of mat

In either case, the output is a vector.

apply

apply(arr, margin, fct)
Apply the function fct along some dimensions of the array arr, according to margin, and return a vector or array of the appropriate size.

```
> x
     [,1] [,2] [,3]
[1,]    5    7    0
[2,]    7    9    8
[3,]    4    6    7
[4,]    6    3    5
> apply(x, 1, sum)
[1] 12 24 17 14
> apply(x, 2, sum)
[1] 22 25 20
```


lapply

- When the same or similar tasks need to be performed multiple times for all elements of a list or for all columns of an array.
  - May be easier and faster than "for" loops
- **lapply(li, *function*)**
  - To each element of the list li, the function *function* is applied.
  - The result is a list whose elements are the individual *function* results.

```
> li = list("klaus","martin","georg")
> lapply(li, toupper)
> [[1]]
> [1] "KLAUS"
> [[2]]
> [1] "MARTIN"
> [[3]]
> [1] "GEORG"
```


Relatives of "apply"

lapply(list,fun) applies the function to every element of list

tapply(x,factor,fun) uses the factor to split x into groups, and then applies fun to each group

sapply

sapply(li, fct)
Like apply, but tries to simplify the result, by converting it into a vector or array of appropriate size

```
> li = list("klaus","martin","georg")
> sapply(li, toupper)
[1] "KLAUS"  "MARTIN" "GEORG"

> fct <- function(x) { return(c(x, x*x, x*x*x)) }
> sapply(1:5, fct)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    4    9   16   25
[3,]    1    8   27   64  125
```


Frequently Used Operators

`<-`	Assign
`+`	Sum
`-`	Difference
`*`	Multiplication
`/`	Division
`^`	Exponent
`%%`	Mod
`%*%`	Dot product
`%/%`	Integer division
`%in%`	Subset

`|`	Or
`&`	And
`<`	Less
`>`	Greater
`<=`	Less or =
`>=`	Greater or =
`!`	Not
`!=`	Not equal
`==`	Is equal

Frequently Used Functions

`c`	Concatenate
`cbind, rbind`	Concatenate vectors
`min`	Minimum
`max`	Maximum
`length`	# values
`dim`	# rows, cols
`floor`	Max integer in
`which`	TRUE indices
`table`	Counts

`summary`	Generic stats
`Sort, order, rank`	Sort, order, rank a vector
`print`	Show value
`cat`	Print as char
`paste`	`c()` as char
`round`	Round
`apply`	Repeat over rows, cols

Statistical Functions

`rnorm, dnorm, pnorm, qnorm`	Normal distribution random sample, density, cdf and quantiles
`lm, glm, anova`	Model fitting
`loess, lowess`	Smooth curve fitting
`sample`	Resampling (bootstrap, permutation)
`.Random.seed`	Random number generation
`mean, median`	Location statistics
`var, cor, cov, mad, range`	Scale statistics
`svd, qr, chol, eigen`	Linear algebra

Graphical Functions

`plot`	Generic plot eg: scatter
`points`	Add points
`lines, abline`	Add lines
`text, mtext`	Add text
`legend`	Add a legend
`axis`	Add axes
`box`	Add box around all axes
`par`	Plotting parameters (lots!)
`colors, palette`	Use colors

Thanks !