# R使用经验汇编

李东风
LMAM, 北京大学数学科学学院
E-mail: ldf@math.pku.edu.cn

2010-12-10

## 目 录

## 摘 要

本文总结了R运用中一些问题的解决方法，大多是只是提示用什么函数或包，更具体的做法需要去查手册。这是作为入门教材的补充，不能用作入门教材。

# §1 R使用问题

## §1.1 与Emacs配合使用

安装Emacs的ESS包。然后，在自己的设置文件中，设置

```
(setq-default inferior-R-program-name
              "c:\\R\\R-2.9.0\\bin\\Rterm")
```

其中的路径应该换成自己安装R的路径。

## §1.2 重要链接

R主页: `http://www.r-project.org/`.

R-Forge: `http://r-forge.r-project.org/` 各种R附加软件包的存放地点。

为了查找解决某类问题的相关函数，可以安装`sos`包。安装后用`findFn(字符串)`的方法查找关键字或函数名，结果显示在浏览器中。

## §1.3 计时和运行记录

用`system.time`(表达式)计算一个表达式的运行时间；用`proc.time`得到当前时间，两次调用的差表示时间间隔。

数据分析运行中间可能输出大量的信息，在调试程序时可能需要查看这些信息。在Emacs中用ESS支持直接在Emacs窗口中运行R当然可以保存这些输出，但是在Emacs中运行如果出错（尤其是调用自己编写的C程序的错误）不容易恢复。用`sink()`函数可以保存输出到文件，但这时在屏幕上就看不到输出。其实，`sink()`函数加一个`split=TRUE`选项就可以实现既输出到文件同时又显示在屏幕的效果。如果图形也能作到这点就好了。

## §1.4 Linux中非超级用户安装包

假设你在自己的主目录下建立了`.R/library`子目录，并建立了一个`~/.Renviron`文件，其中包含如下行:

```
R_LIBS="~/.R/library"
```

则可以把R的附加包安装在自己的目录中:

```
R CMD INSTALL -l ~/.R/library mypackage.tar.gz
```

进入R后可以直接访问在`~/.Renviron`中定义在`R_LIBS`目录中安装的包。

# §2 基本编程

## §2.1 一些S语言规则

.Last.value保存上一个计算但未赋值的表达式的值。

S数据的基本属性为mode和length。

为了删除某一属性，可以把此属性赋值为NULL，如去掉了x的name属性。

用as.xxx()函数转换类型，用is.xxx()函数测试类型。如as.numeric(x), is.numeric(x)。

用args(func)查询一个函数func的参数表。

## §2.2 工作空间管理

R中在命令行定义的变量保存在工作空间中，用ls函数可以列表查看。ls支持一个pattern=参数，可以给出查询的变量名模式，需要用正则表达式（见??），但也可以用glob2rx 把通配符"*"和"?"转换为正则表达式。比如，为了查询所有以"tmp."开头的变量名，可以用

```
> ls(pattern="^tmp[.]")
[1] "tmp.abc" "tmp.x"
```

用rm函数删除若干个对象，rm(list=)可以用一个包含对象名的字符型向量指定要删除的对象。

用object.size求得对象占用空间大小，单位是字节。下面定义的函数可以查看工作空间中的所有对象和大小：

```
ls.sizes <- function(){
  li <- ls(parent.frame())
  si <- numeric(length(li))
  for(ii in seq(along=si)){
    si[ii] <- object.size(get(li[ii], parent.frame()))
  }
  ord1 <- rev(order(si))
  res <- si[ord1]
  names(res) <- li[ord1]
  print(res)
  invisible(res)
}
```

R中查找变量除了对应于命令行的工作空间外，还可以用attach把数据框或列表的变量引入其“搜索路径”。用find 可以查找在搜索路径不同位置的变量名。用conflicts可以查找在不同搜索路径位置重名的对象。

使用attach要特别谨慎，用完后应该立即用detach断开，否则容易造成名字冲突。

## §2.3 数据类型

### §2.3.1 向量

S中下标功能丰富而强大，可以用正整数下标、负整数下标、逻辑向量下标、字符串下标。

读取时下标超过最后元素位置返回NA，写入时下标超过最后元素位置将添加新元素，中间空隙填NA。

函数replace可以把向量某部分替换后返回，但不改变原始变量；函数append在向量后面添加元素后返回，但不改变原始变量。

### §2.3.2 矩阵和数组(array)

矩阵和数组的下标除了可以用向量下标的各种模式以外，还可以

- 空置，这代表这一下标的所有值；
- 用一个$k$列的矩阵作为下标，$k$为数组维数，矩阵的每一行是一个元素的下标。

矩阵和数组所有维的字符串下标保存在列表dimnames中，每个元素是一维的字符串下标（名字）。某些维可以取NULL。

使用矩阵和数组的子集时要特别注意的是，某一维只有一个下标点时，该维会被舍弃，比如若A是一个矩阵, 则A[,1]是一个向量，而不再是矩阵或数组。如果下标是计算得到的集合则有可能再个别情况下集合只有一个元素，数组降维从而造成程序错误。可以用如A[,ind,drop=FALSE] 这样的办法避免下标降维。

a[]返回数组a的所有元素，是按列优先格式排序的（FORTRAN格式）。

函数nrow和ncol返回矩阵的行数和列数，注意函数名没有用复数。函数row(A)和col(A)返回和矩阵A 相同形状的矩阵，分别为对应元素的行号和列号。如

```
> A <- rbind(1:3, 4:6); A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> row(A)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
```

```
> col(A)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
```

比如我们表示矩阵A的上三角元素，可以用A[col(A) >= row(A)], 元素排列次序是按列拉直的。矩阵的下三角元素可以用lower.tri(A)表示。

对于数组，row和col的替代函数为slice.index(A,k)，其中k为某一维序号，将产生和A形状相同的数组，结果元素为原数组对应位置元素的第k下标值。

### §2.3.3 列表

c函数除了连接向量以外也可以连接列表。如果连接列表时加入一个recursive = TRUE选项，则可以把结果变成一个向量。如

```
> c(list(1:2,3:4), list(11:12,13:14))
[[1]]
[1] 1 2

[[2]]
[1] 3 4

[[3]]
[1] 11 12

[[4]]
[1] 13 14

> c(list(1:2,3:4), list(11:12,13:14), recursive=TRUE)
[1]  1  2  3  4 11 12 13 14
```

对于列表，可以用unlist转换为向量。如

```
> unlist(list(x=1, y=2:3))
 x y1 y2
 1  2  3
```

### §2.3.4 因子

对于因子，用factor生成，在读入数据框时缺省情况下字符型值会被转换为因子。为了保证因子水平为指定次序，可以在调用factor时加leves=选

项。用table(f)可以对因子f进行频数统计。

用ordered函数生成有序因子，最好用levels=指定水平。函数cut可以对输入的数值型向量离散化为有序因子，用breaks指定分组数或分隔点。

用unclass去除变量的类，比如对一个因子unclass(f)会输出其编码的整数值。

gl(n,k,length)指定一个因子水平有n个，每个水平连续重复k次，总长度为length。可以用labels选项指定水平名。如

```
> gl(3, 2, length=12)
 [1] 1 1 2 2 3 3 1 1 2 2 3 3
Levels: 1 2 3
```

为了产生两个因子的所有组合并重复，可以对每个因子使用gl函数。设因素$A$有三个水平，因素$B$有两个水平，为了完全试验并且重复二次，总共需要12次试验，可以用

```
> A <- gl(3, 4, labels=c("A1", "A2", "A3"))
> B <- gl(2, 2, length=12, labels=c("B1", "B2"))
> cbind(A, B)
      A B
 [1,] 1 1
 [2,] 1 1
 [3,] 1 2
 [4,] 1 2
 [5,] 2 1
 [6,] 2 1
 [7,] 2 2
 [8,] 2 2
 [9,] 3 1
[10,] 3 1
[11,] 3 2
[12,] 3 2
```

### §2.3.5 数据框

用data.frame生成数据框时，为了使字符型变量不被自动转换为因子，一种办法是用I()函数保护起来。可以用stringsAsFactors=FALSE选项，但这样所有字符型列都不转换。

用str(d)查看数据框d的基本信息。可以看到观测数、变量数、变量类型、不同值列表等。用summary(d)进行变量的基本概括。

### §2.3.6 R的日期类

Date是R的日期类，实际保存为实数值。转换如d=as.Date("1996-03-17", format="%Y-%m-%d"), 而显示可用如format(d, format="%Y-%m-%d")。格式中%Y是四位的年份，%m是数字的月份，%d是日期，%y是两位的年份，%b是英文所写的月份，%B是英文全称的月份。参见strptime的帮助。

顺便提一句，在SAS中如果要把如s="1997-03-15"这样的字符串值转换成SAS日期，方法是date=input(s, yymmdd10.)。

## §2.4 数学功能

### §2.4.1 四则运算

round(x)作四舍五入取整，round(x,2)四舍五入到两位小数，round(x,-2)四舍五入到百位。

用%/%整除，用%%计算余数。$x \div y$结果为商x %/% y，余x %% y，总有x = (x %/% y)*y + (x %% y)成立。

### §2.4.2 数学函数

常见函数如abs, sign, log, log10, sqrt, exp, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh。还有gamma, lgamma(伽玛函数的自然对数)。

sum对向量求和, prod求乘积，cumsum和cumprod计算累计，得到和输入等长的向量结果。mean计算均值，var计算样本方差，sd计算样本标准差。

max和min求最大和最小，cummax和cummin累进计算。range返回最小值和最大值两个元素。max, min, range 如果有多个自变量可以把这些自变量连接起来后计算。

pmax(x1,x2,...)对若干个等长向量计算对应元素的最大值，不等长时短的被重复使用。pmin类似。比如，pmax(0, pmin(1,x))把x限制到$[0,1]$内。

### §2.4.3 排序

sort返回排序结果。可以用decreasing=TRUE选项进行降序排序。sort可以有一个partial=选项，这样只保证结果中partial=指定的下标位置是正确的。比如，

```
> sort(c(3,1,4,2,5), partial=3)
[1] 2 1 3 4 5
```

只保证结果的第三个元素正确。可以用来计算样本分位数估计。选项na.last指定缺失值的处理，取NA则删去缺失值，取TRUE则把缺失值排在最后面，取FALSE则把缺失值排在最前面。

order返回排序用的下标序列, 它可以有多个自变量，按这些自变量的字典序排序。可以用decreasing=TRUE选项进行降序排序。如果只有一个自变量，可以使用sort.list函数。

rank计算秩统计量，可以用ties.method指定同名次处理方法，如ties.method="min"取最小秩。

order, sort.list, rank也可以有na.last选项，只能为TRUE或FALSE。

rev反转序列。

### §2.4.4 规则序列

用冒号运算符或seq生成等差数列。

用rep重复一个序列，如

```
> rep(0,3)
[1] 0 0 0
> rep(1:2, 3)
[1] 1 2 1 2 1 2
```

rep的第二个参数可以指定每个元素重复的次数，如

```
> rep(1:2, c(3,5))
[1] 1 1 1 2 2 2 2 2
```

rep可以用一个each=选项指定对所有元素分别重复相同次数，如

```
> rep(1:2, each=4)
[1] 1 1 1 1 2 2 2 2
```

### §2.4.5 集合函数

- union(x, y) 并集；
- intersect(x, y) 交集；
- setdiff(x, y) 差集；
- setequal(x, y) 两个集合是否同一集合；
- is.element(el, set) 属于关系。

### §2.4.6 fft的数学公式

R中fft函数可以进行离散傅立叶变换。设x为长度n的向量，y=fft(x)，则

```
y[k] = sum(x * complex(argument = -2*pi * (0:(n-1)) * (k-1)/n))
```

即

$$w_k = \frac{k-1}{n}, k = 1, 2, ..., n$$
$$y_k = \sum_{j=1}^{n} x_j \exp(-i2\pi w_k(j-1)).$$

注意没有除以$n$。另外，若`y=fft(x)`, `z=fft(y, inverse=T)`, 则`x == z/length(x)`。

### §2.4.7 Kronecker积

用`A %x% B`表示。

### §2.4.8 优化

用`optim`, `nlm`, `optimize`函数。类似作用的有`uniroot`(一元方程求根)，`diriv`(数值微分)，`constrOptim`(约束优化)。

### §2.4.9 数值积分

一元用`integrate`，多元用adapt包中的`adapt`函数。

### §2.4.10 代替循环的函数

对矩阵用`apply`，对列表用`lapply`和`sapply`，对向量用`sapply`, 分组处理用`tapply`。

错误处理在options函数调用中设error=recover可以使得出错时进入调试状态。设warn=1可以使警告信息输入不滞后。设warn=0取消警告信息，而warn=2使警告信息变成错误。

### §2.4.11 表格数据的简单操作

`table()`计算频数。

`tapply(x, fac, func)`按`fac`对`x`分组在每组内计算函数`func`。`fac`还可以是由若干因子向量组成的列表，这时将交叉分类后计算`func`函数。

几个因子完全组合的结构数据集可以用`expand.grid(li)`实现，`li`是一个列表，每个列表变量包含一个因子的水平。如果`ll`是某个包含若干因子变量的列表，则expand.grid(lapply(ll, levels))可以构造所有可能水平组合的数据框。

## §2.5 数据框的一些简单处理

### §2.5.1 数据框合并

用`merge` 函数按照某关键列横向合并。如果直接按行号合并可以用`cbind`。

纵向合并直接用rbind。

## §2.6 与其它系统的接口

### §2.6.1 在C++/C中使用NA值

有时需要把NA传到C++/C中。为此，C中要包含R.h头文件，并用ISNA(x[0])这样的方法判断x[0]是否为NA，编译时最好用标准的

```
R CMD SHLIB yourcprog.cpp
```

方法编译。在R中调用时.C中要加上NAOK=TRUE选项。

### §2.6.2 在为R准备的C程序中使用R的随机数发生器

要调入<Rmath.h>头，在生成随机数前调用GetRNGstate()，生成完毕调用SetRNGstate()。实际产生一个正态分布随机数如rnorm(0.0, 1.0)。编译用

```
R CMD SHLIB mycode.c
```

## §2.7 目录和文件

- 用getwd()查询当前工作目录，用setwd('subdir')设置当前工作目录。
- 调用list.files()或dir()可以查看目录中内容，用list.files(pattern='.*[.]r$')可以列出所有以“.r”结尾的文件。
- file.info(filenames)可以显示文件的详细信息。
- create.dir()新建目录。
- file.exists()查看文件是否存在。
- file.create()生成文件。
- file.remove()或unlink()删除文件。unlink()可以删除目录。
- file.rename()为文件改名。
- file.append()把两个文件相连。
- file.copy()复制文件。
- file.access()考察文件的访问权限。
- file.path()把目录和文件名组合得到文件路径。
- basename()和dirname() 从一个全路径文件名获取文件名和目录。

# §3 数据输入输出和管理

## §3.1 数据访问

### §3.1.1 R对象数据的读写

R本身也可以作为一个对象数据库使用。为了把工作空间中的变量保存到文件中，可以用如

```
save(x, y, file="保存的文件名.RData")
```

的办法，x, y是要保存的变量。为了读入已保存的变量，用如

```
load("保存的文件名.RData")
```

用dump函数可以把一些R对象保存为一个S源程序文件，只要在R中用source运行就可以恢复这些对象。dump用法为

```
dump(list, file = "dumpdata.R", append = FALSE)
```

list是包含要保存的R对象名的字符型向量

### §3.1.2 文本格式文件的读写

用scan, read.table, read.csv, read.delim, read.fwf等函数读取。用read.table读入的数据如果没有行名，有列名，应该使用row.names=NULL, header=TRUE选项；可以用na.string=选项指定代表缺失值的字符串; 可以用skip=指定需要略过的行数。要注意在Windows中文件名路径中如果用反斜杠\需要用两个反斜杠表示。

为了读入数据文件中如下的一个矩阵:

```
1 2 3
4 5 6
```

可以用scan如

```
A <- matrix(scan("文件名"), col=3, byrow=TRUE)
```

scan也可以有skip选项。scan加一个what=列表选项可以实现类似read.table的效果。比如对如下文件：

```
姓名 班级 分数
李明  1  87
```

```
张颖  2  80
刘明  1  69
```

可以读入如

```
> dl <- scan("data01.txt", skip=1,
    what=list("姓名"="", "班级"="", "分数"=0)); dl
Read 3 records
$姓名
[1] "李明" "张颖" "刘明"

$班级
[1] "1" "2" "1"

$分数
[1] 87 80 69
```

注意读入结果是一个列表而不是数据框，但可以用as.data.frame(dl)转换。

scan有一个multi.line=TRUE选项，与what=列表配合允许一个记录占用多行。

用write.table, write.csv把数据框写成文本格式的数据文件。

要把矩阵按其显示格式写入一个文本文件，办法如

```
write(t(A), file="文件名", ncolumns=ncol(A))
```

对于巨大文件，先用file打开，然后用readLines分批读入，或用scan加what参数和nlines参数分批读入。

可以读写系统剪贴板中的表格数据。用法如

```
read.table("clipboard")。
write.table(x, file="clipboard")
```

### §3.1.3 Excel数据读写

基本办法是转换为其他格式如CSV格式再读。在Windows下有xlsReadWrite包，可以读写.xls文件。RODBC包也可以支持Excel文件的读写。xlsx包利用一个Java API实现.xlsx文件的读写。RExcel包实现了R与Excel的互操作。

小量数据可以选中一个矩形区域复制，然后在R中用

```
myDF <- read.delim("clipboard")
```

也可以从R中复制数据集:

```
## export 'iris' data to clipboard
write.table(iris, "clipboard", sep = "\t", col.names = NA)
## then open up MS Excel and paste (ctrl-v) in iris data
```

**用ODBC读取Excel文件** 下面的例子从Excel文件Test.xls中读取一个工作簿Sheet1。

```
library(RODBC)
MyExcelData <- sqlFetch(odbcConnectExcel("Test.xls"),
      sqtable = "Sheet1", na.strings = "NA", as.is = T)
odbcCloseAll()
```

RODBC的odbcConnectExcel()函数可以打开对一个Excel文件的连接，返回连接对象con。如果加上readOnly=FALSE选项可以生成新Excel文件或修改已有Excel文件。sqlTables(con)可以返回Excel文件中各个工作簿的信息的数据框，其中TABLE_NAME列为表名。一个测试读取的程序如下：

```
require(RODBC)
fname <- "testread.xls"
con <- odbcConnectExcel(fname)
on.exit(close(con))

tables.info <- sqlTables(con)
print(tables.info)
## $TABLE_NAME是需要的工作簿名单。

## 取出第一个表的SQL语句。
query <-
  paste("SELECT * FROM `",
        tables.info$TABLE_NAME[1], "`",
        sep="")
d1 <- sqlQuery(con, query)
cat("===== 从Excel中读取的第一个工作簿: ====\n")
print(d1)

## 取出第二个表的SQL语句。
query <-
```

```
  paste("SELECT * FROM `",
        tables.info$TABLE_NAME[2], "`",
        sep="")
d2 <- sqlQuery(con, query)
cat("===== 从Excel中读取的第二个工作簿: ====\n")
print(d2)
```

RODBC用sqlSave保存一个表，对Excel表在用odbcConnectExcel打开时需要加上readOnly=FALSE选项。下面是一个生成包含两个工作簿的Excel文件的例子：

```
require(RODBC)

fname <- "testwrite.xls"
if(file.exists(fname)) file.remove(fname)
con <- odbcConnectExcel(fname, readOnly=F)
on.exit(close(con))

## 测试数据集
d1 <- data.frame(id=c("a", "b", "c"),
                 x=1:3, y=11:13)
names(d1) <- c("学号", "数学", "语文")
d2 <- data.frame(id=c("x", "y", "z"),
                 x=101:103, y=211:213)
names(d1) <- c("学号", "数学", "语文")

## 存入表一。
res <-
  sqlSave(con, d1, tablename="理科成绩",
          rownames=F, colnames=F,
          safer=T)
## 存入表二。
res <-
  sqlSave(con, d2, tablename="文科成绩",
          rownames=F, colnames=F,
          safer=T)
```

**用xlsReadWrite的例子** xlsReadWrite包可以把数据框直接存取为Excel文件的工作簿。

```
library( xlsReadWrite )
### create some test bike data
tdat <- data.frame( Price = c( 6399, 3699, 2499 ),
  Amount = c( 2, 3, 1 ),
Date = c( 39202, 39198, 39199 ),
= c( "Pro machine", "Road racer", "Streetfire" ) )
### write
write.xls( tdat, "bikes.xls" )
### read and check: read as data.frame
bikes1 <- read.xls( file = "bikes.xls" )
if (!identical( tdat, bikes1 )) stop( "oops, not good..." )
# read as data.frame (custom colnames, date as iso-string)
bikes2 <- read.xls( file = "bikes.xls",
 c( "", "CHF", "Number", "Date" ),
from = 2, colClasses = c( "numeric", "numeric", "isodate" ) )
if (!all( tdat$Date == isoStrToDateTime( bikes2$Date ) ))
s, not good..." )
# read as matrix
bikes3 <- read.xls( file = "bikes.xls", type = "double" )
if (!identical( as.matrix( tdat ), bikes3 ))
   stop( "oops, not good..." )
```

### §3.1.4 Access表读写

假设有Access数据库在文件c:/Friends/birthdays.mdb中，内有两个表Men和Women，每个表包含域Year, Month, Day, First Name, Last Name, Death。域名应尽量避免用空格。

例1：读入女性所有信息:

```
require(RODBC)
con <- odbcConnectAccess("c:/Temp/birthdays.mdb")
women <- sqlQuery(con, 'select * from Women')
print(women)
close(con)
```

例2：显示数据库目录

```
require(RODBC)
con <- odbcConnectAccess("c:/Temp/birthdays.mdb")
# Table names. Ignore those starting with 'MSys'
sqlTables(con)$"TABLE_NAME"
# Column names in a table
sqlColumns(con, "Women")$"COLUMN_NAME"
close(con)
```

例3：较复杂的查询

```
require(RODBC)
con <- odbcConnectAccess("c:/Temp/birthdays.mdb")
# Read Year, Last Name and First Name.
# Rename Last Name to LastName, First Name to FirstName
# Read first 5 rows where First Name is not 'Carl'
men <- sqlQuery(con, Paste("select top 5 Year, ",
  "[First Name] as FirstName, ",
  "[Last Name] as LastName from Men ",
  "where [First Name]<>'Carl'")
close(con)
```

## §3.2 输出

### §3.2.1 输出格式控制

用format把对象各元素格式化为相同格式转化为字符型。用formatC, sprintf使用C语言的格式规定转化输出。

### §3.2.2 输出为LaTeX表

用Hmisc包中的latex函数，如latex(summary(lm(y~x)))。加NA=""可以把缺失值显示成空白，输出数据集时，行名将作为第一栏。

还可以用xtable包中的xtable函数。

### §3.2.3 输出为HTML

用R2HTML包。定义了一系列输出HTML内容的函数，可以组装一个HTML输出文件。还可以类似sink函数，把交互运行的一个会话输出为带链接的HTML结果。可以把结果表格都先生成，存为R的对象，然后用不同的输出函数，输出为不同格式，如HTML, LaTeX。

## §3.3 文件

输入输出可以针对命令行，针对文件，R支持扩展的文件类型，称为“connection”。用open命令打开文件，返回一个句柄，用close关闭打开的文件。在打开的时候文件可以顺序访问。打开连接的函数有

```
file("path", open="", blocking=T,
     encoding = getOption("encoding"), raw = FALSE)

url(description, open = "", blocking = TRUE,
    encoding = getOption("encoding"))

gzfile(description, open = "",
       encoding = getOption("encoding"),
       compression = 6)

bzfile(description, open = "", encoding = getOption("encoding"),
       compression = 9)

xzfile(description, open = "", encoding = getOption("encoding"),
       compression = 6)

unz(description, filename, open = "",
    encoding = getOption("encoding"))

textConnection(description, open="r", local = FALSE,
               encoding = c("", "bytes", "UTF-8"))
```

file打开普通文件，url打开一个网址进行读取，gzfile, bzfile, xzfile读写压缩文件，gzfile既可以用于gzip压缩的文件，也可以用于普通非压缩文本文件或用bzip2, xz, lzma压缩的文件。unz从ZIP文件中读取一个文件。

调用这些函数时并不打开文件，用open可以打开这些函数的返回结果。如果直接从这些函数的结果读写，则该读写函数打开连接，进行读写，然后关闭连接。对于用open打开的连接则读写函数不会自动关闭连接。

连接的open参数可以取”r”或”rt”(读取文本), ”w”或”wt”(写入文本), ”a”或”at”(添加文本), ”rb”, ”wb”, ”ab”为二进制格式读、写、附加，”r+”, ”r+b”可以同时进行读和写，”w+”, ”w+b”可以同时进行读和写，开始时截断文件内容，”a+”, ”a+b”可以同时进行读和附加。

使用特殊文件名"clipboard"可以读写系统剪贴板。

可以把一个字符串作为输入源用textConnection函数打开，当作一个文件读取。

为了把一个字符串当作文件来写入，用textConnection('变量名', 'w', local=TRUE)生成一个字符串变量用于保存输出各行，关闭句柄后可以在指定的变量中找到输出的各行。

用readLines和writeLines可以读写若干行。

### §3.4 数据汇总

可以用table函数进行单个分类变量或多个分类变量的频数统计。更复杂的任务，一类针对数组和列表，包括apply, sweep, mapply, sapply, lapply; 另一类针对数据框，包括aggregate, by等。

#### §3.4.1 table函数

table函数接受单个向量，或多个向量组成的列表或数据框，每个向量都用于分组，只有一个向量作为自变量时结果返回一个数值向量，类为table; 多个向量输入时结果返回数组，数组维数和向量个数相同。可以用as.data.frame把table的结果转换为数据框，转换的数据框每行仅有一个单元格的频数。

有了table的结果，可以用addmargins函数增加行和、列和，还可以对每行、每列求其它汇总值。

为了求单元格百分比，可以用sweep函数，或直接对table的结果用 prop.table转换，prop.table可以用margin=指定行百分比(margin=1)或列百分比(margin=2)。

如果有多个交叉分类变量，table的结果是多维数组，这时可以用ftable函数把多维数组转换为二维数组。

xtabs函数与table函数类似，主要针对数据框，指定交叉分类变量使用公式方式指定，公式的右边指定分类变量，坐标空缺或指定一个频数变量。

#### §3.4.2 汇总函数的选择

这里的汇总是指把数据分组后对每组计算总和、平均等汇总。需要考虑：数据如何分组，要汇总的变量，希望最终汇总结果如何排列。

- 当分组是把每个组保存为列表的一个元素时，一般用sapply, lapplay。一般最好使用sapply, sapply会把返回结果以向量、数组格式保存，而lapply总是返回列表。split函数可以把一个数据框拆分为若干个小数据框组成的列表，然后就可以利用sapply或lapply。

  比如，eigen返回一个列表，为了得到列表元素的类，可以用sapply(eigen(A), class)。

- 如果分组是用矩阵的行或矩阵的列表时的，即需要对矩阵的每行或每列进行汇总，可以用apply函数。apply函数一般返回向量或数组，但如果结果长度不规则可以返回列表。

apply可以用于scale函数中计算每列的中心和分散程度，如scale(df, center=apply(df, 2, median), scale=apply(df, 2, mad))。

对矩阵求行和、列和等可以用函数rowSums、colSums、rowMeans等。

如果对每行或每列进行不同处理，操作的不同只是不同行（列）采用某个向量的不同元素作为参数，这时可以用sweep函数。sweep与apply的差别在于apply中的操作是需要输入单个自变量的，而sweep中的操作时针对两个自变量，第一个自变量是矩阵的行（列），第二个自变量是sweep函数的第三个自变量。如

```
max1 <- apply(state.x77, 2, max) # 每列的最大值
sweept <- sweep(state.x77, 2, max1, "/")
  # 每列除以该列最大值
```

mapply函数是把操作只接受一个自变量作为输入的apply函数进行了推广，使得操作可以接受多个自变量，如如mapply(func, df1, vec1)将操作func应用于数据框df1的每一列和向量vec1的对应元素。

- 如果有一个或几个分组变量表示分组，有多个函数可以进行汇总。如果交叉分组后每个组只有一个向量，结果每个组只有一个标量，最好用aggregate函数，此函数总是返回数据框。

  如果每个组织有一个向量，但是汇总结果是向量，可以使用tapply。tapply可以返回向量或数组，结果中的单个组的结果可能很容易访问，但是整个结果的显示则难以解释。所以这时可以考虑使用reshape包，可以产生多种形式的输出。

  如果需要汇总多个变量，比如计算变量之间的相关系数，可以把分组的下标传递给tapply函数，或使用by函数，by函数的结果在显示的时候比较易读，但访问单个结果麻烦一些。

如果难以找到适当方法，用split函数把数据框分成子集列表再用sapply或lapply处理，实在不行还可以写循环来处理，循环的交叉分组可以用unique和intersect函数辅助。

### §3.4.3 关于向量或列表的逐项操作

用lapply和sapply可以对向量或列表进行逐个元素的操作，用户指定一个函数作为操作。例如

```
> mylist <- list(x=1:3, name=c('a', 'b', 'c'))

> length(mylist)
```

```
[1] 2
> sapply(mylist, length)
   x name
   3    3
```

其中直接用length函数求得的是列表mylist的元素个数，而使用sapply则可求得列表中每个元素的长度。sapply尽可能返回向量、数组，而lapply总是返回一个列表，输入的每个元素对应于结果列表的一个元素。

又例如，为了求数据框df中所有的数值型列，可以用

```
  df[, sapply(df,class)=='numeric']
```

sapply和lapply相当于对每个列表元素循环，如果仅仅是为了对某个函数（一般是随机模拟函数）循环调用若干次，可以用replicate函数，如

```
replicate(10000,
  t.test(rnorm(10, rnorm(10)))$\backslash$statistic)
```

可以重复进行t检验10000次并返回10000个t统计量值组成的向量。

split函数可以把一个向量或数据框按照某个等长的因子划分为若干个组，结果为列表，然后可以利用sapply或lapply计算组汇总。tapply可以直接对一个向量按照因子分组进行汇总。比如，iris数据框中有3个品种的鸢尾花的各50个测量结果，为了计算各品种的平均测量值，可用如

```
iris.sp <- split(iris[,-5], iris[,5])
sapply(iris.sp, colMeans)
```

结果为

```
             setosa versicolor virginica
Sepal.Length  5.006      5.936     6.588
Sepal.Width   3.428      2.770     2.974
Petal.Length  1.462      4.260     5.552
Petal.Width   0.246      1.326     2.026
```

可以看出sapply的结果已经转换为矩阵。

### §3.4.4 对矩阵的每行或每列的操作

使用apply函数。apply(mat,1,fun)表示对矩阵mat的每行使用函数fun处理，apply(mat,2,fun)表示对每列处理。

例如，scale函数可以把矩阵的每列标准化，缺省操作是减去列的平均值再除以列的标准差。如果需要规定其它的减数和除数，可用如

```
> sstate <- scale(state.x77,
+   center=apply(state.x77, 2, median),
+   scale=apply(state.x77, 2, mad))
```

其中mad是正态情况下标准差的一个稳健估计。

如果函数的结果返回多个值，apply把结果表示为一个矩阵。例如，为了求state.x77中各列的有效观测个数、均值、标准差，可用

```
> summfn <- function(x){
+   c(n=sum(!is.na(x)), mean=mean(x), sd=sd(x))}
> t(apply(state.x77, 2, summfn))
             n       mean           sd
Population 50  4246.4200 4.464491e+03
Income     50  4435.8000 6.144699e+02
Illiteracy 50     1.1700 6.095331e-01
Life Exp   50    70.8786 1.342394e+00
Murder     50     7.3780 3.691540e+00
HS Grad    50    53.1080 8.076998e+00
Frost      50   104.4600 5.198085e+01
Area       50 70735.8800 8.532730e+04
```

注意自定义的汇总函数中返回值的元素名最后变成了apply结果的变量名。

对矩阵求行和、列和等可以用函数rowSums、colSums、rowMeans等。

sweep函数推广了apply，其操作需要一个额外的输入，这样对每行或每列操作时可以使用不同的参数。如

```
> maxes <- apply(state.x77, 2, max)
> swept <- sweep(state.x77, 2, maxes, "/")
> cbind(state.x77[1:3, 1:2], swept[1:3, 1:2])
        Population Income Population    Income
Alabama       3615   3624 0.17053496 0.5738717
Alaska         365   6315 0.01721861 1.0000000
Arizona       2212   4530 0.10434947 0.7173397
```

上述程序把state.x77的每一列除以自己的最大值。

sweep进行的操作函数有时不能识别传递给函数的矩阵数据，用mapply函数可以指定有多个自变量的操作，输入多个自变量的值。比如，希望对state.x77的每列计算那些超过中位数的元素的平均值，可用

```
> meds <- apply(state.x77, 2, median)
> meanmed <- function(x, med) mean(x[x>med])
> mapply(meanmed, as.data.frame(state.x77), meds)
Population     Income Illiteracy   Life Exp
  7136.160   4917.920      1.660     71.950
   Murder     HS Grad      Frost       Area
    10.544     59.524    146.840 112213.400
```

### §3.4.5 分组进行操作

为了对数据框或矩阵的一列或多列计算单一统计量，可以使用aggregate函数。用法为aggregate(df, group, fun)，其中df为数据框或矩阵，by是分组变量组成的列表，fun是要进行的操作。如

```
> aggregate(iris[,-5], list(species=iris[,5]), mean)
     species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
```

aggregate缺省情况下会简化结果结构。

aggregate所用的分组变量列表可以是数据框中的一个或多个分类变量，这时可以用单个方括号的下标形式访问数据框，产生的是列表，比如下面的程序按Time 和Diet交叉分组对weight进行平均：

```
> cweights <- aggregate(ChickWeight$weight,
+   ChickWeight[c('Time', 'Diet')], mean)
> head(cweights, 4)
  Time Diet        x
1    0    1 41.40000
2    2    1 47.25000
3    4    1 56.47368
4    6    1 66.78947
```

如果要统计的是一个变量，即使有交叉分组，aggregate给出的统计结果也只占一列。如果使用tapply进行这样的统计，结果将是一个数组，数组的

维数与分组变量个数相同。例如，PlantGrowth数据框包含三个不同处理组的植物重量。为了计算每个组的最大重量，可用

```
> maxweight <- tapply(PlantGrowth$weight,
    PlantGrowth$group, max)
> maxweight
ctrl trt1 trt2
6.11 6.03 6.31
```

没有交叉分组，所以结果是一个有名向量。可以用as.table和as.data.frame转换为纵向的数据框：

```
> as.data.frame(as.table(maxweight))
  Var1 Freq
1 ctrl 6.11
2 trt1 6.03
3 trt2 6.31
```

如果不使用as.table而只使用as.data.frame，结果中的ctr1等名字只能作为行名而不能单独作为一列数据。

如果tapply的操作返回多个结果，则tapply会返回列表，比如：

```
> ranges <- tapply(PlantGrowth$weight,
+   PlantGrowth$group, range)
> ranges
$ctrl
[1] 4.17 6.11

$trt1
[1] 3.59 6.03

$trt2
[1] 4.92 6.31
```

用unlist函数可以把列表的元素的值变到一个向量中。如

```
> unlist(ranges)
ctrl1 ctrl2 trt11 trt12 trt21 trt22
 4.17  6.11  3.59  6.03  4.92  6.31
```

如果列表的每个元素是等长的向量，我们可以把unlist之后的结果转换为一个矩阵，再变成数据框，例如

```
> data.frame(group=names(ranges),
+   matrix(unlist(ranges),
+     ncol=2, byrow=TRUE))
  group   X1   X2
1  ctrl 4.17 6.11
2  trt1 3.59 6.03
3  trt2 4.92 6.31
```

如果tapply交叉分组的操作时返回多个值的，则tapply的返回结果是一个矩阵形式，但矩阵的每个元素是列表，如

```
> ranges1 <- tapply(CO2$uptake,
+   CO2[c('Type', 'Treatment')], range)
> ranges1
             Treatment
Type          nonchilled chilled
  Quebec      Numeric,2  Numeric,2
  Mississippi Numeric,2  Numeric,2
> ranges1[['Quebec', 'chilled']]
[1]  9.3 42.4
```

ranges1的dimnames属性包括行名和列名，可以用expand.grid函数将其两两组合产生，然后用unlist(ranges1)得到具体的数值，用matrix函数转换时要加上byrow=TRUE选项才符合expand.grid的次序。如

```
> data.frame(expand.grid(dimnames(ranges1)),
+   matrix(unlist(ranges1), byrow=TRUE, ncol=2))
         Type  Treatment   X1   X2
1      Quebec nonchilled 13.6 45.5
2 Mississippi nonchilled 10.6 35.5
3      Quebec    chilled  9.3 42.4
4 Mississippi    chilled  7.7 22.2
```

如果tapply函数没有指定操作，其返回值是一个与输入等长的下标向量，代表在操作为标量结果时每行的结果在结果向量中的下标。例如，我们先计算了交叉分组的uptake的中位数：

```
meds <- tapply(CO2$uptake, CO2[c('Type',
  'Treatment')], median)
```

然后，我们用没有操作函数的tapply 得到每行对应的中位数在meds中的下标位置：

```
inds <- tapply(CO2$uptake,
  CO2[c('Type', 'Treatment')])
```

然后就可以对每种组合减去相应的中位数：

```
adj.upate <- CO2$uptake - meds[inds]
```

函数ave可以计算分组的统计量, 从而也可以进行上述的调整。ave分组计算各组的均值并返回与输入等长的均值向量，用参数FUN可以指定不是计算均值而是计算其它统计量。如

```
adj.uptake <- CO2$uptake - ave(CO2$uptake,
  CO2$Type, CO2$Treatment, FUN=median)
```

上面讲了用aggregate处理一列或多列分组因变量按一个或多个分组变量分组对每个因变量计算单一统计量的问题，用tapply处理一列因变量按一个或多个分组变量对每个因变量计算单一或多个统计量的问题。那么，如果操作是需要同时使用多个列计算怎么办?

举例说明。要求iris数据框中四个数值型变量的相关阵的最大特征值，并分不同品种计算，可以用split和sapply配合，如

```
> maxeig <- function(df) {
+   eigen(cor(df))$val[1] }
> frames <- split(iris[-5], iris[5])
> sapply(frames, maxeig)
    setosa versicolor  virginica
  2.058540   2.926341   2.454737
```

其中split的调用中iris的下标用了单方括号，对一个列表或数据框用单方括号下标得到的结果还是列表，这是split的分组用自变量（第二自变量）需要的数据格式。

这个例子的第二种解法是用tapply，被分组的是行下标向量，在操作函数中使用下标子集处理。如

```
> tapply(seq(nrow(iris)), iris['Species'],
+   function(ind, data)
+   eigen(cor(data[ind,-5]))$val[1],
+   data=iris)
Species
    setosa versicolor  virginica
  2.058540   2.926341   2.454737
```

这个例子的第三种解法是使用by函数。by函数与tapply类似，但尽可能返回向量或矩阵。如

```
> max.e <- by(iris, iris$Species, function(df)
+    eigen(cor(df[-5]))$val[1])
> max.e
iris$Species: setosa
[1] 2.058540
------------------------------------------------------------------------------
iris$Species: versicolor
[1] 2.926341
------------------------------------------------------------------------------
iris$Species: virginica
[1] 2.454737
> as.data.frame(as.table(max.e))
  iris.Species     Freq
1       setosa 2.058540
2   versicolor 2.926341
3    virginica 2.454737
```

用了as.table和as.data.frame转换结果为数据框。

再来看CO2数据框交叉分组计算每组的最小、最大值的问题，tapply的结果是一个矩阵形式的列表，上面用unlist、matrix、expand.grid 函数把结果转换为数据框，我们用by来解此问题。

```
> bb <- by(CO2$uptake, CO2[c('Type', 'Treatment')], range)
> bb
Type: Quebec
Treatment: nonchilled
[1] 13.6 45.5
```

```
------------------------------------------------------------
Type: Mississippi
Treatment: nonchilled
[1] 10.6 35.5
------------------------------------------------------------
Type: Quebec
Treatment: chilled
[1]  9.3 42.4
------------------------------------------------------------
Type: Mississippi
Treatment: chilled
[1]  7.7 22.2
```

每个组合有两个结果。如果能把各行用rbind合并则成为一个合适的数据框。使用do.call可以把一个列表的各个元素作为不同的自变量传递给一个函数：

```
> do.call(rbind, bb)
     [,1] [,2]
[1,] 13.6 45.5
[2,] 10.6 35.5
[3,]  9.3 42.4
[4,]  7.7 22.2
```

结果中没有适当的行名和列名。用expand.grid可以加上交叉分组标识：

```
> cbind(expand.grid(dimnames(bb)),
+   do.call(rbind, bb))
         Type  Treatment    1    2
1      Quebec nonchilled 13.6 45.5
2 Mississippi nonchilled 10.6 35.5
3      Quebec    chilled  9.3 42.4
4 Mississippi    chilled  7.7 22.2
```

注意其中的cbind函数是在左右合并两个数据框，不需要区分数值型和字符型。

### §3.4.6 用reshape包汇总

**融化和转换** reshape包提供了常用的数据形状变换和汇总功能，先作“融化”（melt），把变量分成标识(id)变量和测量变量，不同的测量变量也堆叠到

一列value放置，新引入变量识别列分类变量variable区分某行到底是哪一个测量变量的值，融化后只有一列是测量值, 每行为一个特定组合的某一个测量变量的测量结果。如果特定组合对应多个观测，则测量结果会被汇总起来。

对融化了的数据可以“转换”（cast）为需要的形状，也可分类汇总。

所有的测量变量要求是同类型的，比如数值型、因子型、日期型，这是因为所有测量变量的测量值融化后会放在一列中。因为融化的目的是为了转换，能进行转换的测量变量一般是同类型的。

**基本的融化** melt函数的基本用法是melt(df, id, measured), 其中df一般是数据框（也可以对数组、矩阵列表等操作）。如

```
> require(reshape)
> smiths
     subject time age weight height
1 John Smith    1  33     90   1.87
2 Mary Smith    1  NA     NA   1.54
> melt(smiths)
Using subject as id variables
     subject variable value
1 John Smith     time  1.00
2 Mary Smith     time  1.00
3 John Smith      age 33.00
4 Mary Smith      age    NA
5 John Smith   weight 90.00
6 Mary Smith   weight    NA
7 John Smith   height  1.87
8 Mary Smith   height  1.54
```

只有subject作为id变量。time, age, weight 都作为测量并堆叠到value列放置，用variable列区分是哪一个变量的测量值。

可以指定id和measure变量名，如

```
> smithsm <- melt(smiths, id=c('subject', 'time'),
+    measured=c('age', 'weight', 'height'))
> smithsm
     subject time variable value
1 John Smith    1      age 33.00
2 Mary Smith    1      age    NA
```

```
3 John Smith    1   weight 90.00
4 Mary Smith    1   weight    NA
5 John Smith    1   height  1.87
6 Mary Smith    1   height  1.54
>
```

如果只指定id和measured之一，则其它变量自动归入另一类。如

```
> melt(smiths, id=c('subject', 'time'))
> melt(smiths, id=1:2)
> melt(smiths, measured=c('age','weight','height'))
```

**处理和时间编码于变量名中的情况** 假设有4个受试者，两个处理A, B，A处理有两个观测，B处理有一个观测，测量值保存在A1, A2, B1这3列中。融化时为了提取处理和观测信息可以使用reshape包的colsplit函数产生处理列和时间列，如

```
> trial <- data.frame(id=factor(1:4),
+   A1=c(1,2,1,2), A2=c(2,1,2,1),
+   B1=c(3,3,3,3))
> (trialm <- melt(trial))
Using id as id variables
   id variable value
1   1       A1     1
2   2       A1     2
3   3       A1     1
4   4       A1     2
5   1       A2     2
6   2       A2     1
7   3       A2     2
8   4       A2     1
9   1       B1     3
10  2       B1     3
11  3       B1     3
12  4       B1     3
> (trialm <- cbind(trialm,
+   colsplit(trialm$variable,
+     names=c('treatment', 'time'))))
   id variable value treatment time
```

```
1   1       A1      1       A    1
2   2       A1      2       A    1
3   3       A1      1       A    1
4   4       A1      2       A    1
5   1       A2      2       A    2
6   2       A2      1       A    2
7   3       A2      2       A    2
8   4       A2      1       A    2
9   1       B1      3       B    1
10  2       B1      3       B    1
11  3       B1      3       B    1
12  4       B1      3       B    1
>
```

更一般的变量名编码可以用正则表达式拆分。

如果数据已经符合融化的要求，只要保证测量值列命名为value。

**缺失数据问题** 有些分组带有结构性缺失，比如性别和是否怀孕交叉分组则男性怀孕单元缺失。在融化状态下，只要某种组合的行不出现在数据框中则隐含表示此组合缺失。在melt函数调用时加上na.rm=TRUE可以删除缺失的组合。在融化时如果不使用na.rm=TRUE 则在后续的汇总（求和、平均等）时需要考虑缺失值处理。

**融化数据的变形** 用cast函数对融化的数据变形。需要输入融化的数据框和一个公式。如果不指定公式，cast的结果是把堆叠到一列的测量变量再拆分为多个变量列，而分组变量不变。cast可以指定一个fun.aggregate函数用来汇总分组后的多个观测，指定margins来说明计算那些边缘统计。

cast的公式的左面和右面都是用加号连接的变量名，左面是放置在列的变量，右面是放置在行的变量，只写一个点表示没有变量。比如 . ~ x表示只有一行x。用...表示公式中没有出现的所有其余变量，包含此项则不会发生汇总。如果公式进行了汇总，每种组合的汇总结果都有多个值，可以用result_variable指定汇总结果。

为了把smithsm数据用时间和受试者在行分类，把变量在列展现，可用如

```
> cast(smithsm, time + subject ~ variable)
  time    subject age weight height
1    1 John Smith  33     90   1.87
2    1 Mary Smith  NA     NA   1.54
```

注意value变量是不需要的，它和variable变量是匹配的。

下例用...表示公式中没有出现的所有变量：

```
> cast(smithsm, ... ~ variable)
     subject time age weight height
1 John Smith    1  33     90   1.87
2 Mary Smith    1  NA     NA   1.54

> cast(smithsm, ... ~ subject)
  time variable John Smith Mary Smith
1    1      age      33.00         NA
2    1   weight      90.00         NA
3    1   height       1.87       1.54

> cast(smithsm, ... ~ time)
     subject variable     1
1 John Smith      age 33.00
2 John Smith   weight 90.00
3 John Smith   height  1.87
4 Mary Smith      age    NA
5 Mary Smith   weight    NA
6 Mary Smith   height  1.54
```

cast的结果是数据框，而数据框的显示有一些限制，比如在上面最后一个结果中时间变量的值1只能显示为一个数值1，而不能显示出这是time的值等于1。

**融化数据的汇总** 使用cast时如果分组后每组有不止一个数值则会进行汇总。缺省汇总是特定组合中的元素个数。这种功能类似于SAS系统PROC TABULATE的功能。以reshape包中的french_fries数据框为例。要比较3种不同的煎炸油（treatment），有10个评判者（subject），每次评判重复两个样品，一共10周时间（time）。对5个方面进行评分(potato, buttery, grassy, rancid, painty)。

```
names(french_fries)
ffm <- melt(french_fries, id=1:4, na.rm=TRUE)
```

下面程序按照处理进行汇总，求每种处理的测量值个数：

```
> cast(ffm, treatment ~ ., length)
  treatment (all)
1         1  1159
2         2  1157
3         3  1155
```

下面的程序演示如何安排交叉分组。可以分别占据行和列，和可以都在行上。

```
> cast(ffm, rep ~ treatment, length)
  rep   1   2   3
1   1 579 578 575
2   2 580 579 580

> cast(ffm, treatment ~ rep, length)

> cast(ffm, treatment + rep ~ ., length)
  treatment rep (all)
1         1   1   579
2         1   2   580
3         2   1   578
4         2   2   579
5         3   1   575
6         3   2   580

> cast(ffm, rep + treatment ~ ., length)
```

公式中的变量先后次序使得结果展现不同，为了使结果便于比较，应该使要比较的单元格相邻放置，这只要把要比较的因素放在靠后的次序。另外，显示时表格宽是受限制的，所以分类多的变量应该放在表的行维。

**产生多维数组** 在cast中可以使用多个~产生多维数组。如

```
> dim(cast(ffm, time ~ variable ~ treatment, mean))
[1] 10  5  3
```

cast结果为3维数组，相当于每种处理有一张表，为各时间各测量属性的平均值。又如

```
> dim(cast(ffm, time ~ variable ~ treatment + rep,
    mean))
[1] 10  5  6

> dim(cast(ffm, time ~ variable
    ~ treatment ~ rep, mean))
[1] 10  5  3   2
```

这些多维数组形式适用于用sweep进行标准化，或用iapply修改。

因为~结果要交叉每种组合，所以比在公式中用加号交叉分组产生更多的缺失值。

**产生列表** 在cast的公式中用|可以指定条件变量，对条件变量的每个组合分别给出结果数组。如

```
> cast(ffm, treatment ~ rep | variable, mean)
```

结果为一个分不同测量属性(variable)的列表，每个列表的元素为该测量属性分处理(treatment)和重复号(rep)的平均值。与产生3维数组的结果类似。

下面是一个复杂的列表结果例子。

```
> sapply(cast(ffm, treatment ~ rep
+     | time + variable, mean), length)
 1  2  3  4  5  6  7  8  9 10
 5  5  5  5  5  5  5  5  5  5
```

结果列表是10个元素的，每个元素对应一个时间值，每个元素又是有5个元素的子列表，自列表的每个元素对应一个测量属性，自列表元素值为$3 \times 3$的矩阵。所以以上sapply的结果返回的是各子列表长度，都等于5。

**边缘值** 边缘值指的是如同列联表行和、列和这样的统计量。在cast函数调用时加上margins=TRUE选项可以计算所有边缘值，也可以指定那些分类要求不分类的总计，如margins=c('subject', 'time')。指定求总计的分类时名字'grand_col'和'grand_row'用来指总的列和行。边缘值标记为要总计的分类变量显示为"(all)"。

例：

```
> cast(ffm, treatment ~ ., sum,
+   margins=TRUE)
  treatment   (all)
```

```
1         1  3702.4
2         2  3640.4
3         3  3640.2
4     (all) 10983.0
```

计算各处理的各测量值的总和，这里指定了要计算边缘值所以还有一个不分处理的综合。结果数据框的第4行为处理的边缘值。这个例子边缘的规定写成margins='grand_row'效果也是一样的。

例：

```
> cast(ffm, treatment ~ rep, sum, margins=TRUE)
  treatment      1      2   (all)
1         1 1857.3 1845.1  3702.4
2         2 1836.5 1803.9  3640.4
3         3 1739.1 1901.1  3640.2
4     (all) 5432.9 5550.1 10983.0
```

结果的最后一行是不分处理的列和，最后一列是不分重复号的行和。

例：

```
> cast(ffm, treatment + rep ~ ., sum,
+     margins=TRUE)
   treatment   rep   (all)
1          1     1  1857.3
2          1     2  1845.1
3          1 (all)  3702.4
4          2     1  1836.5
5          2     2  1803.9
6          2 (all)  3640.4
7          3     1  1739.1
8          3     2  1901.1
9          3 (all)  3640.2
10     (all) (all) 10983.0
```

结果矩阵的第3、6、9行为3个处理各自关于所有重复号(rep)的边缘值，第10行为不分处理也不分重复号的边缘值，没有每个重复号各自关于处理的边缘值。

例：

```
> cast(ffm, treatment + rep ~ time,
     sum, margins=TRUE)
```

```
> cast(ffm, treatment + rep ~ time,
     sum, margins='treatment')

> cast(ffm, treatment + rep ~ time,
     sum, margins='rep')
```

其中margins='treatment'表示每种处理进行不分重复号(rep)的汇总，margins='rep'表示每个重复号进行不分处理的汇总。

**返回多个值的汇总** cast可以处理返回多个值的汇总。比如：

```
> cast(ffm, treatment ~ ., summary)
  treatment Min. X1st.Qu. Median  Mean X3rd.Qu. Max.
1         1    0        0    1.6 3.194      5.4 14.9
2         2    0        0    1.4 3.146      5.4 14.9
3         3    0        0    1.5 3.152      5.7 14.5

> cast(ffm, treatment ~ rep, range)
  treatment 1_X1 1_X2 2_X1 2_X2
1         1    0 14.9    0 14.3
2         2    0 14.9    0 13.7
3         3    0 14.5    0 14.0
```

上面求range的例子1_X1, 1_X2对应于重复号1的最小值和最大值。

**reshape包的其它函数** rescaler对数据框的列进行标准化，提供不同的标准化方法。

merge.all按同名列合并多个数据框。

rbind.fill上下合并两个数据框，第二个数据框中缺失的列填充缺失值。

iapply是apply的一个改进，可以尽可能保持输入数组的原来形状。

# §4 统计功能

## §4.1 回归

**公式** S中公式表示成response ~ expression的形式，response是因变量，有些方法中没有此项；expression是表达式，在不同方法中有不同的解释。能接受公式为输入的函数常常还可以有weight, data, subset, na.action参数。weight指定权重；data指定一个数据框，公式中的变量优先在此数据框中查找；subset用data中的变量构成一个逻辑下标或数值下标，对数据取行子集。

公式中可以用“.”表示原公式中的内容，用“+”表示添加的项，用“-”表示扣除的项。

**回归的更新与检验** 用update更新修改一个回归模型。如

```
lm1 <- lm(y ~ x1 + x2)
lm2 <- update(lm1, . ~ . - x2)
```

用anova比较两个有包含关系的模型，如

```
anova(lm1, lm2)
```

**回归系数估计的方差阵**

```
rs <- summary(lm.res)
CovMat <- rs$sigma^2 * rs$cov
```

**加权回归** 只要在lm调用时加weight选项。

**局部多项式曲线拟合** 用loess(y ~ x)。

为了绘制拟合后的光滑曲线，可以用spline(x, y)得到插值后的结果再绘图。

**Logit和probit回归** 用glm。

**多项logit回归** 用nnet包中的multinom函数。

**有序Logit/Probit** 用MASS包中的polr函数。

**带删失变量的回归** 用survreg并指定dist="gaussian"即可。

**分位数回归** 用quantreg包中的`rq`函数。

**稳健回归** MASS包中的`lqs`函数以最小化某个残差排序后的分位数为目标。`rlm`函数使用M估计进行回归。

**非线性最小二乘** 用`nls`函数。

**树回归** 用`tree`包的`tree`函数。

### §4.2 神经网络

前馈神经网络(feed forward neural networks)是一般性的非线性回归方法，它是一种黑箱子类型的方法，比较适于预报。

单隐层的神经网络的模型为

$$y_k = \phi_o\left(\alpha_k + \sum_h w_{hk}\phi_h\left(\alpha_h + \sum_i w_{ih}x_i\right)\right) \tag{1}$$

其中$\{x_k\}$为输入，$\{y_k\}$为输出，$\phi_h(\cdot)$为隐层结点的输出函数，一般取为逻辑斯蒂函数

$$l(z) = \frac{\exp(z)}{1+\exp(z)}$$

$\phi_o(\cdot)$为输出接点的输出函数，可能取为线性函数、逻辑斯蒂函数或截断函数(截断函数即$I(z>0)$)。$w_{ih}$是从输入到隐层的连接权重，$w_{hk}$是从隐层到输出的连接权重，$\alpha_h$是隐层的固定偏移，$\alpha_k$是输出层的固定偏移。

模型(1)可以加入在输入和输出层直接连接，跳过了隐层的项，这时模型为

$$y_k = \phi_o\left(\alpha_k + \sum_i w_{ik}x_i + \sum_h w_{hk}\phi_h\left(\alpha_h + \sum_i w_{ih}x_i\right)\right) \tag{2}$$

其中$w_{ik}$表示从输入到输出的连接权重。

使用前馈神经网络，输入和输出最好都归一化到取值$[0,1]$区间内。文献中已证明线性输出单元的神经网络可以一致逼近任意闭集上的连续函数。训练权重的方法是最小化某些准则，如最小二乘

$$E = \sum_p \|t^p - y^p\|^2$$

其中$t^p$是逼近目标，$y^p$是第$p$个输入样例的实际输出。还有如$y\in[0,1]$时的最大似然(实际是最小化负对数条件似然)，这与最小化Kullback-Leibler距离(信息熵)等价：

$$E = \sum_p \sum_k \left[t_k^p \log\frac{t_k^p}{y_k^p} + (1-t_k^p)\log\frac{1-t_k^p}{1-y_k^p}\right]$$

其中下标$k$代表输出的不同分量，上标$p$代表用于训练的不同样例。

为了使得拟合的行数比较光滑，避免过度拟合，可以使用“权重衰减”，对各权重的平方和施加惩罚，对最小二乘准则常用$\lambda \approx 10^{-4} - 10^{-2}$, 对KL准则常用$\lambda \approx 0.01 - 0.1$。

`nnet`包的`nnet`函数用法为

```
nnet(formula, data, weights, size, Wts, linout=F,
  entropy=F, softmax=F, skip=F, rang=0.7, decay=0,
  maxit=100, trace=T)
```

各输入解释如下：

- size: 隐层大小。
- Wts: 可选的初始权重向量。
- linout: 输出单元是否为线性单元。
- entropy: 是使用信息熵准则，还是最小二乘准则。
- softmax: 是否使用对数概率模型。
- skip: 是否有直连。
- rang: 如果不指定初始权重，使用随机权重作为初值，初值在`(-rang, rnag)`内随机均匀取值。
- decay: 权重衰减系数$\lambda$。
- Hess: 是否返回最小值点处的海色阵。
- trace: 最小化过程的输出。

### §4.3 时间序列

`lag`可以计算滞后序列，但只能作用于时间序列对象，对一般向量会得出错误结果。`diff`计算差分。`ts.union`可以形成多元时间序列。`filter`函数可以计算递推的或卷积的滤波。`arima`可以拟合ARIMA模型。`acf`和`pacf`函数可以作自相关和偏自相关图。

fracdiff库的`fracdiff`可以拟合分数ARIMA模型。

tseries包中的`garch`可以拟合GARCH和ARCH模型。

fSeries包中的`garchFit`可以拟合较复杂的包含GARCH的模型。

car包中的`durbin.watson`函数可以对lm对象进行Durbin-Watson自相关检验。ts包中的`Box.test`可以进行Box-Pierce或Ljung-Box白噪声检验。tseries包中的`adf.test`可以进行Dickey-Fuller单位根检验。

# §5 图形和图形界面

## §5.1 图形经验

### §5.1.1 条形图

用barplot()制作条形图。最简单的条形图输入一个有名数值向量，元素值为条形高度，元素名为条形标签，如

```
barplot(c('Female'=9, 'Male'=15))
```

各条形的标签也可以用参数names.arg提供。

table的结果很合适作为barplot的输入，如

```
sex <- factor(c('F', 'F', 'M', 'M', 'M', 'M'))
st <- table(sex)
barplot(st, main='Numbers of female and male')
```

可以用col指定条形填充颜色，如

```
barplot(c('Female'=9, 'Male'=15), col='red')
```

也可以填充为斜线，用angle指定斜线角度，如

```
barplot(c('Female'=9, 'Male'=15), angle=30)
```

### §5.1.2 其它图形经验

- 可以同时打开多个绘图设备，用dev.list()列出当前打开的图形设备，用dev.set选择用哪一个，用dev.off()关闭当前设备。
- curve函数: 用于函数或表达式作图，与add=T选项配合可以在已有的图上添加曲线。
- MASS包的truehist作密度估计直方图，用kde2d(x,y)作二元核密度估计，用con2tr(kde2d(x,y))把二元核密度估计结果转换为lattice包三维图需要的格式。用splom(d)对一个数据框或矩阵作散点图矩阵。
- 添加标注: 用text, lines, points, arrows，legend等。用locator定位。
- 若f是因子，y是数值向量，plot(f, y)或plot(y ~ f)按因子水平分组作y的盒形图。
- 输出到图形文件: 直接用图形设备postscript, pdf, png, jpg, xfig等。

- 图中用数学符号: 用如main=substitute(y==Psi*z-sum(beta[0]gamma))，会变成数学符号。下标用方括号表示。等号用两个等号表示。还可以给substitute一个额外的list参数表示要代入实际值的变量。混合文本如main=substitute(paste("t-stat of ", beta[0]))。
- 等值线图: contour函数。可以用contourLines添加。lattice包中有类似的levelplot和contourplot函数。
- lattice包中用wireframe(z  x + y)作三维曲面图, 用contourplot(z  x + y)作等值线图。
- PDF中的汉字使用: 在打开pdf设备时，用如

```
pdf("文件名", family="GB1")
```

即可。

### §5.2 简单交互式图形界面

R 通过对Tcl/Tk的支持（tcltk包）实现了图形界面(GUI)功能，但是需要的编程略微复杂。rpanel是建立在tcltk包基础上的一个包，它用很简单的几个语句就可以建立一个调控运行参数的界面，主要用来实时更新图形，修改界面上的控制参数则图形立刻刷新。真的很容易。

**用RPanel建立图形界面** 用rp.control()函数建立一个窗口，用title指定标题。返回值保存一个代表此窗口的列表变量，列表中为图形数据。rp.control多余的参数被保存在列表中。如：

```
density.panel <- rp.control(title="Density estimation",
     y=A1203, sp=r/8)
```

其中y, sp是额外的变量，被存放在结果列表density.panel中。

在窗口中添加控件时以窗口变量为第一自变量，如：

```
rp.slider(panel=density.panel, var=sp, from=r/50, to=r/4,
          action=density.draw, title="Bandwidth")
```

在density.panel窗口中增加一个滑块控件，其中var参数指定窗口列表中已有的变量sp，变量的值将由此控件修改。action是修改值时更新用的响应函数。响应函数density.draw以一个窗口变量为输入，并返回可能修改后的窗口变量。

rp.radiogroup添加单选钮，如:

```
rp.radiogroup(panel=density.panel, var=sp.method,
      values=c("normal", "plug-in", "manual"),
      title="Bandwidth Selection", action=density.draw)
```

其中var=sp.method在density.panel列表中定义一个sp.method变量并由此单选钮组选择其取值。

rp.checkbox添加复选框，如：

```
rp.checkbox(panel=density.panel, var=model,
      title="Normal band", action=density.draw)
```

指定density.panel列表变量model并由此控件控制开关。

rp.doublebutton添加加减按钮，如:

```
rp.doublebutton(panel=density.panel, var=sp, step=1.02,
     log=TRUE, range=c(diff(range(A1203))/50, NA),
     title="Bandwidth")
```

结果是并排的减号和加号，可以控制一个窗口列表变量的增减。var=sp指定要控制的窗口列表变量，step是变化量，log=TRUE指定是用等比方式变化，range给出变换的界限。

### §5.3 R中tcltk包的使用

例子见http://bioinf.wehi.edu.au/~wettenhall/RTclTkExamples/ 。Tcl/Tk的原始参考资料见http://www.tkdocs.com/widgets/index.html 和http://www.tcl.tk/-man/tcl8.5/contents.htm。

要先有一个主窗口，用tktoplevel生成，如:

```
root <- tktoplevel()
```

要指定窗口标题，用如:

```
tktitle(root) <- "tcltk测试"
```

生成按钮用command指定操作，如:

```
exit.but <- tkbutton(root,text="退出",
    command=function() tkdestroy(root))
tkgrid(exit.but)  ## 用来摆放按钮
tkfocus(root)
```

按钮的对应操作是关闭窗口。

提示信息对话框:

```
res <- tkmessageBox(title="计算程序",
    message="试验时间数据输入错误!",
    icon="error",type="ok")
```

可以用icon="info", icon="warning"。如果type="yesnocancel"结果会有三个按钮, 用default="yes"指定缺省按钮。用as.character(res)查看结果。

用tclVar建立tcl变量，用tclvalue访问其值，如:

```
done <- tclVar(0) ## 建立tcl变量done，值为0.
OK.but <- tkbutton(root, text=" OK ",
    command=function() tclvalue(done)<-1)
```

按钮的作用是把tcl变量done的值赋值为1. 可以用tclvalue(done)访问这个值，如果是整数应该用as.integer(tclvalue(done))转换。

用tkbind给某个窗口事件规定响应，如:

```
tkbind(root, "<Destroy>", function() tclvalue(done) <- 2)
```

用tkwait.variable等待tcl变量的值改变，如:

```
tkwait.variable(done)
```

用tklabel添加一个标签控件，如:

```
tkgrid(tklabel(root,text="This is a text label"))
```

如果需要修改标签内容，可以用tcl变量，如:

```
labelText <- tclVar("This is a text label")
label1 <- tklabel(root,text=tclvalue(labelText))
tkconfigure(label1,textvariable=labelText)
   ## 把控件与tcl变量关联。
tkgrid(label1)
ChangeText <- function()
  tclvalue(labelText) <- "This text label has changed!"
ChangeText.but <- tkbutton(tt,text="Change text label",
  command=ChangeText)
tkgrid(ChangeText.but)
```

其中用了tkconfigure把标签控件label1与变量labelText建立了联系。

用tkentry指定文本输入条，如:

```
tcl.en <- tclVar(1.5)
en <- tkentry(root, textvariable=tcl.en, width=20)
```

这样可以用as.double(tclvalue(tcl.en))访问输入的值。

用tktext指定文本输入框，如:

```
txt <- tktext(root, width=20, height=10)
```

其中宽和高的单位是字符。访问内容如:

```
s <- as.character(tclvalue(tkget(txt, "0.0", "end")))
                          ## 全部内容
s <- gsub(",", " ", s)    ## 把逗号替换成空格
sc <- textConnection(s)   ## 生成输入流
x <- scan(sc)
close(sc)
```

用tkframe作为容器容纳控件。如:

```
fr <- tkframe(root, padx=5, pady=5)
```

其中padx和pady是在其必须的大小上加富余的空白。用relief和borderwidth指定边界形式，relief有raised, sunken, flat, ridge, solid, groove, 必须指定borderwidth如borderwidth=3才能显示效果。

用tkwidget生成一个一般Tk控件，用type指定类型，如:

```
fr <- tkwidget(root, type="labelframe", text="选择")
```

摆放控件用tkpack或tkgrid。tkpack沿上、下、左、右之一边缘摆放，tkgrid可以对准摆放，比如，la1和en1是输入的标签和输入条，la2和en2是另一对，则用

```
tkgrid(tklabel(root, text="输入部分"), columnspan=2)
tkgrid(la1, en1)
tkgrid(la2, en2)
```

其中第一个“输入部分”标签占了两列。如果tkgrid不指定摆放的行号和列号，则每次调用指定的控件摆放在同一行，下一次调用的指定的控件摆放在下一行。

用tkgrid.configure(控件名, sticky='wens') 可以规定控件的停靠方向(西、东、北、南)，缺省为中央，如果同时规定两个相反方向就代表允许向两个方向延伸。但是，为了grid摆放的控件能够在窗口放大时利用多出的空间，需要用tkgrid.rowconfigure(容器控件名, 行号, weight=1) 指定grid行扩充，用tkgrid.columnconfigure(容器控件名, 列号, weight=1) 指定grid列扩充，其中行号、列号从0计数，列之间以weight为权重竞争得到放大的空间(缺省的weight为0，不放大)。

在tkpack中用padx和pady指定左右和上下的间隙，如:

```
tkpack(ok.but, side="bottom", padx=10, pady=20)
```

在MS Windows环境下，为了能单独运行图形界面而不显示R原来的图形界面，可以在程序最后写上：

```
tcl.exit <- tclVar(0)
tkbind(root, "<Destroy>",
  function() tclvalue(tcl.exit) <- 1)
tkwait.variable(tcl.exit)
```

这样在批运行时不至于马上结束。批运行的命令是

```
\pathToRInstall\bin\rcmd BATCH testtk.r
```

或

```
\pathToRInstall\bin\rterm --vanilla < testtk.r 2>&1
```

# §6 字符串和正则表达式

## §6.1 字符串运算

注意S的字符型向量每个元素是一个字符串，所以字符型函数一般也是对每个元素操作的。

函数nchar(text)计算字符串长度。如

```
> nchar(c("a", "bc", "def"))
[1] 1 2 3
```

paste是最常用的字符串函数，用来把多个部分组合为一个字符型向量。各部分之间缺省用空格连接，可以用sep=指定连接用的字符串。各部分长度不同时短的自动循环使用。其他类型自动转换为字符型。如如

```
> paste("data", 1:3, ".txt", sep="")
[1] "data1.txt" "data2.txt" "data3.txt"
```

如果加上collapse=选项，可以把连接后的字符型向量各元素再连接起来，连接用的字符串由collapse=的值指定。如

```
> paste("data", 1:3, ".txt", sep="", collapse="***")
[1] "data1.txt***data2.txt***data3.txt"
```

用substring求各元素的子字符串，格式为

```
substring(text, first, last=1000000)
```

如substring("abcdefg", 3, 5)为"cde", substring("abcdefg", 3)为"cdefg",

## §6.2 正则表达式

R中的函数grep, grepl, sub, gsub, regexpr, gregexpr, strsplit与正则表达式有关。

grep的用法是

```
grep(pattern, x, ignore.case = FALSE, extended = TRUE,
     perl = FALSE, value = FALSE, fixed = FALSE,
     useBytes = FALSE, invert = FALSE)
```

其中pattern为正则表达式(regular expression)。返回结果是字符型向量x中匹配的元素下标集合，而不是在字符串中匹配的位置。grepl函数返回与x的元素对应的匹配逻辑值结果。

为了得到具体的匹配发生在字符串中的位置，使用regexpr函数，它返回一个与x等长的数组表示在相应元素中匹配的位置，对不能匹配的元素返回-1。regexpr只匹配第一处，如果需要在字符串值匹配所有位置，需用gregexpr，这时结果为一个列表，x的每个元素对应一个列表元素，每个列表元素为匹配发生的字符串位置的向量，并有一个属性表示匹配长度。

用substring函数提取子字符串。用strsplit函数拆分字符串，是paste的逆。

函数sub替换，gsub全局替换。例如，为了处理文本文件，把文本文件中两个连续的内容行连接为一行，不同行代表不同段，段间有空行，可以用如下程序把文件中的换行替换：

```
lines <- readLines('文件名')
s <- paste(lines, collapse="")
s <- gsub('([^\n]+)\n', '\\1', s)
s <- gsub('([^\n]+)\n', '\\1\n\n', s)
writeLines(strsplit(s, '\n', fixed=TRUE)[[1]],
           con='文件名')
```

选项extended缺省为真值，取假值是不推荐的用法。如果取fixed=TRUE则模式pattern认为是需要原样匹配的字符串，没有任何特殊解释。perl=TRUE则使用Perl的正则表达式语法规定, 很多高级一点的正则表达式都需要使用此选项。

**匹配单个字符** 原样匹配如

```
grep("my", "This is my home")==1
```

表示在x的下标1处有匹配。用regexpr如

```
> regexpr("my", "this is my home")
[1] 9
attr(,"match.length")
[1] 2
```

表示在字符串第9个字符处匹配，匹配长度为2。用gregexpr如

```
> gregexpr("my", "These are my home and my family")
[[1]]
[1] 11 23
```

```
attr(,"match.length")
[1] 2 2
```

表示在字符型向量的第一个元素的第11个字符和第23个字符处匹配。

正则表达式中句点可以匹配任意字符。如

```
> grep("sales.", c("sales1.xls", "order3.xls", "sales2.xls"))
[1] 1 3
```

表示第1个、第3个元素有匹配。

注意，正则表达式并不需要匹配整个字符串，而只需要匹配字符串的一部分。

如果需要匹配句点，用[.]表示。

**匹配一组字符** 模式[ns]a.[.]xls表示第一个字符是n或s，第二个字符是a，第三个字符任意，第四个字符是句点，然后是xls。方括号给定一个字符集合，匹配该字符必须取集合中的字符。模式也可以在中间位置匹配。如

```
> grep("[ns]a.[.]xls", c("sa1.xls", "dna2.xlss", "na3.xls"))
[1] 1 2 3
```

模式[Rr]eg[Ee]x可以匹配RegEx或Regex或regex或regEx。如果希望完全忽略大小写进行匹配，可以使用ignore.case=TRUE选项。

模式[ns]a[0-9][.]xls要求匹配的第三个字符为数字。字符区间如[a-z], [A-Z], [a-zA-Z], [a-zA-Z0-9]等。匹配一个16进制数字用[0-9A-Fa-f]。

模式[ns]a[^0-9][.xls]要求匹配的第三个字符不能为数字。在方括号内第一个位置的^表示对指定的范围取余集。

**使用元字符** 元字符是在正则表达式中有特殊含义的字符。比如句点可以匹配任意一个字符。左方括号代表字符集合的开始。所以元字符不能直接匹配自身，可以用[.]匹配一个句点。为匹配左方括号，用转义\[，而在R字符串中一个\必须用\\表示所以\[写成\\[，如

```
> grep("array x\\[0\\]", c("double x;", "array x[0]"))
[1] 2
```

也可以用[[]表示[，用[]]表示]，如

```
> grep("array x[[]0[]]", c("double x;", "array x[0]"))
[1] 2
```

表示空白的元字符有: \f 换页符; \n换行符; \r回车符; \t制表符; \v垂直制表符。比如，匹配Windows文件的空行用\r\n\r\n，但Unix的空行则用\r\r。

匹配空白字符用\s，等价于[\f\n\r\t\v]。\S匹配非空白字符。

匹配数字用\d，非数字用\D。如

```
> gregexpr("[ns]\\d[.]xls", c("n1.xls", "sa.xls"))
[[1]]
[1] 1
attr(,"match.length")
[1] 6

[[2]]
[1] -1
attr(,"match.length")
[1] -1
```

结果的两个列表元素对应被匹配的两个字符串，第一个字符串匹配成功，第二个字符串未匹配。

匹配字母、数字、下划线字符用\w，等价于[a-zA-Z0-9_]。\W匹配这些字符以外的情况。如

```
> grep("s\\w[.]", c("file-s1.xls", "s#.xls"))
[1] 1
```

两个待匹配字符串只有第一个有匹配。

在正则表达式中，十六进制数用\x引入，比如\0x0A对应\n字符。八进制数用\0引入，比如\011表示\t。如

```
> gregexpr("\\x0A", "abc\nefg\n")
[[1]]
[1] 4 8
attr(,"match.length")
[1] 1 1
```

可以使用Posix字符类表示一类字符，但在正则表达式中使用的时候要写在表示字符集合的方括号内，比如[:alpha:]表示字母，但在正则表达式中要写成[[:alpha:]]。这些字符类包括

- [:alnum:] 字母数字;
- [:alpha:] 字母;

- [:blank:] 空格或制表符；
- [:cntrl:] ASCII控制字符，为0－31号字符，以及127号字符；
- [:digit:] 数字;
- [:graph:] 和[:print:]一样，但不包括空格；
- [:lower:] 小写字母；
- [:print:] 可打印字符；
- [:punct:] 除[:alnum:]和[:print:]以外的所有字符；
- [:space:] 任何一种空白字符，包括空格、制表符、换页符、换行符、回车符；
- [:upper:] 大写字母；
- [:xdigit:] 十六进制数字。

比如

```
> gregexpr("sa[[:alpha:]]", c("psabc", "sa123"))
[[1]]
[1] 2
attr(,"match.length")
[1] 3

[[2]]
[1] -1
attr(,"match.length")
[1] -1
```

又如

```
> grep("s[[:lower:][:digit:]]", c("sa", "s1", "sA"))
[1] 1 2
```

**重复匹配** 在一个字符或字符集合后加后缀+表示一个或多个前一字符。比如

```
> regexpr("sa[[:digit:]]+", c("sa1", "dsa123"))
[1] 1 2
attr(,"match.length")
[1] 3 5
```

中模式[[:digit:]]+匹配一个或多个数字。再比如

```
> grepl("\\w+@\\w+[.]\\w+", "abc123@efg.com")
[1] TRUE
```

又如

```
> regexpr("[[:alnum:].]+@[[:alnum:]]+[.][[:alnum:]]+", "ab.12@efg.com")
[1] 1
attr(,"match.length")
[1] 13
```

在一个字符或字符集合后加后缀+表示零个或多个前一字符。后缀?表示零个或一个前一字符。如"https?://[[:alnum:]./]+"可以匹配http或https开始的网址。

在后缀大括号中写一个整数表示精确的重复次数。如类似#FF00FF这样的颜色编码可以用模式”#[[:xdigit:]]{6}”来描述。也可以指定重复的最小和最大次数，比如月日年的日期格式可以用

```
[[:digit:]]{1,2}[-/][[:digit:]]{1,2}[-/][[:digit:]]{2,4}
```

来匹配。如

```
> grep("[[:digit:]]{1,2}[-/][[:digit:]]{1,2}[-/][[:digit:]]{2,4}",
       "2/4/1998")
[1] 1
```

重复数允许指定为0。后缀3,表示前一模式必须至少重复3次。

+, *, {n,}都没有指定重复上限，其匹配时贪婪型的，即尽可能找到最长的匹配。比如

```
> regexpr("<[Bb]>.*</[Bb]>",
          "<B>First</B> abc <B>Second</B>")
[1] 1
```

```
attr(,"match.length")
[1] 30
```

本来是想匹配一组<B>和</B>包围的内容，但是却匹配到了下一个</B>的后面。如果要求尽可能短的匹配，使用*?, *+?, *{n,}? 等“懒惰型”元字符。如

```
> regexpr("<[Bb]>.*?</[Bb]>",
          "<B>First</B> abc <B>Second</B>")
[1] 1
attr(,"match.length")
[1] 12
```