R programming

Week 1.

Objects:

Numeric, i.e. 1, Inf (e.g. 1/0), NaN (not a number, e.g., 0/0 or a missing value)

Integer, i.e. L

Character

Complex

Logical (T/F)

A vector can only contain objects of the same class except the list

Attributes of an object can be accessed using the attributes ()

Function

The <- symbol is the assignment operator

# character indicates a comment.

The c() function can be used to create vectors of objects)

Objects can be explicitly coerced from one class to another using the as.\* functions. For example,

X <- 0:6

as.logical(x)

matrices are constructed column-wise, so entries can be thought of starting in the upper left corner and running down the columns

matrices creation: cbind(), rbind(),

factors are used to represent categorical data. Factors can be unordered or ordered. One can think of a factor as an integer vector where each integer has a label.

L() or glm()

X <- factor(c(“yes”,”no”,”yes”))

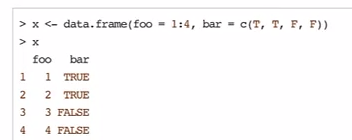
Levels = c(“yes”,”no”)

Missing values:

A NaN is a NA but the reverse is F

Matrices must have every element be the same class, while data frames can store different classes of objects in each column. Data frames are used to store tabular data, which has the same length.

Data frames are usually created by calling read.table() or read.csv()



R objects can also have names.

X <- 1:3

names(x) <- c(“A”,”B”,”C”)

x

names(x)

For list

For matrices

Dimnames(x) <- list(c(“a”,”b”),c(“c”,”d”))

Subsetting

[ ] always returns an object of the same class as the original, can be used to select more than one element.

x[1]; x[1:4]; u <- x>”a”, x[u];

$ is used to extract elements of a list or data frame by name

Matrices can be subsetted in the usual way with (i,j) type indices

By default, when a single element of a matrix is retrieved, it is returned as a vector of length 1 rather than a 1 x 1 matrix. This feature can be turned off.

Subsetting lists

x[]; x[[]],‘[[...]]’ is the operator used to select a single element, whereas ‘[...]’ is a general subscripting operator. Thus the former is the first object in the list Lst, and if it is a named list the name is not included. The latter is a sublist of the list Lst consisting of the first entry only. If it is a named list, the names are transferred to the sublist.

Reading Data

read.table (default separator is a space), read.csv (default separator is a comma) for reading tabular data.

readline, for reading lines of a text file

source, for reading in R code files (inverse of dump)

dget, for reading in R code files (inverse of dput)

load, for reading in saved workspace

unserialize, for reading single R objects in binary form

writing Data

write.table

writeLines

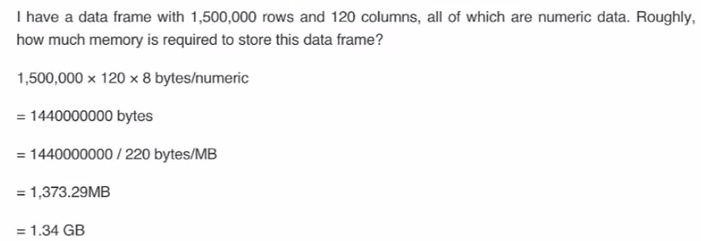
dump (multiple objects)

dput (single object)

save

serialize

when using R with larger datasets, it is useful to know a few things about your system.



Dumping and dputing are useful because the resulting textual format is edit-able, and in the case of corruption, potentially recoverable.

Data are read in using connection interfaces, e.g., file, or url,

Learning R: library (“swirl”), swirl()

In the S language, vectors with integer or real values are both of mode "numeric",

y <- c(1,2,3,NA)  
is.na(y) # returns a vector (F F F T)

x <- read.csv("hw1\_data.csv",sep = ",")

Week 2

Control structures:

If, else: testing a condition

For: execute a loop a fixed number of times

While: execute a loop while a condition is true

Repeat: execute an infinite loop

Break: break the execution of a loop

Next: skip an integration of a loop

Return: exit a function

Most control structures are not used in interactive sessions, but rather when writing functions or longer expressions.

If:

If (condition) {

# do something

} else {

# do something

}

For

For (i in 1:10) {

Print(i)

}

Nested for loops

For (i in 1:10) {

For (j in 1:10) {

Print(x[i,j])

}

}

While

Count <- 0

While (count < 10) {

Print (count)

Count <- count +1

}

While loops can potentially result in infinite loops if condition is not satisfied.

Repeat initiates an infinite loop; these are not commonly used in statistical applications but they do have their uses. The only way to exit a repeat loop is to call break.

Next

Next is used to skip an iteration of a loop

For (i in 1:100)

If (i <= 20) {

Next # skip the first 20 iterations

}

# do something here

}

Functions

x <- function (argument) {

# do something

}

For interactive work, function arguments can also be partially matched. The order of operations when given an argument is:

Check for exact match for a named argument

Check for a partial match

Check for a positional match

Lazy evaluation: arguments to functions are evaluated lazily, so they are evaluated only as needed.

Coding standards for R

1. Always use text files/ text editor
2. Indent you code, every indent is one space
3. Limit the width of your code (80 columns)
4. Fixing line length (80 columns) prevents lots of nesting and very long functions

Lm <- function (x) {x\*x}

How does R know what value to assign to the symbol lm? Why does not it give it the value of lm that is defined in the package?

When R tries to bind a value to a symbol, it searches through a series of environments to find the appropriate value. When you are working on the command line and need to retrieve the value of an R object, the order is roughly.

1. Search the global environment for a symbol name matching the one requested.
2. Search the namespaces of each of the packages on the search list

the global environment or the user’s workspace is always the first element of the search list and the base package is always the last.

The order of the packages on the search list matters.

Users can configure which packages get loaded on startup so you cannot assume that there will be a set list of packages available.

When a user loads a package with library function, the namespace of that package get put in position 2 of the search list by default and everything else gets shifted down the list.

Note that R has separated namespaces for functions and non-functions so it is possible to have an object named c and a function named c.

Scoping rules

The scoping rules for R are the main features that make it different from the original S language.

The scoping rules determine how a value is associated with a free variable in a function

R uses lexical scoping or static scoping. A common alternative is dynamic scoping.

Related to the scoping rules is how R uses the search list to bind a value to a symbol

Lexical scoping turns out to be particularly useful for simplifying statistical computations

f <- function (x,y) {

x^2 + y/z

}

Scoping rules determine how values are assigned to z

Lexical scoping, means that the values of free variables are searched for in the environment in which the function was defined.

What is an environment?

An environment is a collection of (symbol, value) pairs, i.e., x is a symbol and 3.14 might be its value.

Every environment has a parent environment; it is possible for an environment to have multiple children.

The only environment without parent is the empty environment.

A function + an environment = a closure or function closure.

If the value of a symbol is not found in the environment in which a function was defined, then the search is continued in the parent environment.

The search continues down the sequence of parent environment until we hit the top-level environment; this usually the global environment (workspace) or the namespace of a package.

After the top level environment, the search continues down the search list until we hit the empty environment. If a value for a given symbol cannot be found once the empty environment is arrived at, then an error is thrown.

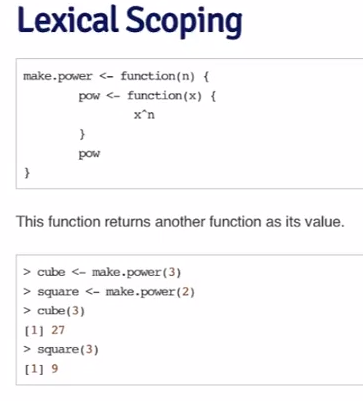
Why does all this matter?

Typically, a function is defined in the global environment, so that the values of free variables are just found in the user’s workspace.

This behavior is logical for most people and is usually the right thing to do

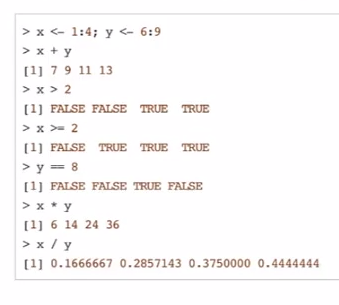
However, in R you can have functions defined inside other functions. Languages like C don’t let you do this.

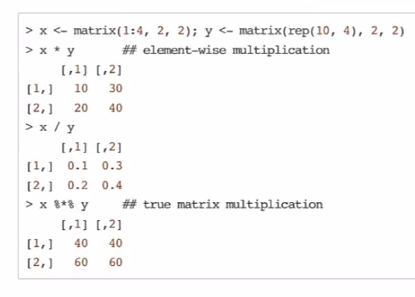
Now things get interesting, in the case the environment in which a function is defined is the body of another function.



Vectorized operation

Many operations in R are vectorized making code more efficient, concise, and easier to read.





Dates and times in R

R has developed a special representation of dates and times

* Dates are represented by the Date class, and can be coerced from a character string using the as.Date () function
* Times are represented by the POSIXct or the POSIXIt class
* Dates are stored internally as the number of days since 1970-01-01
* Times are stored internally as the number of seconds since 1970-01-01

Times in R are represented using the POSIXct or the POSIXIt class

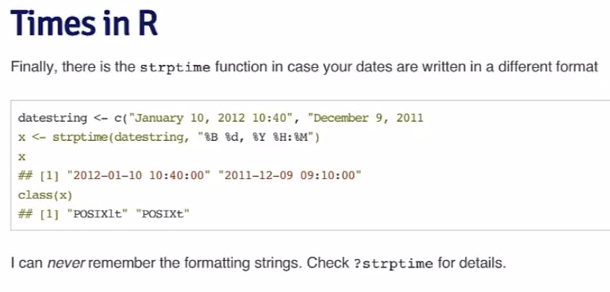
POSIXct is just a very large integer under the hood; it uses a useful class when you want to store times in something like a data frame

POSIXIt is a list underneath and it stores a bunch of other useful information like the day of the week, day of the year, month, day of the month.

There are a number of generic functions that work on dates and times

weekdays; months; quarters





“==” comparison operator

Summary

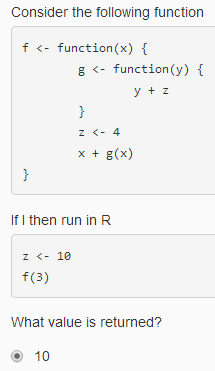
Dates and times have special classes in R that allow for numerical and statistical calculations

Dates use the Date class

Times use the POSIXct and POSIXlt class

Character strings can be coerced to Date/Time classes using the strptime function or the as.Date, as.POSIXlt, or as.POSIXct

Why 10?



parent.env is the environment in which a closure (e.g., function) is defined. parent.frame is the environment from which the closure was invoked/called.

 'if' can only test a single logical statement

Week 3

Loop function

Writing for, while loops is useful when programming but not particularly easy when working interactively on the command line. There are some functions which implement looping to make life easier.

Lapply: loop over a list and evaluate a function on each element.

Sapply: same as lapply but try to simplify the result

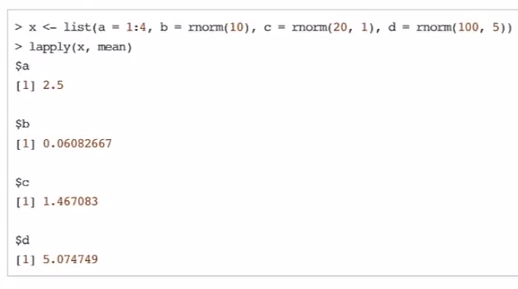
Apply: apply a function over the margins of an array

Tapply: apply a function over subsets of a vector

Mapply: multivariate version of lapply

An auxiliary function split is also useful, particularly in conjunction with lapply.

Lapply always returns a list, regardless of the class of the input.



Sapply will try to simplify the result of lapply if possible

If the result is a list where everylement is length 1, then a vector is returned. If the result is a list where every element is a vector of the same length (> 1), a matrix is returned. If it cannot figure things out, a list is returned.

Apply is used to evaluate a function (often an anonymous one) over the margins of an array.

It is most often used to apply a function to the rows or columns of a matrix

It can be used with general arrays, e.g. taking the average of an array of matrices

It is not really faster than writing a loop, but it works in one line.

Example

Apply (x, margin, function,…)

X is an array or matrix, margin is an integer vector indicating which margins should be retained

Function is a function to be applied

… is for other arguments to be passed to function

e.g. apply (x,1,mean)

rowSums = apply(x,1,sum)

rowMeans = apply(x,1,mean)

colSums = apply(x,2,sum)

colMeans = apply(x,2,mean)

x <- matrix(rnorm(200), 20, 10)

apply (x, 1, quantile, probs = c (0.25, 0.75))

tapply

function (x, index, fun, …, simplify = TRUE)

x is a vector, index is a factor or a list of factors, simplify, should we simplify the result? If not, a list is return

e.g. x <- c(rnorm(10), runif(10), rnorm(10,1))

f <- gl (3, 10)

tapply (x, f, mean)

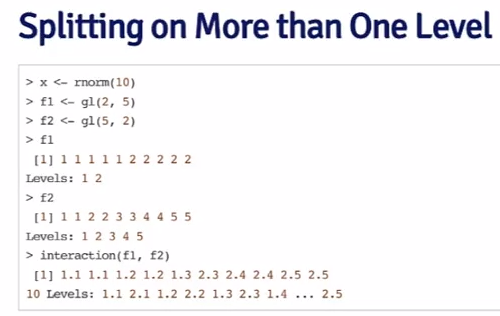
split

split(x, index, drop)

drop indicates whether empty factors levels should be dropped.

e.g. s <- split (x,x$Month)

gl (n,k) where n is an integer giving the number of levels, k is an integer giving the number of replications.



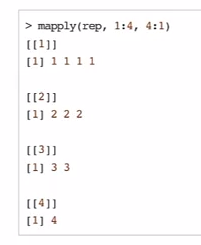
Mapply is a multivariate apply of sorts which applies a function in parallel over a set of arguments.

Mapply (fun, …, MoreArgs = Null, SIMPLIFY = True)

MoreArgs is a list of other arguments to FUN.

SIMPLIFY indicates whether the result should be simplified.

Rep (x, times)



Something’s wrong --- debugging

Message

Warning

Error

Condition

How do you know that something is wrong with you function?

What was your input? How did you call the function?

What were you expecting? Output, messages, other results

What did you get

How does what you get differ from what you were expecting?

Were your expectations correct in the first place?

Can you reproduce the problem (exactly)?

The primary tools for debugging functions in R are

* Traceback: prints out the function call stack after an error occurs; does nothing if there’s no error. It only gives you the recent error.
* Debug: flags a function for debug mode which allows you to step through execution of a function one line at a time
* Browser: suspends the execution of a function wherever it is called and puts the function in debug mode
* Trace: allows you to insert debugging code into a function a specific places
* Recover: allows you to modify the error behavior so that you can browse the function call stack

Week 4

Str: compactly display the internal structure of an R object

* A diagnostic function and an alternative to summary
* It is especially well suited to compactly display the (abbreviated) contents of (possibly nested) lists.
* Roughly one line per basic object

The goal of str function is what is in this object?

Str(str)

Str(lm)

Str(ls)

X <- rnorm (100,2,4)

Summary(x)

Str(x)

F <- gl (40,10)

Str(f)

Summary (f)

Simulation

Generating random numbers

Rnorm: generate random normal varieties with a given mean and standard deviation

Dnorm: evaluate the normal probability density (with a given mean/SD) at a point (or vector of points)

Pnorm: evaluate the cumulative distribution function for a normal distribution

Rpois: generate random Poisson variables with a given rate

Probability distribution functions usually have four functions associated with them. The functions are prefixed with a

* d for density
* r for random number generation
* p for cumulative distribution
* q for quantile function

dnorm (x, mean = 0, sd = 1, log = FALSE)

pnorm (q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

qnorm (p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

rnorm (n, mean = 0, sd = 1)

if Φ is the cumulative distribution function for a standard Normal distribution, then pnorm (q) = Φ(q) and qnorm = Φ-1(p).

e.g.

x <- rnorm (10, 20, 2)

summary (x)

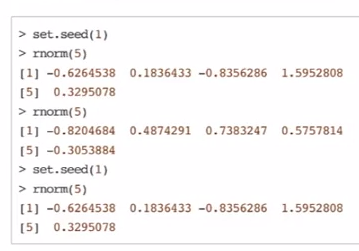
generating random numbers

setting the random number seed with set.seed ensures reproducibility

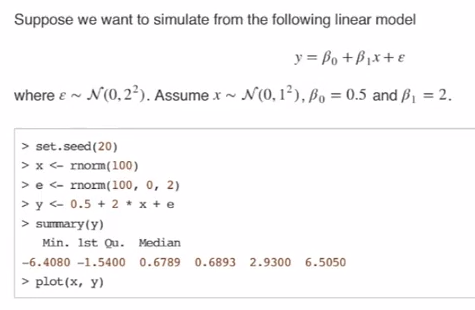
rnorm(5)

set.seed(1)

always set the random number seed when conducting a simulation.



Generating random numbers from a linear model



Random sampling

The sample function draws randomly from a specified set of scalar objects allowing you to sample from arbitrary distributions.

Set.seed (1)

Sample (1:10, 4, replace = TRUE)

Simulation

Summary

Drawing samples from specific probability distributions can be done with r\* functions

Standard distributions are built in: Normal, Poisson, Binomial, Exponential, Gamma, etc.

The sample function can be used to draw random samples from arbitrary vectors

Setting the random number generator seed via set.seed is critical for reproducibility.

Why is my code so slow?

Profiling is a systematic way to examine how much time is spend in different parts of a program

Useful when trying to optimize your code

Often code runs fine once, but what if you have to put it in a loop for 1,000 iterations? Is it still fast enough?

Profiling is better than guessing.

Getting biggest impact on speeding up code depends on knowing where the code spends most of its time

This cannot be done without performance analysis or profiling

Premature optimization is the root of all evil

General principles of optimization

Design first, the optimize

Remember: premature optimization is the root of all evil

Measure (collect data), do not guess

If you are going to be a scientist, you need to apply the same principles here.

Using system.time()

Computes the time (in seconds) needed to execute an expression

If there is an error, gives time until the error occurred

Returns an objects of class proc\_time

User\_time: time charged to the CPUs for this expression

Elapsed time: wall clock time

Usually the user time and elapsed time are relatively close, for straight computing tasks

Elapsed time may be greater than user time if the CPU spends a lot of time waiting around

Elapsted time may be smaller than the user time if your machine has multiple cores/processors (and is capable of using them)

Multi-threaded BLAS libraries

Parallel processing via the parallel package



Assume you already know where the problem is and can call system.time() on it

What if you do not know where to start?

The R profiler

The Rprof() function starts the profiler in R, R must be compiled with profiler support (but this is usually the case)

The summaryRprof() function summarizes the output from Rprof() (otherwise it is not readable)

Do not use system.time() and Rprof() together or you will be sad

Rprof() keeps track of the function call stack at regularly sampled intervals and tabulates how much time is spend in each function

Default sampling interval is 0.02 seconds

Note: if your code runs very quickly, the profiler is not useful, but then you probably do not need it in that case

The summaryRprof() function tabulates the R profiler output and calculates how much time is spend in which function

There are two methods for normalizing the data

By.total divides the time spend in each function by the total run time

By.self does the same but first subtracts out time spent in functions above in the call stack

Rprof() runs the profiler for performance of analysis of R code

summaryRprof() summarizes the output of Rprof() and gives percent of time spent in each function (with two types of normalization)

Good to break your code into functions so that the profiler can give useful information about where time is being spent

C or Fortran code is not profiled

Why is any of this information useful?

Optimization routines in R like optim, nlm, and optimize require you to pass a function whose argument is a vector of parameters (e.g., a log-likelihood)

However, an object function might depend on a host of other things besides its parameters (like data)

When writing software which does optimization, it may be desirable to allow the user to hold certain parameters fixed.

Lexical scoping summary

Objective functions can be built which contain all of the necessary data for evaluating the function

No need to carry around long argument lists – useful for interactive and exploratory work

Code can be simplified and cleaned up

Learning R in 153 min

Demo(graphics)

To use Bibtex in latex, use: citation (package = ‘package’)

To get help:

* Help(topic)
* Help.search(“topic”)
* Apropos(“tab”)
* RSiteSearch()

R 引入了命令自动补全功能，使用Tab 键能自动补全R 命令；或使用第二次Tab 后，返回所有可能的补全命令列表。

Q: 如何清除变量？

A: 清除单个变量使用rm() 函数，清除内存中所有的变量：rm( l i s t = l s ( a l l = TRUE) )

Q: 如何更改小数点后显示数字位数？

A: options(digits = ) ，digits 后面的参数为1 至22 的数字，

**Q: Windows** 下升级**R** ，但不想重装**packages** ？

鉴于R 的更新速度（大概每3 个月会有一次升级），频繁的更新主程序，可能是我们在

Windows 环境下最常遇到的问题，再加之本地Contributed Packages 数量很大，这个问题会更

加突出。一般来说，R 本地的目录结构相对固定，可以通过在其他目录下安装R ，再将旧版本保留的library 目录下的文件拷贝至新版本library 目录下或卸载R ，把R 装到旧的目录下，最后使

用如下命令更新：

update . packages ( c h e c k B u i l t = TRUE, ask = FALSE)

# or i n s t a l l . packages ( . packages ( a l l = TRUE) )

**24.** 我怎样保存自己的工作？

使用save.image() 函数。它将在R 的起始目录保存记􁗶区（working space）至.RData 文

件；或者使用save(..., file = ) 保存需要保存的R 对象。

**28.** 如何得到加载**Package** 的列表？

search() 函数返回当前加载的包的情况，使用

packages ( a l l . a v a i l a b l e = TRUE) 命令获得本地安装的包列表。当R 启动后，R 在内存中会自动加载若干Package：

**29.** 如何使用**R** 内置的数据集？

R 在datasets 包中共提供了102 个可以使用的数据集，这些数据集都可以通过data() 函数

加载入内存。

Everything in R is an object;

Every object is R has a class.

**32.** 如何得到函数的代码？

通常情况你只需要在R 平台下写出你需要查看的函数名，回车即可。比如：d i s t

但有时候这个函数可能是一个泛型函数（Generic Function），上面的方法就需要稍稍改进一下：

先使用methods() 函数来查看这个类函数的列表，找到具体需要的函数5 ，写出来，回车—问

题解决。

summary # I t i s a g e n e r i c f u n c i t o n

methods ( summary ) # l i s t o f the S3 methods

summary . lm # maybe you want to know the l i n e a r models ’ s summary

如果要究根问底，可以去下载源代码压缩包（\*.tar.gz，比如R-2.5.1.tar.gz）

**R** 里面可以使用科学计数法么？

1 e10 == 10000000000

1．2 e4 == 0.00012

**36. R** 可以读取其他统计软件录入的数据么？

可以，使用foreign 包，它可以读取Minitab, S, SAS, SPSS, Stata, Systat, dBase 保存的数

据。或者使用Hmisc 包中sas.get, spss.get 函数来获取SAS 和SPSS 软件格式数据。但随着这些商业软件的版本更新，依旧会出现一些文件不能通过foreign 包读取的情况，建议在交换数据时尽量使用.csv 这种通用格式。

将Excel 另存为csv(Comma Separated Values) 文件，使用read.csv() 函数读取（推荐）；

**38.** 可以将**R** 中显示的结果输出到文件么？

可以。使用sink()函数。

data (CO2)

s i n k ( ”CO2. t x t ” )

CO2

s i n k ( ) # go to your work d i r e c t o r y , you w i l l get CO2. t x t

R 提供了capture.output() 函数，这个函数可以将R 的输出信息转化为字符或文件。

glmout <*-* capture . output ( example ( glm ) )

glmout [ 1 : 5 ]

**40. R** 可以从内存直接读写数据么？

可以。拷贝需要读取的内容，使用

data <*-* read . t a b l e ( ” c l i p b o a r d ” )

w r i t e . t a b l e ( ” c l i p b o a r d ” )

这个技巧对于临时性的分析工作非常有益。

**43.** 为什么当我使用**source()** 时，不能显示输出结果？

对需要显示输出的对象使用print() ，或者使用source(file, echo = TRUE)。如果R 代码里面包含sink() 之类的函数，必须使用source(file, echo = TRUE) 才能得到正确的输出结果，否则sink 的对象将为空。

**44. R** 可以输出可供**TEX** 使用的文本么？

可以，参考Hmisc 包中的latex() 函数和xtable 包中的xtable()函数。

a <*-* matrix ( 1 : 6 , nr=1) # r e q u i r e ( x t a b l e )

2 colnames ( a ) <*-* p a s t e ( ” c o l ” , 1 : 6 )

x t a b l e ( a )

xtable() 函数可以用于产生HTML 格式的原码，这样R 生成的表格就可以非常方便、漂亮地插入到word、powerpoint 这类文字处理软件。输出LATEX 格式的表格还可以quantreg 包中latex.table()函数。RMySQL 6 包中提供了到MySQL 数据库的接口；RODBC 包提供了更为广泛数据库接口的解决方案— 支持所有标准ODBC 接口的数据库。通过这种方式，相同的R 代码可以方便地应用于不同类型的数据库。

l i b r a r y (RODBC)

ch <*-* odbcConnect ( ” stocksDSN ” , uid = ” myuser ” ,pwd = ” mypassword ” )

s t o c k s <*-* sqlQuery ( ch , ” s e l e c t \* from quotes ” )

odbcClose (ch)

在R 中使用NA（not available）表示缺失值，要注意R（S）语言中NA 同样是一个逻辑值，

x <*-* NA

x > 3

c l a s s ( x )

故当判断是否相等时不能使用 x == NA来判断缺失值。而是使用函数is.na()来判断是否为缺失值，使用x [ ! i s . na ( x ) ]删除缺失值。

如何对不规则数组进行统计分析？

参考tapply() ：

n <*-* 1 7 ; f a c <*-* f a c t o r ( rep ( 1 : 3 , l e n = n ) , l e v e l s = 1 : 5 )

t a b l e ( f a c )

tapply ( 1 : n , fac , sum)

tapply ( 1 : n , fac , mean )

## or r e v e r s e a l i s t

to <*-* l i s t ( a = 1 , b = 1 , c = 2 , d = 1)

tapply ( to , u n l i s t ( to ) , names )

tapply() 的常见于方差分析中对各个组别进行mean、var（sd）的计算。说到概要统计，不得不

说另外一个函数aggregate()，它将tapply() 函数对象为向量的限制扩展到了数据框。

attach ( warpbreaks )

tapply ( breaks , l i s t ( wool , t e n s i o n ) , mean )

a g g r e g a t e ( breaks , l i s t ( wool , t e n s i o n ) , mean )

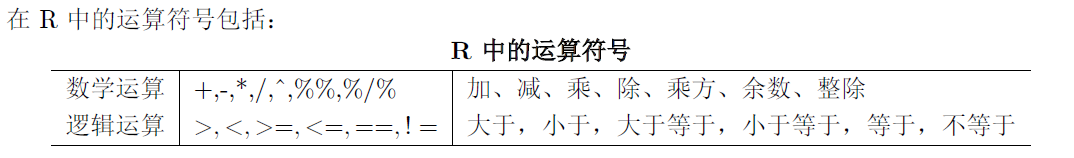
## from the help

a g g r e g a t e ( s t a t e . x77 ,

l i s t ( Region = s t a t e . region, Cold = s t a t e . x77 [ , ” Frost ” ] > 1 3 0 ) ,mean )

**61.** 一组数中随机抽取数据？

函数sample()



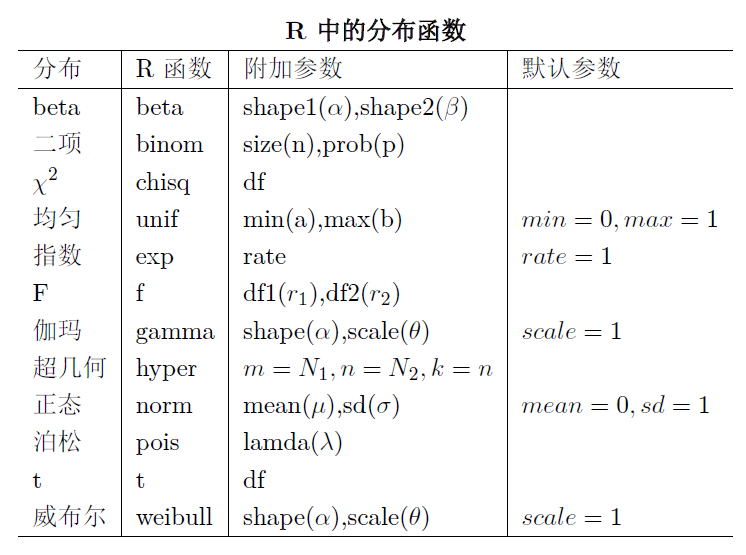
**73.** 如何计算组合数或得到所有组合？

choose() 用于计算组合数

(*n k*)，函数combn() 可以得到所有元素的组合。使用factorial() 计算阶乘。

**74.** 如何在**R** 里面求（偏）导数？

使用函数D()



**115.** 有没有直接计算峰度和偏度的函数？

当然自己写一个也费不了太多时间。FBasics 包中提供了

skewness ( )

k u r t o s i s ( )

lm() 函数提供了线性回归的计算方法。

lm . s w i s s ,<-lm ( F e r t i l i t y ~ . , data = s w i s s )

lm() 的结果是一个包含回归信息的列表，它包含以下信息：

coefficients：回归系数（矩阵）

residuals：返回模型残差（矩阵）

fitted.values：模型拟合值

…：…

可以使用如下命令得到列表名称:

names ( lm . s w i s s )

summary() 和anova() 分别返回回归模型的概要信息和方差分析表。

# the same as summary. lm ( )

summary ( lm . s w i s s )

anova ( lm . s w i s s )

提取模型信息的类函数有很多，其他可以参考R-intro 中Statistical models in R 一节。

如果处理数据的量很大，可以使用biglm 包中的biglm() 函数。这个函数可以用于“海量”

数据的回归模拟。

**116.** 如何做交叉列联表？

table() 函数。table(x) 为x 的频数表；table(x,y) 为交叉列联表。

x <*-* with ( a i r q u a l i t y , t a b l e ( cut (Temp, q u a n t i l e (Temp) ) , Month ) )

prop . t a b l e ( x , 1 )

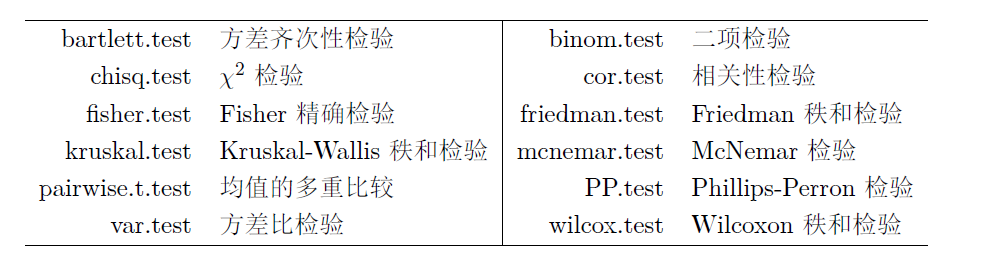
**118.** 如何更新模型？

参考update() 函数：

1 summary ( f 0 <*-* lm ( F e r t i l i t y ~ . , data = s w i s s ) )

f 1 <*-* update ( f0 , . ~ . *􀀀* Examination )

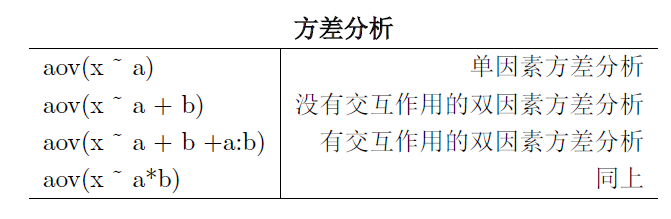
3 summary ( f 1 )

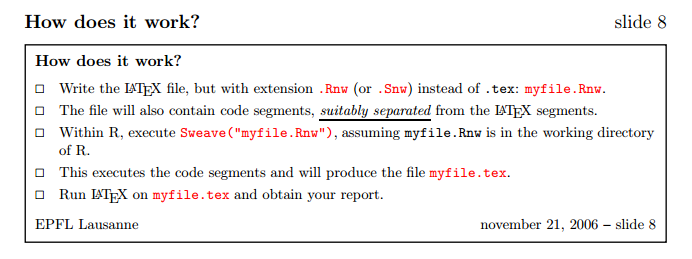


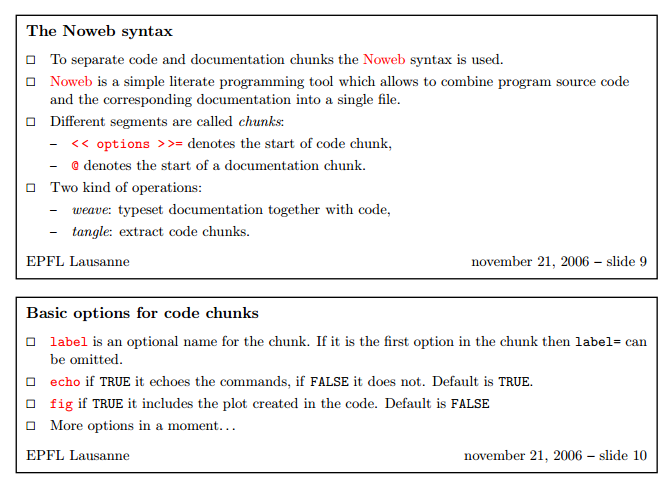
**129.** 如何使用方差分析（**ANOVA**）？

方差分析同线性回归模型很类似，毕竟它们都是线性模型。最简单实现方差分析的函数为

aov()，通过规定函数内公式形式来指定方差分析类型：







Difference between probability and density functions?