R for Beginners

Chinese Edition 2.0

Emmanuel Paradis
Institut des Sciences de l'Évolution
Université Montpellier II
F-34095 Montpellier cédex 05
France
E-mail: *paradis@isem.univ-montp2.fr*
Co-translated by: XF Wang, YH Xie, JT Li and GH Ding

中文版说明

“R for beginners”是一本公认的经典手册，非常适合R的初学者。英文原版初著于2002年，而此稿是基于作者在2005年重新修订的第二版。

Emmanuel Paradis博士为本稿提供了原版所有LaTeX源文件。翻译工作由四名志愿者共同完成（Chap1–2: 王学枫；Chap3: 谢益辉；Chap4: 李军焘；Chap5–7: 丁国徽）。由华东师范大学汤银才老师负责本文档的编辑校订。北京大学李东风老师审阅了全稿并提出了大量宝贵意见。在此一并表示衷心感谢！

编译仓促，差错难免，亟盼诸R友襄正。任何意见请通过pwxf@hotmail.com联系我们。

译者

2006年4月

版权 © 2002, 2005, Emmanuel Paradis

Permission is granted to make and distribute copies, either in part or in full and in any language, of this document on any support provided the above copyright notice is included in all copies. Permission is granted to translate this document, either in part or in full, in any language provided the above copyright notice is included.

目录

1 导言

该手册是关于R的一个入门教材.由于主要针对初学者，我将重点放在了对R的工作原理的解释上。R涉及广泛，因此对于初学者来讲，了解和掌握一些基本概念及原理是很有必要的。在打下扎实的基础后，进行更深入的学习将会变得轻松许多。本着深入浅出的宗旨，本手册将大量配合图表等形式，尽可能使用通俗的语言，使读者容易理解而并不失细节。

R是一个有着统计分析功能及强大作图功能的软件系统，是由Ross Ihaka和Robert Gentleman1共同创立。R语言可以看作是由AT&T贝尔实验室所创的S语言发展出的一种方言[1]。因此，R即是一种软件也可以说是一种语言。S语言现在主要内含在由Insightful[2]公司经营的S-PLUS软件中。R和S在设计理念上存在有着许多不同：关于这方面的详细内容大家可以参考Ihaka & Gentleman (1996) 或R-FAQ[3]，该文档同时随R一起发布。

R是在GNU协议General Public Licence[4]下免费发行的，它的开发及维护现在则由R开发核心小组*R Development Core Team*具体负责。

R的安装文件有多种形式，有在Unix 或Linux系统下所需的一些源代码(主要用C及Fortran 编写)，及在Windows, Linux及Macintosh上使用的预编译二进制码。这些安装文件以及安装说明都可以在*Comprehensive R Archive Network* (CRAN)[5] 网站上下载。该网站提供的关于Linux的安装文件只适用于较新版本的Linux。详情请参考CRAN网站。

R内含了许多实用的统计分析及作图函数。作图函数能将产生的图片展示在一个独立的窗口中，并能将之保存为各种形式的文件(jpg, png, bmp, ps, pdf, emf, pictex, xfig; 具体形式取决于操作系统)。统计分析的结果也能被直接显示出来，一些中间结果(如*P*-值，回归系数，残差等)既可保存到专门的文件中，也可以直接用作进一步的分析。

在R语言中，使用者可以使用循环语句来连续分析多个数据集，也可将多个不同的统计函数结合在一个语句中执行更复杂的分析。R使用者还可以借鉴网上提供的用S编写的大量程序[6],而且大多数都能被R直接调用。

非专业人员起初可能觉得R相对比较复杂。其实，R的一个非常突出的优点正是它的灵活性。一般的软件往往会直接展示分析的结果，而R则将这些结果都存在一个对象“object”里面，所以常常在分析执行结束后并不显示任何结

[1] Ihaka R. & Gentleman R. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5: 299–314.

[2] http://www.insightful.com/products/splus/default.asp

[3] http://cran.r-project.org/doc/FAQ/R-FAQ.html

[4] For more information: http://www.gnu.org/

[5] http://cran.r-project.org/

[6] For example: http://stat.cmu.edu/S/

果。使用者可能会对此感到困惑，其实这样的特点是非常有用的，因为我们可以选择的从结果中只抽出我们感兴趣的部分。例如，我们要运行20个回归分析而只想比较其回归系数，在R中就可以选择只显示所有分析得出的回归系数，这样结果仅仅占了一排，而用有些软件可能会一下打开20个窗口。而在下面的章节中，我们会看到更多能展示R相比传统软件更为灵活优越的例子。

2 基本原理与概念

如果R已经被安装在你的计算机中，它就能立即运行一些可执行的命令了。R默认的命令提示符是'>'，它表示正在等待输入命令。如在Windows系统中打开Rgui.exe，就能直接运行下拉菜单中的一些操作命令(如在线帮助，打开文件...)。到这里，有些人可能会急着想知道更多的语句命令。其实，在学习这些内容前，了解掌握一些R的基本工作原理是非常有必要的。这正是本章所要讲的主要内容。

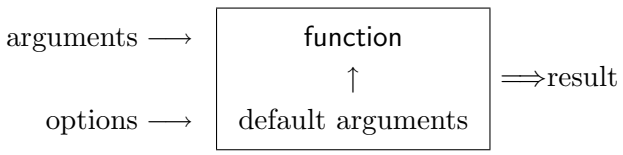
本章首先简要描述R的工作原理。在第二节中，我将介绍一些基本的赋值分配("assign") 的操作，如怎样产生对象(object)，如何操作管理这些对象等.最后简要介绍R中非常有用的在线帮助。

2.1 基本原理

因为R是一种编程语言，一些对编程不太熟悉的人可能会望而却步。这种障碍其实是完全没有必要，首先，R是一种解释型语言，而不是编译语言，也就意味着输入的命令能够直接被执行，而不需要像一些语言要首先构成一个完整的程序形式(如C，Fortan, Pascal, ...)。

第二，R的语法非常之简单和直观。例如，线性回归的命令`lm(y ~ x)` 表示"以x为自变量，y为反应量来拟合一个线性模型"。合法的R函数总是带有圆括号的形式，即使括号内没有内容(如`ls()`)。如果直接输入函数名而不输入圆括号，R则会自动显示该函数的一些具体内容。在本手册中除在部分文字已作出清楚的说明外，所有的函数后都接有圆括号以区别于对象(object)。

当R运行时，所有变量，数据，函数及结果都以对象(*objects*)的形式存在计算机的活动内存中，并冠有相应的名字代号。我们可以通过用一些运算符(如算术，逻辑，比较等)和一些函数(其本身也是对象)来对这些对象进行操作。运算操作非常简单，其细节将留在下章讨论(p. 26). 关于R中的函数可用下面的图例来形象的描述：



上图中的参量(argument)可能是一些对象(如数据，方程，算式...)。有些参量在函数里被预设为缺省值，用户则可按需对其作个别的修改。所以运行一个R函数可能不需要设定任何参量，原因是所有的参量都可以被默认为缺

省值，当然也有可能该函数本身就不含任何参量。由于这里主要是讲述R的工作原理，对R函数的介绍将不再展开，在后面的章节中我们会看到关于构建及使用各种函数的详细内容(p. 68)。

在R中进行的所有操作都是针对存储在活动内存中的对象的, 因此就不涉及到任何临时文件夹的使用(Fig. 1)。对数据，结果或图表的输入与输出都是通过在对计算机硬盘中的文件读写而实现。用户通过输入一些命令调用函数，分析得出的结果可以被直接显示在屏幕上，也可以被存入某个对象或被写入硬盘(如图片对象)。因为产生的结果本身就是一种对象，所以它们也能被视为数据并能像一般数据那样被处理分析。数据文件即可从本地磁盘读取也可通过网络传输从远程服务器端获得。

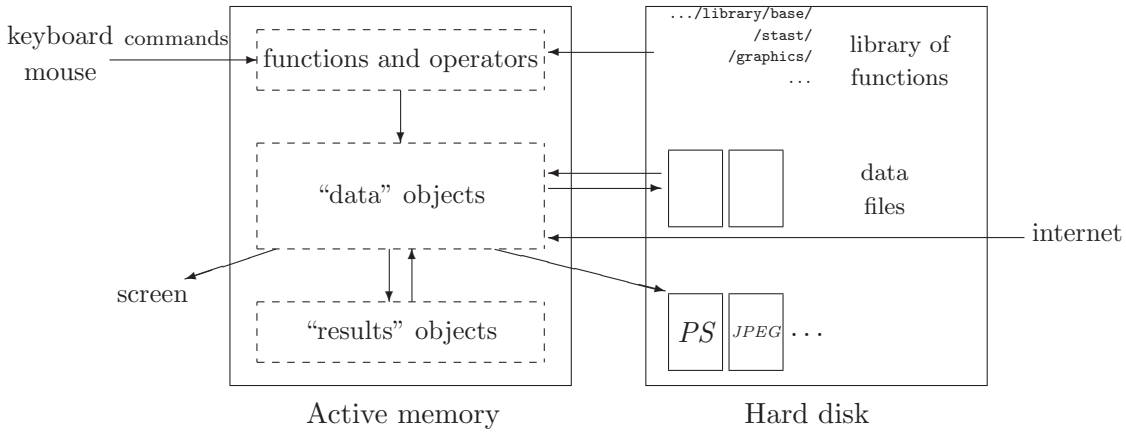


图 1: R工作原理示意图.

所有能使用的R函数都被包含在一个库(library) 中, 该库存放在磁盘的R_HOME/library 目录下(R_HOME 是最初安装R的地址)。这个目录下含有具有各种功能的包(*packages*)，各个包也是按照目录的方式组织起来的。其中名为base的包可以算是R的核心，因为它内嵌了R语言中所有像数据读写与操作这些最基本的函数。在上述目录中的每个包内，都有一个子目录R，这个目录里又都含有一个与此包同名的文件(例如在包base中，有这样一个文件R_HOME/library/base/R/base)。该文件正是存放所有函数的地方。

R语言中最简单的命令莫过于通过输入一个对象的名字来显示其内容了。例如，一个名为n的对象，其内容是数值10：

```
> n
[1] 10
```

方括号中的数字1表示从n的第一个元素开始显示。其实该命令的功能在

这里于函数`print` 相似,输出结果与`print(n)` 相同(但有些情况下，例如内嵌在一个函数或循环中时，就必须得用`print`函数)。

对象的名字必须是以一个字母开头(A–Z 或a–z), 中间可以包含字母，数字(0–9)，点(.)及下划线(_). 因为R对对象的名字区分大小写，所以`x` 和`X`就可以代表两个完全不同的对象(在Windows操作系统中也是如此)。

2.2 对象的产生，排列及删除

一个对象可以通过赋值操作来产生，R语言中的赋值(“assign”) 符号一般是由一个尖括号与一个负号组成的箭头形标志。该符号可以是从左到右的方向，也可以相反:

```
> n <- 15
> n
[1] 15
> 5 -> n
> n
[1] 5
> x <- 1
> X <- 10
> x
[1] 1
> X
[1] 10
```

如果该对象已经存在，那么它以前的值将会自动被新值冲掉(这种修改只会影响内存中的数据，操作结果暂时不会被保存到硬盘中)。在R中给对象赋值有多种形式，可以是直接赋一个数值，也可以是一个算式或一个函数的结果:

```
> n <- 10 + 2
> n
[1] 12
> n <- 3 + rnorm(1)
> n
[1] 2.208807
```

运行`rnorm(1)` 将产生一个服从平均数为0标准差为1的标准正态分布的随机变量(p. 18)。当然你也可以只是输入函数或表达式而不把它的结果赋给某个对象，但如果这样在窗口中展示的结果将不会被保存到内存中:

```
> (10 + 2) * 5
[1] 60
```

本文中，在不影响读者理解的情况下，一些赋值符号将会被省略。

函数ls的功能是显示所有在内存中的对象：只会列出对象名，如:

```
> name <- "Carmen"; n1 <- 10; n2 <- 100; m <- 0.5
> ls()
[1] "m"    "n1"   "n2"   "name"
```

注意在R中应该用分号来隔开同一行中的不同命令语句。如果只要显示出在名称中带有某个指定字符的对象，则通过设定选项pattern来实现(可简写为pat)):

```
> ls(pat = "m")
[1] "m"    "name"
```

如果进一步限为显示在名称中以某个字母开头的对象，则可:

```
> ls(pat = "^m")
[1] "m"
```

运行函数ls.str()将会展示内存中所有对象的详细信息:

```
> ls.str()
m :  num 0.5 n1 :  num 10 n2 :  num 100 name :  chr "Carmen"
```

选项pattern在这里同样适用。在ls.str函数中另一个非常有用的选项是max.level, 它将规定显示所有关对象信息的详细级别。缺省情况下，ls.str 将会列出关于对象的所有信息，包括数据框，矩阵，数据列表的列数信息。因此展示结果可能会很长。但如果设定max.level =-1 就可以避免这种情况了:

```
> M <- data.frame(n1, n2, m)
> ls.str(pat = "M")
M : 'data.frame':       1 obs. of  3 variables:
 $ n1: num 10
 $ n2: num 100
 $ m : num 0.5
> ls.str(pat="M", max.level=-1)
M : 'data.frame':       1 obs. of  3 variables:
```

要在内存中删除某个对象，可利用函数rm: 运行rm(x)将会删除对象x,运行rm(x,y) 将会删除对象x和y，而运行rm(list=ls())则会删除内存中的所有对象. 在上面所讲的ls() 函数中的一些选项同样也可以运用到rm中来，以选择的删除一些对象，如: rm(list=ls(pat="^m"))。

2.3 在线帮助

R中给予的在线帮助能提供关于如何使用函数的非常有用的信息。关于某个特定函数的帮助能够直接被调出来，如运行：

```
> ?lm
```

会立即显示关于函数`lm()`(线性模型)的帮助页面。命令`help(lm)`和`help("lm")`具有同样的效果。但在查询关于某特殊语法意义字符的帮助时必须用后一种形式，如：

```
> ?*
Error: syntax error
> help("*")
Arithmetic              package:base              R Documentation

Arithmetic Operators ...
```

启动帮助将会打开一个页面(取决于操作系统)，第一行一般会显示某函数或操作命令的所属的包(package)，然后是标题，标题下面是则是一些详细信息。

Description: brief description.

Usage: for a function, gives the name with all its arguments and the possible options (with the corresponding default values); for an operator gives the typical use.

Arguments: for a function, details each of its arguments.

Details: detailed description.

Value: if applicable, the type of object returned by the function or the operator.

See Also: other help pages close or similar to the present one.

Examples: some examples which can generally be executed without opening the help with the function `example`.

对初学者而言，参考帮助中**Examples**部分的信息是很有用的。而一般应该仔细阅读**Arguments**中的一些说明也是非常有必要的。帮助中还包含了其它一些说明部分，如**Note**, **References**或**Author(s)**等。

默认状态下，函数help只会在被载入内存中的包中搜索。选项try.all.package在缺省值是FALSE, 但如果把它设为TRUE，则可在所有包中进行搜索：

```
> help("bs")
No documentation for 'bs' in specified packages and libraries:
you could try 'help.search("bs")'
> help("bs", try.all.packages = TRUE)
Help for topic 'bs' is not in any loaded package
but can be found in the following packages:

  Package               Library
  splines               /usr/lib/R/library
```

但注意在这种情况下，不会显示关于函数bs的帮助页面，如果使用者确实想打开这样的页面而所属包又没有被载入内存时，可以使用package这个选项:

```
> help("bs", package = "splines")
bs                package:splines                R Documentation

B-Spline Basis for Polynomial Splines

Description:

     Generate the B-spline basis matrix for a polynomial spline.
...
```

Html格式的帮助可以通过输入下面的函数启动

```
> help.start()
```

在html格式的帮助页面中还可以使用关键词进行搜索。在**See Also**部分中，可以通过超文本链接到其他相关函数的帮助页面。使用关键词的搜索在R中也可以通过函数help.search来实现。这种方法能在所有已安装的包中搜索包含给定字符串的相关内容。例如，运行help.search("tree")会列出所有在帮助页面含有"tree"的函数。注意如果有一些包是最近才安装的，应该首先使用函数help.search中的rebuild选项来刷新数据库(e.g., help.search("tree", rebuild = TRUE))。

使用函数apropos则能找出所有在名字中含有指定字符串的函数，但只会在被载入内存中的包中进行搜索:

```
> apropos(help)
[1] "help"         ".helpForCall" "help.search"
[4] "help.start"
```

3 R的数据操作

3.1 对象

我们已经看到R通过一些对象来运行，当然首先这些对象是用它们的名称和内容来刻画的，其次也通过对象的数据类型即*属性*来刻画。为了理解这些属性的用处，我们以一个在{1,2,3}中取值的变量为例：这个变量可以是一个整数变量（例如巢中蛋的个数），或者也可以是一个分类变量的编码（例如某些甲壳类动物的三种性别：雄、雌和雌雄同体）。

显然对这个变量的统计分析在以上两例中将是不相同的，对象的属性在R中提供着所需的信息。更技术性也更一般地说，对于作用于一个对象的函数，其表现将取决于对象的属性。

所有的对象都有两个*内在*属性：*类型*和*长度*。类型是对象元素的基本种类，共有四种：数值型，字符型，复数型[7]和逻辑型(FALSE或TRUE)，虽然也存在其它的类型，但是并不能用来表示数据，例如函数或表达式；长度是对象中元素的数目。对象的类型和长度可以分别通过函数mode和length得到。

```
> x <- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> A <- "Gomphotherium"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"
```

无论什么类型的数据，缺失数据总是用NA(*不可用*)来表示；对很大的数值则可用指数形式表示：

```
> N <- 2.1e23
> N
[1] 2.1e+23
```

R可以正确地表示无穷的数值，如用Inf和-Inf表示$\pm\infty$，或者用NaN(*非数字*)表示不是数字的值。

[7]本手册中不讨论复数型

```
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

字符型的值输入时须加上双引号"，如果需要引用双引号的话，可以让它跟在反斜杠\后面；这两个字符合一起\"在某些函数如cat的输出显示或write.table写入磁盘(参见p. 14,函数的qmethod选项)时会被以特殊的方式处理。

```
> x <- "Double quotes \" delimitate R's strings."
> x
[1] "Double quotes \" delimitate R's strings."
> cat(x)
Double quotes " delimitate R's strings.
```

也有另一种表示字符型变量的方法，即用单引号(')来界定变量，这种情况下不需要用反斜杠来引用双引号（但是引用单引号时必须要用！）

```
> x <- 'Double quotes " delimitate R\'s strings.'
> x
[1] "Double quotes \" delimitate R's strings."
```

下表给出了表示数据的对象的类别概览：

对象	**类型**	**是否允许同一个对象中有多种类型？**
向量	数值型，字符型，复数型，或 逻辑型	否
因子	数值型或 字符型	否
数组	数值型，字符型，复数型，或 逻辑型	否
矩阵	数值型，字符型，复数型，或 逻辑型	否
数据框	数值型，字符型，复数型，或 逻辑型	是
时间序列(ts)	数值型，字符型，复数型，或 逻辑型	否
列表	数值型，字符型，复数型，逻辑型，函数，表达式，...	是

向量是一个变量，其意思也即人们通常认为的那样；因子是一个分类变量；数组是一个k维的数据表；矩阵是数组的一个特例，其维数$k = 2$。注意，数组或者矩阵中的所有元素都必须是同一种类型的；数据框是由一个或几个向量和（或）因子构成，它们必须是等长的，但可以是不同的数据类型；“ts”表示时间序列数据，它包含一些额外的属性，例如频率和时间；列表可以包含任何类型的对象，包括列表！

对于一个向量，用它的类型和长度足够描述数据；而对其它的对象则另需一些额外信息，这些信息由*外在*的属性给出。这些属性中的是表示对象维数的*dim*，比如一个2行2列的的矩阵，它的`dim`是一对数值[2,2]，但是其长度是4。

3.2 在文件中读写数据

对于在文件读取和写入的工作，R使用工作目录来完成。可以使用命令`getwd()` (*获得工作目录*)来找到目录，使用命令`setwd("C:/data")`或者`setwd("/home/paradis/R")`来改变目录。如果一个文件不在工作目录里则必须给出它的路径[8]。

R可以用下面的函数读取存储在文本文件（ASCII）中的数据：`read.table` (其中有若干参数，见后文)，`scan`和`read.fwf`。R也可以读取以其他格式的文件(Excel, SAS, SPSS, ...) 和访问SQL类型的数据库，但是基础包中并不包含所需的这些函数。这些功能函数对于R的高级应用是十分有用的，但是我们在这里将读取文件限定在ASCII格式。

函数`read.table`用来创建一个数据框，所以它是读取表格形式的数据的主要方法。举例来说，对于一个名为data.dat的文件，命令：

```
> mydata <- read.table("data.dat")
```

将创建一个数据框名为`mydata`，数据框中每个变量也都将被命名，缺省值为`V1`, `V2`, ...并且可以单独地访问每个变量，代码为：`mydata$V1`, `mydata$V2`, ..., 或者用`mydata["V1"]`, `mydata["V2"]`, ..., 或者还有一种方法，`mydata[, 1]`, `mydata[,2 ]`, ...[9] 这里有一些选项的缺省值(即如果用户不设定那么R将自动使用的值)见于下表：

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".",
           row.names, col.names, as.is = FALSE, na.strings = "NA",
           colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
```

[8]在Windows中，为Rgui.exe创建一个快捷方式是比较有用的，在快捷方式“属性”的“起始位置”中改变目录，然后用此快捷方式启动R时这个目录就会成为工作目录

[9]注意这几种方法的结果是有区别的：`mydata$V1`和`mydata[, 1]`是向量，而`mydata["V1"]`是数据框。后面(p. 19)将会讲到关于处理对象的详情。

```
strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#")
```

file	文件名（包在""内，或使用一个字符型变量），可能需要全路径（注意即使是在Windows下，符号\ 也不允许包含在内，必须用 / 替换），或者一个URL链接（http://...）（用URL对文件远程访问）
header	一个逻辑值(FALSE or TRUE)，用来反映这个文件的第一行是否包含变量名
sep	文件中的字段分离符，例如对用制表符分隔的文件使用sep="\t"
quote	指定用于包围字符型数据的字符
dec	用来表示小数点的字符
row.names	保存着行名的向量,或文件中一个变量的序号或名字,缺省时行号取为1, 2, 3, ...
col.names	指定列名的字符型向量(缺省值是：V1, V2, V3, ...)
as.is	控制是否将字符型变量转化为因子型变量(如果值为FALSE)，或者仍将其保留为字符型（TRUE）。as.is可以是逻辑型，数值型或者字符型向量，用来判断变量是否被保留为字符。
na.strings	代表缺失数据的值(转化为NA)
colClasses	指定各列的数据类型的一个字符型向量
nrows	可以读取的最大行数(忽略负值)
skip	在读取数据前跳过的行数
check.names	如果为TRUE，则检查变量名是否在R中有效
fill	如果为TRUE且非所有的行中变量数目相同，则用空白填补
strip.white	在sep已指定的情况下，如果为TRUE，则删除字符型变量前后多余的空格
blank.lines.skip	如果为TRUE，忽略空白行
comment.char	一个字符用来在数据文件中写注释，以这个字符开头的行将被忽略（要禁用这个参数，可使用comment.char = ""）

read.table的几个变种因为使用了不同的缺省值可以用在几种不同情况下：

```
read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, ...)
read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",",
          fill = TRUE, ...)
read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".",
           fill = TRUE, ...)
read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=",",
            fill = TRUE, ...)
```

函数scan比read.table要更加灵活，它们的区别之一是前者可以指定变量的类型，例如：

```
> mydata <- scan("data.dat", what = list("", 0, 0))
```

读取了文件data.dat中三个变量，第一个是字符型变量，后两个是数值型变量。另一个重要的区别在于scan()可以用来创建不同的对象，向量，矩阵，数据框，列表... 在上面的例子中，mydata是一个有三个向量的列表。在缺省情况下，也就是说，如果what被省略，scan()将创建一个数值型向量。如果读取的数据类型与缺省类型或指定类型不符，则将返回一个错误信息。这些选项在下面进行说明。

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
     quote = if (sep=="\n") "" else "'\"", dec = ".",
     skip = 0, nlines = 0, na.strings = "NA",
     flush = FALSE, fill = FALSE, strip.white = FALSE, quiet = FALSE,
     blank.lines.skip = TRUE, multi.line = TRUE, comment.char = "")
```

file	文件名(在""之内), 可能包含它的路径（符号\ 不允许使用，必须用/替代, 即使是在Windows下面），或者使用一个URL链接（http://...）；如果file=””，数据从键盘输入（使用一个空白行终止输入）
what	指定数据的类型（缺省值为数值型）
nmax	要读取数据的最大数量，如果what是一个列表，nmax则是可以读取的行数（在缺省情况下，scan读取到文件最末端为止的所有数据）
n	要读取数据的最大数量(在缺省情况下，没有限制)
sep	文件中的字段分隔符
quote	用来包围字符型值
dec	用来表示小数点的字符
skip	在读取数据前跳过的行数
nlines	要读取的行数
na.string	表示缺失数据的字符串（转化为为NA）
flush	一个逻辑值，如果为TRUE，当读取完指定列数后scan将转到下一行（这样就允许用户在数据文件中添加注释，即添加在指定列数之后）
fill	如果为TRUE，且非所有的行中变量数目相同，则用空白填补
strip.white	在sep已指定的情况下，如果为TRUE，则删除字符型变量前后多余的空格
quiet	一个逻辑值，如果为FALSE，scan显示一行信息说明哪些字段被读取
blank.lines.skip	如果为TRUE，忽略空白行
multi.line	当what是一个列表时，若为FALSE则表示列表中每个个体的所有变量都在同一行中
comment.char	指定注释开始字符，一行中以这个字符开头的部分将被忽略（缺是禁用此选项）

函数read.fwf可以用来读取文件中一些固定宽度格式的数据：

```
read.fwf(file, widths, sep="\t", as.is = FALSE,
         skip = 0, row.names, col.names, n = -1, ...)
```

除了widths用来说明读取字段的宽度外，选项与read.table()基本相同。举例来说，如果在一个名为data.txt的文件中有一组如右所示的数据，可以读取这些数据用下面的命令：

```
A1.501.2
A1.551.3
B1.601.4
B1.651.5
C1.701.6
C1.751.7
```

```
> mydata <- read.fwf("data.txt", widths=c(1, 4, 3))
> mydata
  V1   V2  V3
1  A 1.50 1.2
2  A 1.55 1.3
3  B 1.60 1.4
4  B 1.65 1.5
5  C 1.70 1.6
6  C 1.75 1.7
```

3.3 存储数据

函数write.table可以在文件中写入一个对象，一般是写一个数据框，也可以是其它类型的对象（向量，矩阵...）。参数和选项：

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"))
```

x	要写入的对象的名称
file	文件名（缺省时对象直接被“写”在屏幕上）
append	如果为TRUE则在写入数据时不删除目标文件中可能已存在的数据，采取往后添加的方式
quote	一个逻辑型或者数值型向量：如果为TRUE，则字符型变量和因子写在双引号""中；若quote是数值型向量则代表将欲写在""中的那些列的列标。(两种情况下变量名都会被写在""中；若quote = FALSE则变量名不包含在双引号中)
sep	文件中的字段分隔符
eol	使用在每行最后的字符("\n"表示回车)
na	表示缺失数据的字符
dec	用来表示小数点的字符
row.names	一个逻辑值，决定行名是否写入文件；或指定要作为行名写入文件的字符型向量
col.names	一个逻辑值（决定列名是否写入文件）；或指定一个要作为列名写入文件中的字符型向量
qmethod	若quote=TRUE，则此参数用来指定字符型变量中的双引号"如何处理：若参数值为"escape" (或者"e"，缺省)每个"都用\"替换；若值为"d"则每个"用""替换

若想用更简单的方法将一个对象写入文件，可以使用命令write(x, file ="data.txt"),其中x是对象的名字（它可以是向量，矩阵，或者数组）。这里有两个选项：nc(或者ncol)，用来定义文件中的列数（在缺省情况下，如果x是字符型数据，则nc=1；对于其它数据类型nc=5），和append（一个逻辑值），若为TRUE则添加数据时不删除那些可能已存在在文件中的数据；若为FALSE（缺省值）则删除文件中已存在的数据。

要记录一组任意数据类型的对象，我们可以使用命令save(x, y, z, file= "xyz.RData")。可以使用选项ASCII＝TRUE使得数据在不同的机器之间更简易转移。数据（用R的术语来说叫做工作空间）可以在使用load("xyz.RData")之后被加载到内存中。函数save.image()是save(list =ls(all=TRUE), file=".RData")的一个简捷方式。

3.4 生成数据

3.4.1 规则序列

例如一个从1到30的规则整数序列，可以这样产生：

```
> x <- 1:30
```

这个结果向量x有30个元素。算子‘:’的优先级可从如下表达式中看出：

```
> 1:10-1
 [1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
```

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

函数seq可以生成如下的实数序列：

```
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

其中第一个数字表示序列的起点，第二个表示终点，第三个是生成序列的步长。也可以这样使用：

```
> seq(length=9, from=1, to=5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

还可以用函数c直接输入数值：

```
> c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

如果想用键盘输入一些数据也是可以的，只需要直接使用默认选项的scan函数：

```
> z <- scan()
1: 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
10:
Read 9 items
> z
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

函数rep用来创建一个所有元素都相同的向量：

```
> rep(1, 30)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

函数sequence创建一系列连续的整数序列，每个序列都以给定参数的数值结尾：

```
> sequence(4:5)
[1] 1 2 3 4 1 2 3 4 5
> sequence(c(10,5))
 [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5
```

函数gl(*生成不同的水平/层次数据*)十分有用，因为它能产生规则的因子序列。这个函数的用法是gl(k,n)，其中k是水平数（或类别数），n是每个水平重复的次数。此函数有两个选项：length用来指定产生数据的个数，labels用来指定每个水平因子的名字。例如：

```
> gl(3, 5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels:  1 2 3
> gl(3, 5, length=30)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels:  1 2 3
> gl(2, 6, label=c("Male", "Female"))
 [1] Male   Male   Male   Male   Male   Male
 [7] Female Female Female Female Female Female
Levels:  Male Female
> gl(2, 10)
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels:  1 2
> gl(2, 1, length=20)
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels:  1 2
> gl(2, 2, length=20)
 [1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels:  1 2
```

最后，expand.grid()创建一个数据框，结果是把各参数的各水平完全搭配：

```
> expand.grid(h=c(60,80), w=c(100, 300), sex=c("Male", "Female"))
   h   w    sex
1 60 100   Male
2 80 100   Male
3 60 300   Male
4 80 300   Male
5 60 100 Female
6 80 100 Female
7 60 300 Female
8 80 300 Female
```

3.4.2 随机序列

分布名称	函数
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomial	`rbinom(n, size, prob)`
multinomial	`rmultinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistics	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`

在统计学中，产生随机数据是很有用的，R可以产生多种不同分布下的随机数序列。这些分布函数的形式为`rfunc(n,p1,p2,...)`，其中`func`指概率分布函数，`n`为生成数据的个数，`p1`, `p2`, ... 是分布的参数数值。上面的表给出了每个分布的详情和可能的缺省值（如果没有给出缺省值，则意味着用户必须指定参数）。

大多数这种统计函数都有相似的形式，只需用`d`、`p`或者`q`去替代`r`，比如密度函数(`d`*`func`*`(x, ...)`)，累计概率密度函数（也即分布函数）(`p`*`func`*`(x, ...)`)和分位数函数(`q`*`func`*`(p, ...)`，$0 < p < 1$)。最后两个函数序列可以用来求统计假设检验中P值或临界值。例如，显著性水平为5%的正态分布的双侧临界值是：

```
> qnorm(0.025)
[1] -1.959964
> qnorm(0.975)
[1] 1.959964
```

对于同一个检验的单侧临界值，根据备择假设的形式使用`qnorm(0.05)`或`1 - qnorm(0.95)`。

一个检验的P值，比如自由度$df = 1$的$\chi^2 = 3.84$：

```
> 1 - pchisq(3.84, 1)
[1] 0.05004352
```

3.5 使用对象

3.5.1 创建对象

我们在前面看到了用赋值操作创建对象的不同方法；在这样的创建中对象的数据类型和模式通常都已经预先确定了。在创建一个对象时是有可能指定它的数据类型、长度、类别等等的。从处理对象的角度来看这些方法是很有趣的。举例来说，我们可以创建一个空的对象并且逐步修改其中的元素，这比把所有的元素一起用`c()`放进去更有效。在这里也可以使用下标系统，后面我们将会看到(p. 27)。

从一些对象创建其它对象也是很方便的，比如，若想拟合一系列线性模型，可以将这一系列模型的公式放在一个列表中，然后不断地取出元素（即那些公式）插入到函数`lm`中。

在这个R的学习阶段，我们很有必要了解下面的函数和数据结构。直接创建数据结构不仅能让我们对数据有更好的理解，而且也会更深入地领会前文中提到的一些概念。

向量（Vector） 函数`vector`有两个参数：类型(`mode`)和长度(`length`)，创建的向量中元素值取决于参数所指定的数据类型：数值型向量则元素值都为0，逻辑型都为`FALSE`，字符型都为`""`。以下三个函数有几乎相同的效果（创建一个向量）并且只有一个参数即长度：`numeric()`，`logical()`，和`character()`。

因子（Factor） 一个因子不仅包括分类变量本身还包括变量不同的可能水平（即使它们在数据中不出现）。因子函数`factor`用下面的选项创建一个因子：

```
factor(x, levels = sort(unique(x), na.last = TRUE),
       labels = levels, exclude = NA, ordered = is.ordered(x))
```

`levels`用来指定因子可能的水平（缺省值是向量`x`中互异的值）；`labels`用来指定水平的名字；`exclude`表示从向量`x`中剔除的水平值；`ordered`是一个逻辑型选项用来指定因子的水平是否有次序。回想数值型或字符型的`x`。下面有一些例子：

```
> factor(1:3)
[1] 1 2 3
```

```
Levels:  1 2 3
> factor(1:3, levels=1:5)
[1] 1 2 3
Levels:  1 2 3 4 5
> factor(1:3, labels=c("A", "B", "C"))
[1] A B C
Levels:  A B C
> factor(1:5, exclude=4)
[1] 1  2  3  NA 5
Levels:  1 2 3 5
```

函数levels用来提取一个因子中可能的水平值：

```
> ff <- factor(c(2, 4), levels=2:5)
> ff
[1] 2 4
Levels:  2 3 4 5
> levels(ff)
[1] "2" "3" "4" "5"
```

矩阵（Matrix） 一个矩阵实际上是有一个附加属性（维数dim）的向量，维数即为一个长度为2的向量，用来指定矩阵的行数和列数。一个矩阵可以用函数matrix来创建：

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

选项byrow表示数据给出的值是要按列填充（缺省值）还是按行填充（如果为TRUE）。可以通过选项dimnames给行列命名。

```
> matrix(data=5, nr=2, nc=2)
     [,1] [,2]
[1,]    5    5
[2,]    5    5
> matrix(1:6, 2, 3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6, 2, 3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

另一种创建矩阵的方法是给维数适当的赋值（初始值为NULL）：

```
> x <- 1:15
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> dim(x)
NULL
> dim(x) <- c(5, 3)
> x
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

数据框（Data frame） 前面我们已经看到一个数据框可以由函数read.table间接创建；这里也可以用函数data.frame来创建。数据框中的向量必须有相同的长度，如果其中有一个比其它的短，它将“循环”整数次（以使得其长度与其它向量相同）：

```
> x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
> data.frame(x, n)
  x  n
1 1 10
2 2 10
3 3 10
4 4 10
> data.frame(x, M)
  x  M
1 1 10
2 2 35
3 3 10
4 4 35
> data.frame(x, y)
Error in data.frame(x, y) :
    arguments imply differing number of rows: 4, 3
```

如果一个因子包含在一个数据框中，它必须和其中的向量有相同的长度。列名也是可以改变的，例如，data.frame(A1=x, A2=n)。用户也

可以使用row.names给行命名，但是，这个命名向量必须是字符型的而且长度等于这个数据框的行数。最后，注意数据框和矩阵一样有维数这个属性。

列表（List） 列表可以用list函数创建，方法与创建数据框类似。它对其中包含的对象没有什么限制。和data.frame()比较，缺省值没有给出对象的名称；用前面的向量x和y举例：

```
> L1 <- list(x, y); L2 <- list(A=x, B=y)
> L1
[[1]]
[1] 1 2 3 4

[[2]]
[1] 2 3 4

> L2
$A
[1] 1 2 3 4

$B
[1] 2 3 4

> names(L1)
NULL
> names(L2)
[1] "A" "B"
```

时间序列（Time-series） 函数ts可以由向量（一元时间序列）或者矩阵（多元时间序列）创建一个ts型对象，并且有一些表明序列特征的选项（带有缺省值），它们是：

```
ts(data = NA, start = 1, end = numeric(0), frequency = 1,
   deltat = 1, ts.eps = getOption("ts.eps"), class, names)
```

data	一个向量或者矩阵
start	第一个观察值的时间，为一个数字或者是一个由两个整数构成的向量（参见下面的例子）
end	最后一个观察值的时间，指定方法和start相同
frequency	单位时间内观察值的频数（频率）
deltat	两个观察值间的时间间隔（例如，月度数据的取值为1/12）；frequency和deltat必须并且只能给定其中之一
ts.eps	序列之间的误差限。如果序列之间的频率差异小于ts.eps则认为这些序列的频率相等。
class	对象的类型；一元序列的缺省值是"ts"，多元序列的缺省值是c("mts", "ts")
names	一个字符型向量，给出多元序列中每个一元序列的名称；缺省为data中每列数据的名称或者Series 1, Series 2, ...

用ts创建时间序列的一些例子：

```
> ts(1:10, start = 1959)
Time Series:
Start = 1959
End = 1968
Frequency = 1
 [1]  1  2  3  4  5  6  7  8  9 10
> ts(1:47, frequency = 12, start = c(1959, 2))
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1959       1   2   3   4   5   6   7   8   9  10  11
1960  12  13  14  15  16  17  18  19  20  21  22  23
1961  24  25  26  27  28  29  30  31  32  33  34  35
1962  36  37  38  39  40  41  42  43  44  45  46  47
> ts(1:10, frequency = 4, start = c(1959, 2))
     Qtr1 Qtr2 Qtr3 Qtr4
1959         1    2    3
1960    4    5    6    7
1961    8    9   10
> ts(matrix(rpois(36, 5), 12, 3), start=c(1961, 1), frequency=12)
         Series 1 Series 2 Series 3
Jan 1961        8        5        4
Feb 1961        6        6        9
Mar 1961        2        3        3
```

```
Apr 1961        8        5        4
May 1961        4        9        3
Jun 1961        4        6       13
Jul 1961        4        2        6
Aug 1961       11        6        4
Sep 1961        6        5        7
Oct 1961        6        5        7
Nov 1961        5        5        7
Dec 1961        8        5        2
```

表达式（Expression） 表达式类型的对象在R中有着很基础的地位，是R能够解释的字符序列。所有有效的命令都是表达式。一个命令被直接从键盘输入后，它将被R*求值*，如果是有效的则会被执行。在很多情况下，构造一个不被求值的表达式是很有用的：这就是函数expresssion要做的。当然也可以随后用eval()对创建的表达式进行求值。

```
> x <- 3; y <- 2.5; z <- 1
> exp1 <- expression(x / (y + exp(z)))
> exp1
expression(x/(y + exp(z)))
> eval(exp1)
[1] 0.5749019
```

表达式也可以在其它地方用来在图表中添加公式(p. 43)；表达式可以由字符型变量创建；一些函数把表达式当作参数，例如可以求偏导数的函数D。

```
> D(exp1, "x")
1/(y + exp(z))
> D(exp1, "y")
-x/(y + exp(z))^2
> D(exp1, "z")
-x * exp(z)/(y + exp(z))^2
```

3.5.2 对象的类型转换

读者必然会发现一些类型的对象之间的差异是很小的；因此改变一个对象的某些属性使它转换为另一种类型的对象是合乎逻辑的。as.*something*这种形式的函数可以完成转换。R（2.1.0版）的base和utils包中有98个这种函数在里面，所以我们在这里不做深入的阐述。

很明显转换取决于被转换对象的属性。一般来说，转换遵循一些很直观的规则。对于类型的不同转换，下表总结了不同的情况。

转换目标	函数	规则
数值型	as.numeric	FALSE → 0
		TRUE → 1
		"1", "2", ... → 1, 2, ...
		"A", ... → NA
逻辑型	as.logical	0 → FALSE
		其它数字 → TRUE
		"FALSE", "F" → FALSE
		"TRUE", "T" → TRUE
		其它字符 → NA
字符型	as.character	1, 2, ... → "1", "2", ...
		FALSE → "FALSE"
		TRUE → "TRUE"

有许多函数可以用来转换对象的类型(as.matrix, as.ts, as.data.frame, as.expression, ...)，这些函数在转换时会影响除了类型之外的属性，将得到的结果在一般情况下也是容易预见的。将因子转换为数值型是R中经常遇到的情况，这种情况下R将因子的水平转化为数值编码：

```
> fac <- factor(c(1, 10))
> fac
[1] 1 10
Levels:  1 10
> as.numeric(fac)
[1] 1 2
```

这对于一个字符型因子来说是很有意义的：

```
> fac2 <- factor(c("Male", "Female"))
> fac2
[1] Male   Female
Levels: Female Male
> as.numeric(fac2)
[1] 2 1
```

注意这个结果可能不像根据上面表格预期的那样是NA。

要想将一个数值型因子转换为一个数值型向量并且保持最初指定的水平值，就必须先转换成字符型然后再转换成数值型。

```
> as.numeric(as.character(fac))
[1] 1 10
```

当一个文件中的数值型变量具有非数值的值时，这种做法就很有用；前面我们已经看到了`read.table()`在缺省情况下会读取这列为一个因子。

3.5.3 运算符

我们在前面已经看到了R[10]中三种主要类型的运算符。下面是运算符的列表：

运算符					
数学运算		比较运算		逻辑运算	
+	加法	<	小于	! x	逻辑非
-	减法	>	大于	x & y	逻辑与
*	乘法	<=	小于或等于	x && y	同上
/	除法	>=	大于或等于	x | y	逻辑或
^	乘方	==	等于	x || y	同上
%%	模	!=	不等于	xor(x, y)	异或
%/%	整除				

数学运算符和比较运算符作用于两个元素上(`x + y`, `a < b`)；数学运算符不只是作用于数值型或复数型变量，也可以作用在逻辑型变量上；在后一种情况中，逻辑型变量被强制转换为数值型。比较运算符可以适用于任何类型：结果是返回一个或几个逻辑型变量。

逻辑型运算符适用于一个(对“!”运算符)或两个逻辑型对象（其它运算符），并且返回一个（或几个）逻辑性变量。运算符“逻辑与”和“逻辑或”存在两种形式：“&”和“|”作用在对象中的每一个元素上并且返回和比较次数相等长度的逻辑值；“&&”和“||”只作用在对象的第一个元素上。

对于$0 < x < 1$这种类型的不等式必须使用算子“逻辑与”，这个不等式将被改写成`0 <x & x < 1`。表达式`0 < x < 1`是合法的，但是不会返回所预期的结果：因为两个运算符相同，它们将从左至右连续执行。首先完成`0 < x`并返回一个逻辑变量，它将与1比较(`TRUE`或者`FALSE< 1`)：在这种情形下，这个逻辑变量被强制转换为数值(`1`或`0 < 1`)。

```
> x <- 0.5
> 0 < x < 1
[1] FALSE
```

[10]这些也是R中的运算符：`$`, `@`, `[`, `[[`, `:`, `?`, `<-`, `<<-`, `=`, `::`。可以用`?Syntax`命令得到描述运算符之间的优先顺序表。

比较运算符作用在两个被比较对象的每个元素上（如果需要，将循环使用最短的变量），从而返回一个同样大小的对象。为了“整体”比较两个对象，可以使用两个函数：identical和all.equal

```
> x <- 1:3; y <- 1:3
> x == y
[1] TRUE TRUE TRUE
> identical(x, y)
[1] TRUE
> all.equal(x, y)
[1] TRUE
```

identical比较数据的内在关系，如果对象是严格相同的返回TRUE，否则返回FALSE。all.equal用来判断两个对象是否“近似相等”，返回结果为TRUE或者对二者差异的描述。后一个函数在比较数值型变量时考虑到了计算过程中的近似。在计算机中数值型变量的比较有时很是令人惊奇。

```
> 0.9 == (1 - 0.1)
[1] TRUE
> identical(0.9, 1 - 0.1)
[1] TRUE
> all.equal(0.9, 1 - 0.1)
[1] TRUE
> 0.9 == (1.1 - 0.2)
[1] FALSE
> identical(0.9, 1.1 - 0.2)
[1] FALSE
> all.equal(0.9, 1.1 - 0.2)
[1] TRUE
> all.equal(0.9, 1.1 - 0.2, tolerance = 1e-16)
[1] "Mean relative  difference: 1.233581e-16"
```

3.5.4 访问一个对象的数值：下标系统

下标系统可以用来有效、灵活且有选择性地访问一个对象中的元素；下标可以是*数值型的或逻辑型的*。举例来说，要访问向量x中的第3个值，我们只要输入x[3]就可以获取或改变这个值：

```
> x <- 1:5
> x[3]
[1] 3
> x[3] <- 20
```

```
> x
[1]  1  2 20  4  5
```

下标本身也可以是一个数值型向量：

```
> i <- c(1, 3)
> x[i]
[1]  1 20
```

如果x是一个矩阵或者数据框，第i行第j列的值可以通过x[i,j]来访问。要访问一个给定的行或列中所有的值，只要忽略适当的下标（不要忘记逗号！）：

```
> x <- matrix(1:6, 2, 3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x[, 3] <- 21:22
> x
     [,1] [,2] [,3]
[1,]    1    3   21
[2,]    2    4   22
> x[, 3]
[1] 21 22
```

读者必然注意到了最后一个结果是一个向量而不是一个矩阵。R的缺省规则是返回一个维数尽可能低的对象。这可以通过修改选项drop的值来改变，它的缺省值是TRUE:

```
> x[, 3, drop = FALSE]
     [,1]
[1,]   21
[2,]   22
```

下标系统也普遍适用于数组，使用和数组维数同样多的下标（例如，一个三维数组：x[i, j, k], x[, , 3], x[, , 3, drop = FALSE]，等等）。记住下标必须使用方括号，而圆括号是用来指定函数的参数的：

```
> x(1)
Error: couldn't find function "x"
```

通过使用负数下标也可以用来不显示一个或一些行或列。比如，x[-1,]将不显示第一行，x[-c(1,15),]将不显示第1到第15行。使用上面定义的矩阵：

```
> x[, -1]
     [,1] [,2]
[1,]    3   21
[2,]    4   22
> x[, -(1:2)]
[1] 21 22
> x[, -(1:2), drop = FALSE]
     [,1]
[1,]   21
[2,]   22
```

对于向量，矩阵和数组，可以用一个比较运算表达式作为下标来访问元素的值：

```
> x <- 1:10
> x[x >= 5] <- 20
> x
 [1]  1  2  3  4 20 20 20 20 20 20
> x[x == 1] <- 25
> x
 [1] 25  2  3  4 20 20 20 20 20 20
```

下面是使用逻辑型下标的一种应用（选择一个整数变量中的偶数）：

```
> x <- rpois(40, lambda=5)
> x
 [1]  5  9  4  7  7  6  4  5 11  3  5  7  1  5  3  9  2  2  5  2
[21]  4  6  6  5  4  5  3  4  3  3  3  7  7  3  8  1  4  2  1  4
> x[x %% 2 == 0]
 [1] 4 6 4 2 2 2 4 6 6 4 4 8 4 2 4
```

在上面的例子中，下标系统使用了比较运算符返回的逻辑值。这些逻辑值可以被预先计算，如果需要，将循环使用：

```
> x <- 1:40
> s <- c(FALSE, TRUE)
> x[s]
 [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

逻辑下标也可以在数据框中使用，但是要注意数据框中不同的列可能有不同的数据类型。

对于列表，访问不同的元素（可以是任何类型的对象）可以通过单一的或者双重的方括号来实现；它们的区别是：单个括号返回一个列表，而双重括号将*提取*列表中的对象。在得到的结果中也可以使用下标，就像之前在向量、矩阵等情况中看到的那样。例如，一个列表中的第3个对象是一个向量，它的取值可以使用`my.list[[3]][i]`来访问，如果是一个三维数组则使用`my.list[[3]][i, j, k]`等等；另一个区别是`my.list[1:2]`将返回一个列表，包含原始列表的第1个和第2个元素，而`my.list[[1:2]]`不会给出期望的结果。

3.5.5 访问对象的名称

*names*是一个对象元素的字符型标签，它们一般情况下是可选的属性，名称有多个种类(*names*, *colnames*, *rownames*, *dimnames*)。

*names*是一个和对象有同样长度的向量并且可以通过函数`names`来访问。

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("a", "b", "c")
> x
a b c
1 2 3
> names(x)
[1] "a" "b" "c"
> names(x) <- NULL
> x
[1] 1 2 3
```

对于矩阵和数据框，*colnames*和*rownames*分别是列和行的标签。它们可以通过各自的函数来访问，或者通过*dimnames*返回包含两个名称向量的列表。

```
> X <- matrix(1:4, 2)
> rownames(X) <- c("a", "b")
> colnames(X) <- c("c", "d")
> X
  c d
a 1 3
b 2 4
> dimnames(X)
```

```
[[1]]
[1] "a" "b"

[[2]]
[1] "c" "d"
```

对于数组，可以用dimnames来访问各维的名字：

```
> A <- array(1:8, dim = c(2, 2, 2))
> A
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

> dimnames(A) <- list(c("a", "b"), c("c", "d"), c("e", "f"))
> A
, , e

  c d
a 1 3
b 2 4

, , f

  c d
a 5 7
b 6 8
```

如果一个对象的元素有名称，它们可以通过名称被提取，和使用下标一样。实际上，因为原始对象的属性将仍被保留，所以这里的“提取”应该叫作“取子集”更合适。例如，如果数据框DF包含变量x，y和z，命令DF["x"]将返回一个只包含x的数据框；DF[c("x", "y")]将返回包含两个变量的数据框。如果列表中的对象有名称，这种方法也将在列表中起作用。

就像读者必然认识到的那样，在这里使用的下标是一个字符型向量。就像在前面看到的数值型或逻辑型向量那样，这个向量可以被预先定义然后用来提取相应的值。

要从一个数据框中提取一个向量或者一个因子，可以使用运算符$(例如DF$x)，这对列表同样有效。

3.5.6 数据编辑器

我们可以使用一个类似于电子表格的图形编辑器去编辑一个“数据”对象。例如，如果X是一个矩阵，命令data.entry(X)将打开一个图形编辑器并且可以通过点击适当的单元格修改数值或者添加新的行或列。

函数data.entry可以直接修改作为其参数给出的对象，而不需要对其结果赋值。另一方面，函数de可以返回一个列表，其中包含作为函数参数给出的原始对象（可能已被修改过）。在缺省情况下这个结果被输出在屏幕上，但是，像对大多数函数一样，它也可以被赋值给一个对象。

使用数据编辑器的具体情况取决于操作系统。

3.5.7 数学运算和一些简单的函数

在R中有很多用来处理数据的函数，前面我们已经看到了最简单的一个函数c，它用来连接列在圆括号中的对象。如：

```
> c(1:5, seq(10, 11, 0.2))
 [1]  1.0  2.0  3.0  4.0  5.0 10.0 10.2 10.4 10.6 10.8 11.0
```

向量可以进行那些常规的算术运算：

```
> x <- 1:4
> y <- rep(1, 4)
> z <- x + y
> z
[1] 2 3 4 5
```

不同长度的向量可以相加，这种情况下最短的向量将被循环使用。例如：

```
> x <- 1:4
> y <- 1:2
> z <- x + y
> z
[1] 2 4 4 6
> x <- 1:3
> y <- 1:2
```

```
> z <- x + y
Warning message: longer object length
 is not a multiple of shorter object length in: x + y
> z
[1] 2 4 4
```

注意R返回了一个警告消息而不是一个错误消息，因此这个操作实际上是被执行了的。如果我们想要对于一个向量中所有的元素加（或乘）相同的数值：

```
> x <- 1:4
> a <- 10
> z <- a * x
> z
[1] 10 20 30 40
```

R中用来处理数据的函数太多了而不能全部列在这里。读者可以找到所有的基本数学函数(log, exp, log10, log2, sin, cos, tan, asin, acos, atan, abs, sqrt, ...), 专业函数(gamma, digamma, beta, besselI, ...)，同样包括各种统计学中有用的函数。下表中列出了一部分函数：

sum(x)	对x中的元素求和
prod(x)	对x中的元素求连乘积
max(x)	x中元素的最大值
min(x)	x中元素的最小值
which.max(x)	返回x中最大元素的下标
which.min(x)	返回x中最小元素的下标
range(x)	与c(min(x), max(x))作用相同
length(x)	x中元素的数目
mean(x)	x中元素的均值
median(x)	x中元素的中位数
var(x) or cov(x)	x中元素的的方差（用$n-1$做分母）；如果x是一个矩阵或者一个数据框，将计算协方差阵
cor(x)	如果x是一个矩阵或者一个数据框则计算相关系数矩阵（如果x是一个向量则结果是1）
var(x, y) or cov(x, y)	x和y的协方差，如果是矩阵或数据框则计算x和y对应列的协方差
cor(x, y)	x和y的线性相关系数，如果是矩阵或者数据框则计算相关系数矩阵。

这些函数一般返回标量(即为长度为1的向量)，只有range返回一个长度

为2的向量，var, cov, 和cor可能会返回一个矩阵。下面的函数返回更复杂的结果。

round(x, n)	将x中的元素四舍五入到小数点后n位
rev(x)	对x中的元素取逆序
sort(x)	将x中的元素按升序排列；要按降序排列可以用命令rev(sort(x))
rank(x)	返回x中元素的秩
log(x, base)	计算以base为底的x的对数值
scale(x)	如果x是一个矩阵，则中心化和标准化数据；若只进行中心化则使用选项scale=FALSE，只进行标准化则center=FALSE（缺省值是center=TRUE, scale=TRUE）
pmin(x,y,...)	返回一个向量，它的第i个元素是x[i], y[i], ...中最小值
pmax(x,y,...)	同上，取最大值
cumsum(x)	返回一个向量，它的第i个元素是从x[1]到x[i]的和
cumprod(x)	同上，取乘积
cummin(x)	同上，取最小值
cummax(x)	同上，取最大值
match(x, y)	返回一个和x的长度相同的向量，表示x中与y中元素相同的元素在y中的位置（没有则返回NA）
which(x == a)	返回一个包含x符合条件（当比较运算结果为真（TRUE）的下标的向量，在这个结果向量中数值i说明x[i] == a（这个函数的参数必须是逻辑型变量）
choose(n, k)	计算从n个样本中选取k个的组合数
na.omit(x)	忽略有缺失值（NA）的观察数据（如果x是矩阵或数据框则忽略相应的行）
na.fail(x)	如果x包含至少一个NA则返回一个错误消息
unique(x)	如果x是一个向量或者数据框，则返回一个类似的对象但是去掉所有重复的元素（对于重复的元素只取一个）
table(x)	返回一个表格，给出x中重复元素的个数列表(尤其对于整数型或者因子型变量)
table(x, y)	x与y的列联表
subset(x, ...)	返回x中的一个满足特定条件...的子集，该条件通常是进行比较运算：x$V1 < 10；如果x是数据框，选项select给出要保留的变量（或者用负号表示去掉）
sample(x, size)	从x中无放回抽取size个样本，选项replace = TRUE表示有放回的抽样

3.5.8 矩阵计算

R中有矩阵计算和处理的工具，函数rbind()和cbind()分别用上下或左右的方式合并向量或矩阵：

```
> m1 <- matrix(1, nr = 2, nc = 2)
```

```
> m2 <- matrix(2, nr = 2, nc = 2)
> rbind(m1, m2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    2    2
[4,]    2    2
> cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    1    1    2    2
[2,]    1    1    2    2
```

两矩阵乘积的运算符是“%*%”，例如，对上面的矩阵m1和m2：

```
> rbind(m1, m2) %*% cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    2    2    4    4
[2,]    2    2    4    4
[3,]    4    4    8    8
[4,]    4    4    8    8
> cbind(m1, m2) %*% rbind(m1, m2)
     [,1] [,2]
[1,]   10   10
[2,]   10   10
```

矩阵的转置由函数t完成；这个函数也可以作用于数据框。

函数diag可以用来提取或修正一个矩阵的对角元，或者创建一个对角矩阵。

```
> diag(m1)
[1] 1 1
> diag(rbind(m1, m2) %*% cbind(m1, m2))
[1] 2 2 8 8
> diag(m1) <- 10
> m1
     [,1] [,2]
[1,]   10    1
[2,]    1   10
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
```

```
[2,]    0    1    0
[3,]    0    0    1
> v <- c(10, 20, 30)
> diag(v)
     [,1] [,2] [,3]
[1,]   10    0    0
[2,]    0   20    0
[3,]    0    0   30
> diag(2.1, nr = 3, nc = 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.1  0.0  0.0    0    0
[2,]  0.0  2.1  0.0    0    0
[3,]  0.0  0.0  2.1    0    0
```

R中也有一些用于矩阵计算的专门的函数。我们可以使用`solve`求矩阵的逆，用`qr`来分解矩阵，用`eigen`来计算特征值和特征向量，用`svd`来做奇异值分解。

4 R绘图

R提供非常多样的绘图功能。如想了解，可以输入:`demo (graphics)`或者`demo(persp)`。我们在这里不可能详细说明R在绘图方面的所有功能，主要是因为每个绘图函数都有大量的选项使得图形的绘制十分的灵活多变。

绘图函数的工作方式与本文前面描述的工作方式大为不同，不能把绘图函数的结果赋给一个对象[11]，其结果直接输出到一个“绘图设备”上。绘图设备是一个绘图的窗口或是一个文件。

有两种绘图函数：高级绘图函数（*high-level plotting functions*）创建一个新的图形，低级绘图函数（*low-level plotting functions*）在现存的图形上添加元素。绘图参数（*graphical parameters*）控制绘图选项，可以使用缺省值或者用函数`par`修改。

我们先看如何管理绘图和绘图设备，然后详细说明绘图函数和参数，我们会看到一个用这些功能产生图形的实例，最后介绍grid和lattice两个包，这两个包的作法与其他绘图函数不同。

4.1 管理绘图

4.1.1 打开多个绘图设备

当绘图函数开始执行，如果没有打开绘图设备，那么R将打开一个绘图窗口来展示这个图形。绘图设备可以用适当的函数打开。可用的绘图设备种类取决于操作系统，在Unix/Linux 下，绘图窗口称为`x11`，而在Windows下称为`windows`。在所有情况下，都可以用命令`x11()`来打开一个绘图窗口，在Windows下仍然有效是因为上面的命令可以作为`windows()`的别名。可以用函数打开一个文件作为绘图设备，这包括：`postscript()`, `pdf()`, `png()`, ...可用的绘图设备列表可以用`?device`来察看。

最后打开的设备将成为当前的绘图设备，随后的所有图形都在这上面显示。函数`dev.list()` 显示打开的列表。

```
> x11(); x11(); pdf()
> dev.list()
X11 X11 pdf
  2   3   4
```

显示的数字是设备的编号，要改变当前设备必须使用这些编号，为了解当前设备用:

[11]有一些值得注意的例外: `hist()`和`barplot()`仍然生成数据结果作为列表或是矩阵。

```
> dev.cur()
pdf
  4
```

为改变当前的设备：

```
> dev.set(3)
X11
  3
```

函数dev.off()关闭一个设备：默认关闭当前设备，否则关闭有自变量指定编号的设备。R然后显示新的当前设备编号。

```
> dev.off(2)
X11
  3
> dev.off()
pdf
  4
```

在R的Widows版本中，有两个特殊的功能值得提及：Windows Metafile设备可以用函数win.metafile来打开，选定绘图窗口会出现“History”菜单，我们可以利用这个菜单中的功能记录一个会话中所作的所有图形（在缺省状态下，记录系统是关闭的，用户可以点击这个菜单下的“Recording”打开它）。

4.1.2 图形的分割

函数split.screen分割当前的绘图设备，例如：

```
> split.screen(c(1, 2))
```

划分设备为两部分，可以用screen(1)或者screen(2)选择；erase.screen()删除最后绘制的图形。设备的一部分也可以被split.screen() 划分，可以作出复杂的布局。

这些函数和其他的函数是不兼容的（比如layout()或者coplot()），不可以用于多个绘图设备。它们的使用应局限于象图形式探索性数据分析这样的问题。

函数layout把当前的图形窗口分割为多个部份，图形将一次显示在各部分中。它主要的自变量是一个元素都是整数值的矩阵，元素指示子窗口（“sub-windows”）的编号。例如，把设备划分为4个相等的部分：

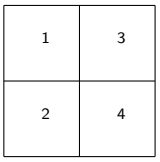
```
> layout(matrix(1:4, 2, 2))
```

当然也可以先产生这个矩阵，以更好的显现设备是如何划分的：

```
> mat <- matrix(1:4, 2, 2)
> mat
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> layout(mat)
```

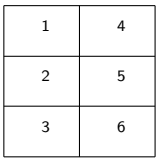
为了看到创建的分割，我们可以使用函数layout.show，其自变量是子窗口的个数（这里是4）。在这个例子中，我们有：

```
> layout.show(4)
```

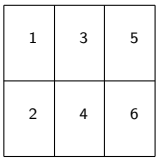


在下面的例子里，我们将看到layout()提供的各种可能性：

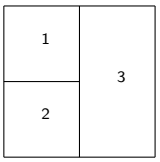
```
> layout(matrix(1:6, 3, 2))
> layout.show(6)
```




```
> layout(matrix(1:6, 2, 3))
> layout.show(6)
```



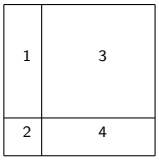

```
> m <- matrix(c(1:3, 3), 2, 2)
> layout(m)
> layout.show(3)
```



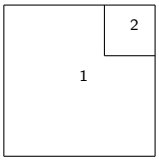
在以上各个例子中，我们没有用matrix()的选项byrow，子窗口按列编号；我们可以指定matrix(..., byrow=TRUE)，则窗口将按行编号。在矩阵中的编号可以用任何次序，例如matrix(c(2, 1, 4, 3), 2,2)。

缺省情况下，layout()用等间隔分配子窗口：可以用选项widths 和heights修改分割的宽和高。这些尺寸是相对给定的(也可以用厘米，详见?layout)，例如：

```
> m <- matrix(1:4, 2, 2)
> layout(m, widths=c(1, 3),
          heights=c(3, 1))
> layout.show(4)
```

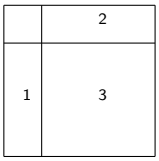



```
> m <- matrix(c(1,1,2,1),2,2)
> layout(m, widths=c(2, 1),
          heights=c(1, 2))
> layout.show(2)
```

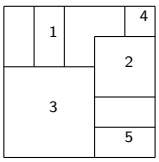


最后，矩阵里面的编号可以包括0，使得复杂的（甚至怪异的）分割成为可能。

```
> m <- matrix(0:3, 2, 2)
> layout(m, c(1, 3), c(1, 3))
> layout.show(3)
```




```
> m <- matrix(scan(), 5, 5)
1:  0 0 3 3 3 1 1 3 3 3
11:  0 0 3 3 3 0 2 2 0 5
21:  4 2 2 0 5
26:
Read 25 items
> layout(m)
> layout.show(5)
```



4.2 绘图函数

下面是R中高级绘图函数的概括:

plot(x)	以x的元素值为纵坐标、以序号为横坐标绘图
plot(x, y)	x(在x-轴上)与y(在y-轴上)的二元作图
sunflowerplot(x, y)	同上 但是以相似坐标的点作为花朵，其花瓣数目为点的个数
pie(x)	饼图
boxplot(x)	盒形图(“box-and-whiskers”)
stripchart(x)	把x的值画在一条线段上，样本量较小时可作为盒形图的替代
coplot(x~y | z)	关于z的每个数值（或数值区间）绘制x与y的二元图
interaction.plot (f1, f2, y)	如果f1和f2是因子，作y的均值图，以f1的不同值作为x轴，而f2的不同值对应不同曲线；可以用选项fun指定y的其他的统计量(缺省计算均值，fun=mean)
matplot(x,y)	二元图，其中x的第一列对应y的第一列，x的第二列对应y的第二列，依次类推。
dotchart(x)	如果x是数据框，作Cleveland点图（逐行逐列累加图）
fourfoldplot(x)	用四个四分之一圆显示2X2列联表情况（x必须是dim=c(2, 2, k)的数组，或者是dim=c(2, 2)的矩阵，如果$k = 1$）
assocplot(x)	Cohen–Friendly图，显示在二维列联表中行、列变量偏离独立性的程度
mosaicplot(x)	列联表的对数线性回归残差的马赛克图
pairs(x)	如果x是矩阵或是数据框，作x的各列之间的二元图
plot.ts(x)	如果x是类"ts"的对象，作x的时间序列曲线，x可以是多元的，但是序列必须有相同的频率和时间
ts.plot(x)	同上，但如果x是多元的，序列可有不同的时间但须有相同的频率
hist(x)	x的频率直方图
barplot(x)	x的值的条形图
qqnorm(x)	正态分位数－分位数图
qqplot(x, y)	y对x的分位数－分位数图
contour(x, y, z)	等高线图（画曲线时用内插补充空白的值），x和y必须为向量，z必须为矩阵，使得dim(z)=c(length(x), length(y))（x和y可以省略）
filled.contour (x, y, z)	同上，等高线之间的区域是彩色的，并且绘制彩色对应的值的图例
image(x, y, z)	同上，但是实际数据大小用不同色彩表示
persp(x, y, z)	同上，但为透视图
stars(x)	如果x是矩阵或者数据框，用星形和线段画出
symbols(x, y, ...)	在由x和y给定坐标画符号（圆，正方形，长方形，星，温度计式或者盒形图），符号的类型、大小、颜色等由另外的变量指定
termplot(mod.obj)	回归模型（mod.obj）的（偏）影响图

每一个函数，在R里都可以在线查询其选项。某些绘图函数的部分选项是一样的；下面列出一些主要的共同选项及其缺省值：

add=FALSE	如果是TRUE，叠加图形到前一个图上（如果有的话）
axes=TRUE	如果是FALSE，不绘制轴与边框
type="p"	指定图形的类型，"p": 点，"l": 线，"b": 点连线，"o": 同上，但是线在点上，"h": 垂直线，"s": 阶梯式，垂直线顶端显示数据，"S": 同上，但是在垂直线底端显示数据
xlim=, ylim=	指定轴的上下限，例如xlim=c(1, 10)或者xlim=range(x)
xlab=, ylab=	坐标轴的标签，必须是字符型值
main=	主标题，必须是字符型值
sub=	副标题（用小字体）

4.3 低级绘图命令

R里面有一套绘图函数是作用于现存的图形上的：称为*低级作图命令(low-level plotting commands)*。下面有一些主要的:

points(x, y)	添加点（可以使用选项type=）
lines(x, y)	同上，但是添加线
text(x, y, labels, ...)	在(x,y)处添加用labels指定的文字；典型的用法是: plot(x, y, type="n"); text(x, y, names)
mtext(text, side=3, line=0, ...)	在边空添加用text指定的文字，用side指定添加到哪一边（参照下面的axis()）；line指定添加的文字距离绘图区域的行数
segments(x0, y0, x1, y1)	从(x0,y0)各点到(x1,y1)各点画线段
arrows(x0, y0, x1, y1, angle= 30, code=2)	同上但加画箭头，如果code=2则在各(x0,y0)处画箭头，如果code=1则在各(x1,y1)处画箭头，如果code=3则在两端都画箭头; angle控制箭头轴到箭头边的角度
abline(a,b)	绘制斜率为b和截距为a的直线
abline(h=y)	在纵坐标y处画水平线
abline(v=x)	在横坐标x处画垂直线
abline(lm.obj)	画由lm.obj确定的回归线（参照第五章）
rect(x1, y1, x2, y2)	绘制长方形，(x1, y1)为左下角，(x2,y2)为右上角
polygon(x, y)	绘制连接各x,y坐标确定的点的多边形
legend(x, y, legend)	在点(x,y)处添加图例，说明内容由legend给定
title()	添加标题，也可添加一个副标题

axis(side, vect)	画坐标轴，side=1时画在下边，side=2时画在左边，side=3时画在上边，side=4时画在右边。可选参数at指定画刻度线的位置坐标
box()	在当前的图上加上边框
rug(x)	在x-轴上用短线画出x数据的位置
locator(n, type="n", ...)	在用户用鼠标在图上点击n次后返回n次点击的坐标(x, y)；并可以在点击处绘制符号(type="p"时)或连线(type="l"时)，缺省情况下不画符号或连线

注意，用text(x, y,expression(...))可以在一个图形上加上数学公式，函数expression把自变量转换为数学公式。例如，

```
> text(x, y, expression(p == over(1, 1+e^-(beta*x+alpha))))
```

在图中相应坐标点(x, y)处显示下面的方程:

$$\mathrm{p} = \frac{1}{1 + \mathrm{e}^{-(\beta \mathrm{x} + \alpha)}}$$

为了能在表达式中代入某个变量的值，我们可以使用函数substitute和as.expression，例如，为了代入R^2的值（之前计算并储存在对象Rsquared中）

```
> text(x, y, as.expression(substitute(R^2==r, list(r=Rsquared))))
```

在图中相应坐标点(x, y)处显示：

$$\mathrm{R}^2 = 0.9856298$$

如果只显示3位小数，我们可以修改代码如下：

```
> text(x, y, as.expression(substitute(R^2==r,
+                           list(r=round(Rsquared, 3)))))
```

将显示：

$$\mathrm{R}^2 = 0.986$$

最后，用斜体字显示R：

```
> text(x, y, as.expression(substitute(italic(R)^2==r,
+                           list(r=round(Rsquared, 3)))))
```

$$R^2 = 0.986$$

4.4 绘图参数

除了低级作图命令之外，图形的显示也可以用绘图参数来改良。绘图参数可以作为图形函数的选项（但不是所有参数都可以这样用），也可以用函数par来永久地改变绘图参数，也就是说后来的图形都将按照par指定的参数来绘制。例如，下面的命令：

```
> par(bg="yellow")
```

将导致后来的图形都以黄色的背景来绘制。有73个绘图参数，其中一些有非常相似的功能。这些参数详细的列表可以参阅?par；下面的表格只列举了最常用的参数。

adj	控制关于文字的对齐方式，0是左对齐，0.5是居中对齐，1是右对齐，值> 1时对齐位置在文本右边的地方，取负值时对齐位置在文本左边的地方；如果给出两个值（例如c(0, 0)），第二个只控制关于文字基线的垂直调整
bg	指定背景色（例如bg="red", bg="blue"; 用colors()可以显示657种可用的颜色名）
bty	控制图形边框形状，可用的值为: "o", "l", "7", "c", "u" 和"]" （边框和字符的外表相像）；如果bty="n"则不绘制边框
cex	控制缺省状态下符号和文字大小的值；另外，cex.axis控制坐标轴刻度数字大小，cex.lab控制坐标轴标签文字大小，cex.main控制标题文字大小，cex.sub控制副标题文字大小
col	控制符号的颜色；和cex类似，还可用：col.axis, col.lab, col.main, col.sub
font	控制文字字体的整数（1: 正常，2: 斜体，3: 粗体，4: 粗斜体）；和cex类似，还可用: font.axis, font.lab, font.main, font.sub
las	控制坐标轴刻度数字标记方向的整数（0: 平行于轴，1: 横排，2: 垂直于轴，3: 竖排）
lty	控制连线的线型，可以是整数（1: 实线，2: 虚线，3: 点线，4: 点虚线，5: 长虚线，6: 双虚线），或者是不超过8个字符的字符串（字符为从"0"到"9"之间的数字）交替地指定线和空白的长度，单位为磅(points)或象素，例如lty="44"和lty=2效果相同
lwd	控制连线宽度的数字
mar	控制图形边空的有4个值的向量c(bottom, left, top, right)，缺省值为c(5.1, 4.1, 4.1, 2.1)
mfcol	c(nr,nc)的向量，分割绘图窗口为nr行nc列的矩阵布局，按列次序使用各子窗口（参照 4.1.2）
mfrow	同上，但是按行次序使用各子窗口（参照 4.1.2）
pch	控制符号的类型，可以是1到25的整数，也可以是""里的单个字符（图 2）
ps	控制文字大小的整数，单位为磅(points)
pty	指定绘图区域类型的字符，"s": 正方形，"m":最大利用
tck	指定轴上刻度长度的值，单位是百分比，以图形宽、高中最小一个作为基数；如果tck=1则绘制grid
tcl	同上，但以文本行高度为基数（缺省下tcl=-0.5）
xaxt	如果xaxt="n"则设置x-轴但不显示（有助于和axis(side=1, ...)联合使用）
yaxt	如果yaxt="n"则设置y-轴但不显示（有助于和axis(side=2, ...)联合使用）

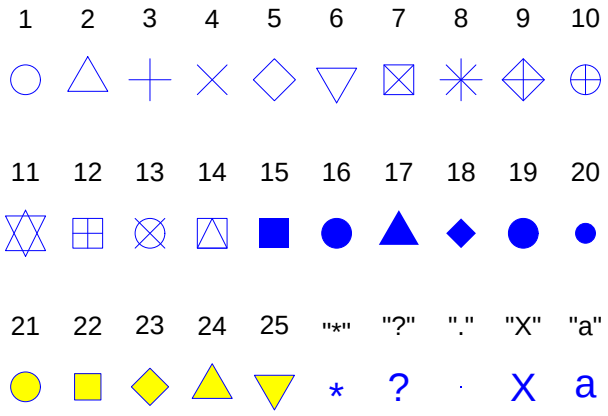


图 2: R (pch=1:25)的绘图符号。用选项col="blue", bg="yellow"来产生如上的颜色，其中背景色选项只对符号21–25有作用。可以使用任意字符作为绘点符号（pch="*", "?", ".", ...）。

4.5 一个实例

为了讲解R的绘图功能，让我们来看一个简单的10对随机值的二维图形的例子。这些值用以下命令生成：

```
> x <- rnorm(10)
> y <- rnorm(10)
```

所需的图可以用plot()来产生；只要输入命令：

```
> plot(x, y)
```

则图形将绘制在当前的绘图设备上。结果见图 3。缺省情况下，R用“智能”的方法绘制图形：R自动计算坐标轴上刻度摆放，标记的位置等，使得图形尽可能的易于理解。

用户仍然可以改变绘图的方法，例如，为了遵照某刊物的要求，或为某个演讲作个性化调整。最简单的方式就是用选项值取代缺省值来修改图形绘制方式。在我们的例子中，我们可以用以下方式大大改变图形：

```
plot(x, y, xlab="Ten random values", ylab="Ten other values",
     xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red",
     bg="yellow", bty="l", tcl=0.4,
     main="How to customize a plot with R", las=1, cex=1.5)
```

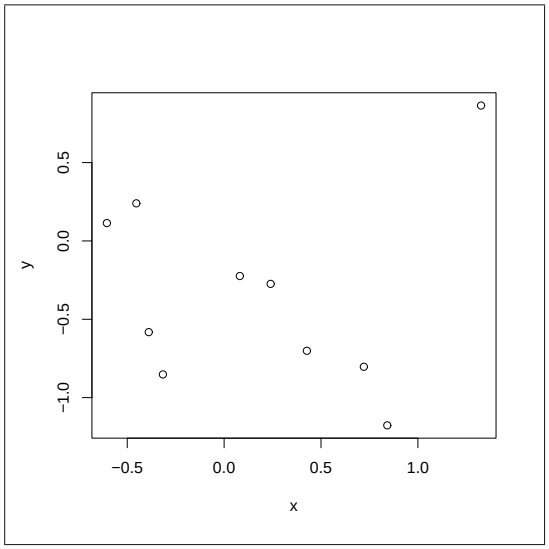


图 3: 没有用任何选项的函数plot。

结果见图 4。我们来详细说明其中的每个选项。首先，xlab 和ylab改变坐标轴标签，缺省情况下是变量的名字。然后，xlim 和ylim允许我们规定两个坐标轴的范围[12]。绘图参数pch在这里用作一个选项：pch=22为正方形，其轮廓颜色和背景色可能不一样，分别由col 和bg指定。图形参数表中说明了bty,tcl,las和cex的作用最后，选项main添加了标题。

绘图参数和低级作图函数使我们可以进一步改善图形。前面我们已经看到，一些绘图参数不允许作为plot这样的函数的自变量。我们可以用par()来修改这些参数，这样就必须输入多行的命令。在改变绘图参数时，预先保存它们的初始值以便以后恢复十分有用。以下命令可以产生图 5。

```
opar <- par()
par(bg="lightyellow", col.axis="blue", mar=c(4, 4, 2.5, 0.25))
plot(x, y, xlab="Ten random values", ylab="Ten other values",
     xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red", bg="yellow",
     bty="l", tcl=-.25, las=1, cex=1.5)
title("How to customize a plot with R (bis)", font.main=3, adj=1)
par(opar)
```

我们详细解释这些命令的作用。首先，缺省的绘图参数被复制到列表opar中。然后有三个参数被修改：bg修改背景色，col.axis修改轴上数字的颜色，mar来修改绘图边空大小。这个图形用几乎和图 4相似的方式画出。边空的修改让我们更好地利用绘图区周围的空白。用低级函数title添加标

[12]缺省下，R在每个坐标轴界限上增加4%。这种习惯可以随着设置绘图参数xaxs="i"和yaxs="i"而改变（他们可以作为plot()的选项被传递）。

图 4: 用了选项的函数`plot`。

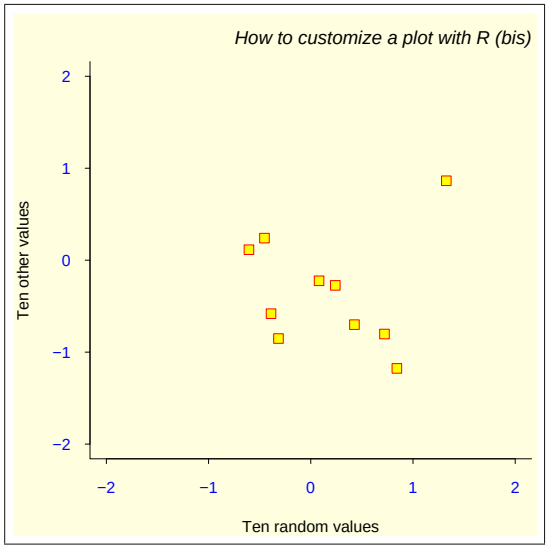


图 5: 函数`par`，`plot`和`title`。

题，这样允许给定一些参数作为自变量而不改变图形的其余部分。最后一行的命令恢复初始的绘图参数。

现在，完全控制！在图 5中，R仍然自动决定了诸如坐标轴刻度的个数，标题与绘图区域之间的距离等少许事情。我们现在将看到如何完全控制图形的绘制。这里用的方法是用plot(...,type="n")绘制一个“空白”的图形，然后用低级函数来添加点，坐标轴，标签等。我们可以想出诸如改变绘图区域颜色这样的安排。命令如下，产生的图形见图 6。

```
opar <- par()
par(bg="lightgray", mar=c(2.5, 1.5, 2.5, 0.25))
plot(x, y, type="n", xlab="", ylab="", xlim=c(-2, 2),
     ylim=c(-2, 2), xaxt="n", yaxt="n")
rect(-3, -3, 3, 3, col="cornsilk")
points(x, y, pch=10, col="red", cex=2)
axis(side=1, c(-2, 0, 2), tcl=-0.2, labels=FALSE)
axis(side=2, -1:1, tcl=-0.2, labels=FALSE)
title("How to customize a plot with R (ter)",
      font.main=4, adj=1, cex.main=1)
mtext("Ten random values", side=1, line=1, at=1, cex=0.9, font=3)
mtext("Ten other values", line=0.5, at=-1.8, cex=0.9, font=3)
mtext(c(-2, 0, 2), side=1, las=1, at=c(-2, 0, 2), line=0.3,
      col="blue", cex=0.9)
mtext(-1:1, side=2, las=1, at=-1:1, line=0.2, col="blue", cex=0.9)
par(opar)
```

和以前一样，先保存缺省的绘图参数，然后修改背景颜色和边空。画图时用type="n"不画出点，用xlab="",ylab=""不画坐标轴标签，和用xaxt="n",yaxt="n"不画坐标轴。这样只画了绘图区域的边框，并用xlim和ylim规定了坐标轴范围。注意，我们可以用选项axes=FALSE，但这样的话不仅不画坐标轴，而且也不画边框。

然后，用低级图形函数在上面确定的坐标区域内加入各种图形元素。在添加点以前，用rect()修改绘图区域的颜色：长方形大小选得比绘图区域大得多。

用points()画点；用了一个新的符号。用axis()添加坐标轴：第二个自变量提供的向量指定坐标刻度位置。选项labels=FALSE指定画坐标轴时不画刻度数字。这个选项也可以用于字符式样的向量，例如labels=c("A", "B", "C")。

用title()添加标题，但是字体稍微改变了。开始的两个边空文字函数mtext()调用画坐标轴的标签。这个函数的第一自变量是要画的文本。选项line指出到绘图区域的距离行数（缺省时line=0），at给出坐标。第二次

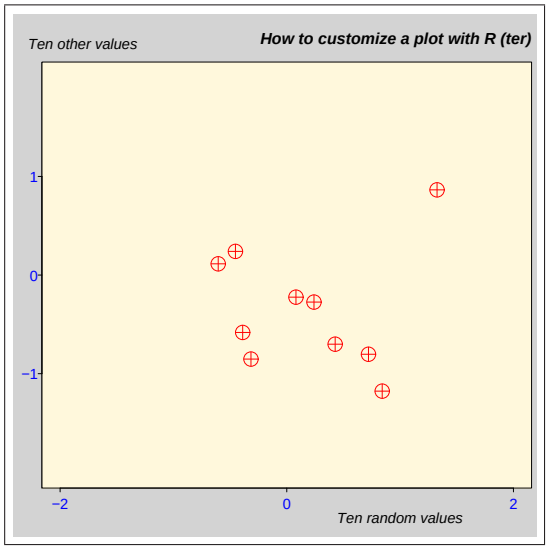


图 6: “手工”图。

调用`mtext()`,调用利用了`side` (3)的缺省值。另外两个`mtext()`用数值型向量作第一自变量，会自动转换为字符型。

4.6 grid 和lattice 包

grid 和lattice包实现grid和lattice系统。grid是一个新的绘图模式，其绘图参数的系统和上面所讲的截然不同。grid与基本的图形比较，主要的不同之处有以下两点：

- 用*viewports*可以以更灵活的方式来分割绘图设备，使其全部覆盖（绘图对象甚至可以在不同的视口中共享，例如，箭头记号）；
- 绘图对象(*grob*)可以被修改或者从一个图形中移除，并不需要重新绘制所有的图形（用基本图形时必须重绘）。

grid图形通常不可以用来和基本图形组合或者混合（必须用gridBase包来作）。可是，在相同的绘图设备上，同一会话里用两种绘图模式是有可能的。

Lattice本来是S-PLUS里的Trellis图形在R中的实现。Trellis是多元数据可视化的方法，特别适用于发现各变量之间的相互作用关系[13]。Lattice（Trellis）的主要想法是不同条件下的多个图：根据某变量的值的不同对两个变量作不同图。函数`coplot`作用类似，但是lattice提供了更广泛的功能。Lattice用grid图形模式。

[13] http://cm.bell-labs.com/cm/ms /departments/sia/project/trellis/index.html

在lattice里，大部分的函数都把一个公式作为他们的主要自变量[14]，例如y ~ x。公式y ~ x | z是指按z值的不同关于y对x作不同图形。

下面的表格给出了lattice中的主要函数。表中列出的作为自变量的公式是典型的用法，但是，所有的这些函数接受条件公式(y ~ x | z)作为主要变量；在后一种情况下，对z的不同取值将作多个图形。

barchart(y ~ x)	y对x的直方图
bwplot(y ~ x)	盒形图
densityplot(~ x)	密度函数图
dotplot(y ~ x)	Cleveland点图(逐行逐列累加图)
histogram(~ x)	x的频率直方图
qqmath(~ x)	x的关于某理论分布的分位数-分位数图
stripplot(y ~ x)	一维图，x必须是数值型，y可以是因子
qq(y ~ x)	比较两个分布的分位数，x必须是数值型，y可以是数值型，字符型，或者因子，但是必须有两个"水平"
xyplot(y ~ x)	二元图（有许多功能）
levelplot(z ~ x*y) contourplot(z ~ x*y)	在x，y坐标点的z值的彩色等值线图（x，y和z等长）
cloud(z ~ x*y)	3-D透视图（点）
wireframe(z ~ x*y)	同上（面）
splom(~ x)	二维图矩阵
parallel(~ x)	平行坐标图

为了举例说明lattice的一些方面，让我们来看一些例子。包必须要用命令library(lattice)来装载在内存中，使得包的函数可以被访问。

让我们从密度函数图形开始。这样的图形可以用densityplot(~ x)简单的作出，将作出经验密度函数曲线并在x-轴处用散点显示各观测值(如rug()所作)。我们的例子将稍微的变复杂一些，在每个图形里，除经验密度曲线之外还叠加一个正态密度拟合曲线。这样必须用自变量panel 来定义每个图上绘制什么。命令如下：

```
n <- seq(5, 45, 5)
x <- rnorm(sum(n))
y <- factor(rep(n, n), labels=paste("n =", n))
densityplot(~ x | y,
            panel = function(x, ...) {
                panel.densityplot(x, col="DarkOliveGreen", ...)
                panel.mathdensity(dmath=dnorm,
                                  args=list(mean=mean(x), sd=sd(x)),
                                  col="darkblue")
            })
```

[14] plot() 也接受一个公式作为其主要变量：如果x和y是两个相同长度的向量，plot(y ~ x)和plot(x,y)将绘制同样的图形。

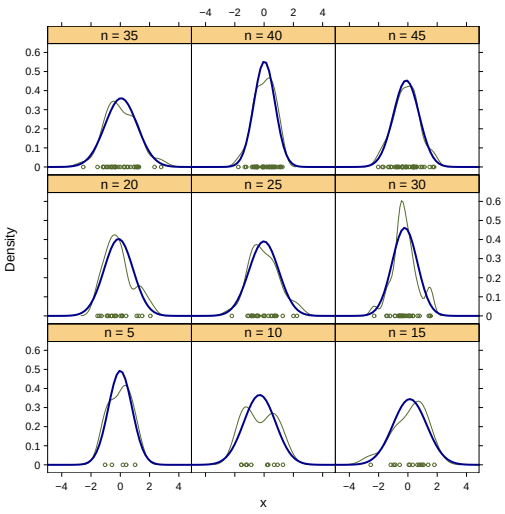


图 7: 函数densityplot。

命令的前三行产生随机独立正态样本，分割成个数等于5, 10, 15, ...，和45 的子样本。然后用densityplot为每个子样本产生图形。panel作为函数的自变量。在我们的例子中，我们定义一个函数调用lattice中的预先确定的两个函数：panel.densityplot绘制经验密度函数，panel.mathdensity绘制拟合的正态分布密度函数。函数panel.densityplot被缺省调用如果没有自变量给panel：命令densityplot (~ x | y)将有和图 7相同的结果，但是没有蓝线。

下一个例子是修改lattice包帮助理的例子得到的，用一些R里面有的数据集：斐济岛附近1000个地震位置，和一些三种鸢尾花朵的测量值。

图 8显示不同深度的地震的地理位置，作图命令如下：

```
data(quakes)
mini <- min(quakes$depth)
maxi <- max(quakes$depth)
int <- ceiling((maxi - mini)/9)
inf <- seq(mini, maxi, int)
quakes$depth.cat <- factor(floor(((quakes$depth - mini) / int)),
                      labels=paste(inf, inf + int, sep="-"))
xyplot(lat ~ long | depth.cat, data = quakes)
```

第一个命令是在内存中装载数据quakes。后面的5行命令创建了一个把深度等分为九个区间的因子：因子水平标签为区间的上下界。下面就只要用适当的公式调用xyplot函数，其中用data自变量指定绘图用的各变量所在的数

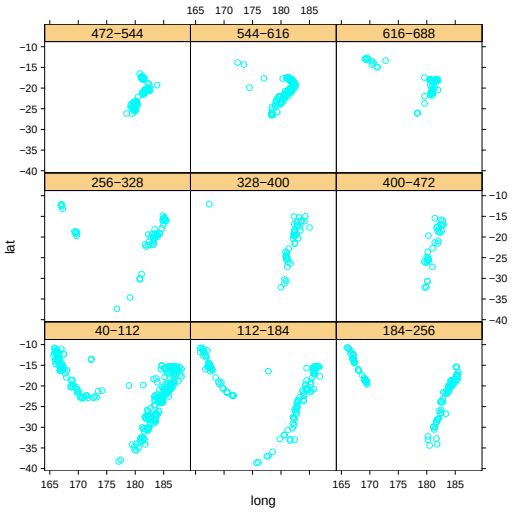


图 8: 应用数据“quakes”的函数xyplot。

据框[15]。

关于数据iris，不同物种的重叠部分十分小，使得可以画在图 9中。命令如下：

```
data(iris)
xyplot(
  Petal.Length ~ Petal.Width, data = iris, groups=Species,
  panel = panel.superpose,
  type = c("p", "smooth"), span=.75,
  auto.key = list(x = 0.15, y = 0.85)
)
```

在这里调用函数xyplot比之前的例子要复杂些，所用的一些选项，我们将详细介绍。选项groups定义组以便被其他选项引用。我们已经知道选项panel是来定义如何在图中显示不同组的：我们在这里用了预先定义的函数panel.superpose，是为了把不用的组重叠的绘制在同一个图里。没有给panel.superpose传递选项，使用缺省颜色用于区分不同的组。选项type和在plot()中一样，指定数据如何显示，但是这里我们可以给出包含若干元素的向量作为自变量："p" 表示画点，"smooth"表示画光滑曲线，其光滑程度用span指定。选项auto.key在图形中添加图例：只需要给出一个指定图例的坐标的列表。注意，这里的坐标是相对与图形大小的比例值（也就是在[0, 1]）。

[15]plot() 不可以用data作为自变量,变量的位置必须明确的给定，例如plot(quakes$long ~quakes$lat)。

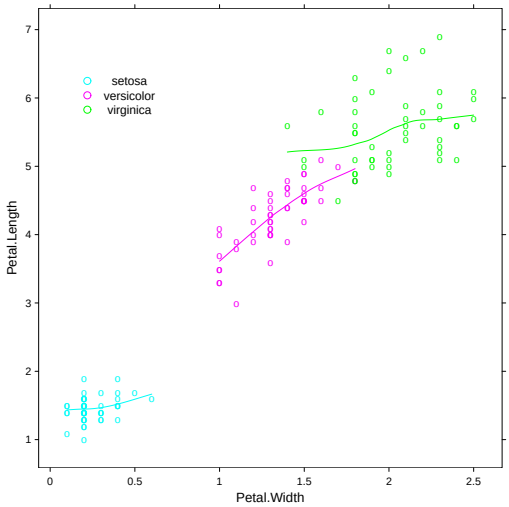


图 9: 应用数据“iris”的函数xyplot。

我们现在看到用函数splom用于相同的数据iris上。用以下命令来绘制图 10：

```
splom(
  ~iris[1:4], groups = Species, data = iris, xlab = "",
  panel = panel.superpose,
  auto.key = list(columns = 3)
)
```

这次主要的自变量是一个矩阵（iris的前四列）。结果是一系列可能的在矩阵列之中的二元图，就像标准函数pairs。缺省下，splom在x-下加上“Scatter Plot Matrix”的文字：要消除的话，可以用选项xlab=""。其他的选项和前面的例子类似，除了auto.key的columns = 3是指定图例显示在3列中。

图 10可以用pairs()做出，但是后者的函数不可以作像图Fig. 11一样的有条件的图形。使用的代码相对简单：

```
splom(~iris[1:3] | Species, data = iris, pscales = 0,
      varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"))
```

子图相对比较小，我们添加了两个选项来改进图的易理解性：pscales = 0 移除坐标轴的刻度（所有的子图用相同的刻度绘制），变量的名字被重新定义，并用两行来显示（"\n"表示换行）。

最后的例子用平行坐标来探索多元数据分析的方法。变量安排在一个坐标轴上（例如y-轴），观测值绘制在另一坐标轴上（变量取值范围调整到

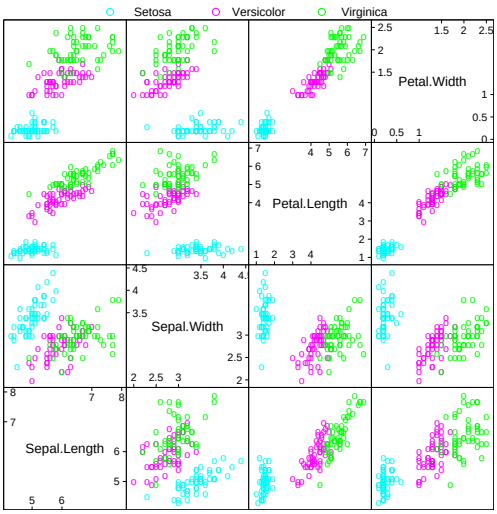


图 10: 应用数据数据“iris”（1）的函数`splom`。

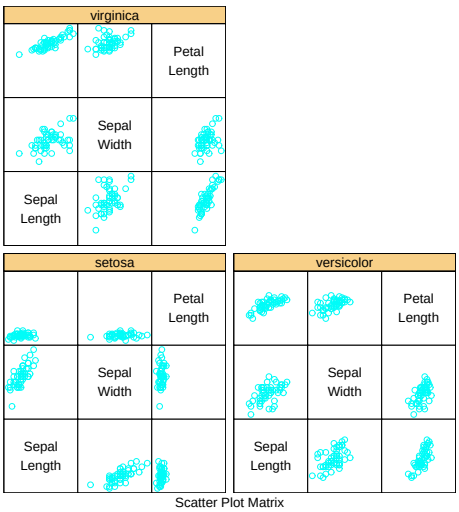


图 11: 应用数据“iris”（2）的函数`splom`。

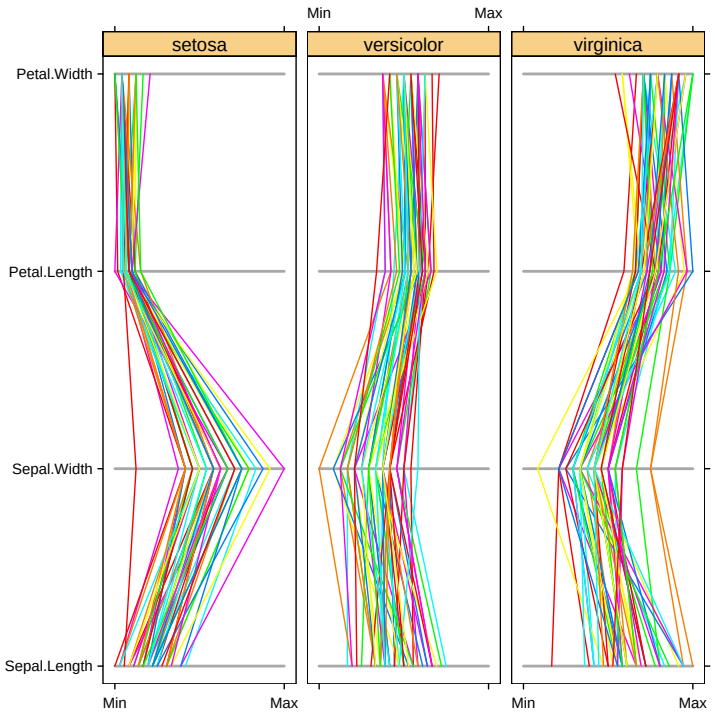


图 12: 应用数据“iris”的函数`parallel`。

可比，例如，标准化之）。相同个体不同变量的值连接在一条线上。用数据`iris`，下面的命令可以得到图 12：

```
parallel(~iris[, 1:4] | Species, data = iris, layout = c(3, 1))
```

5 R的统计分析

尽管R的统计功能比作图功能强很多，但这里不可能很详细地去描述它所有的统计分析功能. 我的目的是在这里给读者对R统计分析功能的一个粗略而又系统的介绍.

包stats 包括了一系列基本的统计分析函数：经典的假设检验, 线性模型(包括最小二乘法回归, 广义线性模型, 和方差分析), 统计分布, 汇总统计, 层次聚类[16], 时间序列分析, 非线性最小二乘法和多元分析. 其他的R包还提供了一些上述统计方法以外的的统计方法. 和基本R安装同时发布的统计包被标注为*推荐包*, 而其他的包标注为*捐献包*并且要求用户自己安装.

我们以一个简单的例子开始。这个例子仅仅需要包stats, 但它可以说明用R进行分析的大体过程。然后,我们将细致讲述两个在所有统计分析中都非常有用的概念, 公式(formulae) 和*泛型函数*(generic functions). 我们还会对所有的包进行一个总结.

5.1 关于方差分析的一个简单例子

在包stats 里面的方差分析函数是aov. 为了示范这个函数, 我们采用R分发的数据集: InsectSprays. 六种杀虫剂将会在田野中进行效价测试, 观测变量是昆虫的个数. 每种杀虫剂重复测试了12 次, 因此共有72次观测致.我们不想在这里讨论这些数据的统计图分析, 我们的焦点在于研究响应变量关于杀虫剂种类的简单方差分析. 通过函数data 把数据导入内存后, 首先对数据进行平方根转换，然后再进行分析:

```
> data(InsectSprays)
> aov.spray <- aov(sqrt(count) ~ spray, data = InsectSprays)
```

函数aov的主要参数(必要的)是一个公式. 公式的左边是响应变量，右边是预测变量，二者通过~ 连接. 可选项data = InsectSprays 表明这些变量是在数据框InsectSprays中. 下面的语句和前面的语句在语法上等价:

```
> aov.spray <- aov(sqrt(InsectSprays$count) ~ InsectSprays$spray)
```

如果我们知道列的编号, 我们还可以如下分析:

```
> aov.spray <- aov(sqrt(InsectSprays[, 1]) ~ InsectSprays[, 2])
```

[16]译者注：hierarchical clustering

当然, 第一种方式是首选的, 因为它最为简明易懂.

前面脚本运行的结果不会在屏幕上显示, 因为它们都被赋给对象aov.spray. 我们必须采用一些函数去解析这些结果, 如函数print 可以对分析结果进行一个简单的总结(一般是要估计的参数), 函数summary 可以显示更多的细节(包括统计检验的结果):

```
> aov.spray
Call:
   aov(formula = sqrt(count) ~ spray, data = InsectSprays)

Terms:
                    spray Residuals
Sum of Squares  88.43787  26.05798
Deg. of Freedom        5        66

Residual standard error: 0.6283453
Estimated effects may be unbalanced
> summary(aov.spray)
            Df Sum Sq Mean Sq F value    Pr(>F)
spray        5 88.438  17.688  44.799 < 2.2e-16 ***
Residuals   66 26.058   0.395
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

需要说明的是，直接键入对象名字和命令print(aov.spray) 的结果是类似的. 可以通过函数plot() 和termplot() 展示分析结果的统计图. 在键入函数plot(aov.spray)前, 我们必须把图形界面分成四部分, 这样就可以让四个统计诊断图在同一个图上显示. 相应的命令是[17]:

```
> opar <- par()
> par(mfcol = c(2, 2))
> plot(aov.spray)
> par(opar)
> termplot(aov.spray, se=TRUE, partial.resid=TRUE, rug=TRUE)
```

运行结果如图 13 和14 所示.

[17]译者注:我个人更喜欢用函数layout

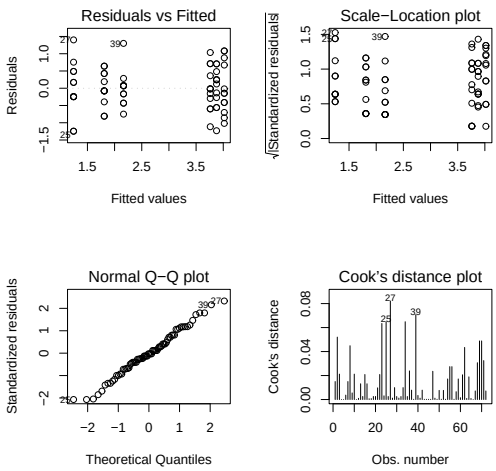


图 13: 通过函数plot() 图形化展示函数aov的分析结果.

5.2 公式

公式是R统计分析里面的关键元素: 几乎所有函数都采用一样的符号[18].公式的典型形式是y ~ *model*, 其中y是响应变量, *model* 是一些元素项的集合而且要为其中一些项估计参数. 这些元素项通过一些有特殊涵义的运算符连接.

a+b	a 和b的相加效应
X	如果X 是一个矩阵, 这将反映各列的相加效应, 即X[,1]+X[,2]+...+X[,ncol(X)]; 还可以通过索引向量选择特定列进行分析(如, X[,2:4])
a:b	a 和b 的交互效应
a*b	相加和交互效应(等价于a+b+a:b)
poly(a, n)	a的n价多项式
^n	包含所有的直到n阶的交互作用, 即 (a+b+c)^2 等价于a+b+c+a:b+a:c+b:c
b %in% a	b 和a的嵌套分类设计(等价于a+a:b, 或者a/b)
-b	去掉因子b的影响, 如: (a+b+c)^2-a:b 等价于a+b+c+a:c+b:c
-1	y~x-1 表示通过原点的线性回归(等价于 y~x+0 或者0+y~x)
1	y~1 拟合一个没有因子影响的模型(仅仅是截距)
offset(...)	向模型中增加一个影响因子但不估计任何参数(如, offset(3*x))

[18]译者注:也有些例外, 基本上趋同.

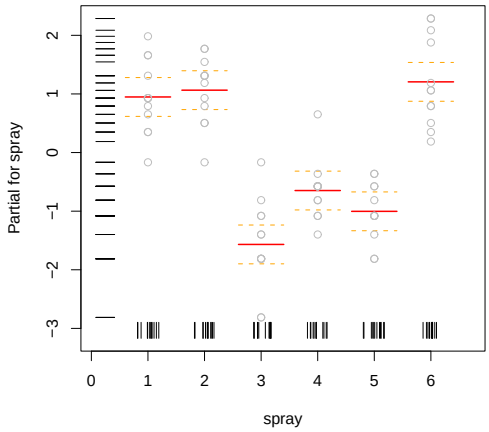


图 14: 通过函数`termplot()` 图形化展示函数`aov`的分析结果.

可以看出, 在R公式里面采用的运算符和表达式里面使用的运算符有着不同的含义.例如, 公式`y~x1+x2` 表示模型$y = \beta_1 x_1 + \beta_2 x_2 + \alpha$, 而不是(如果`+`采用它常规的含义) $y = \beta(x_1 + x_2) + \alpha$. 为了可以在公式中使用常规的运算符, 我们可以使用函数`I`: 公式`y~I(x1+x2)` 表示模型$y = \beta(x_1 + x_2) + \alpha$. 类似的, 为了定义模型$y = \beta_1 x + \beta_2 x^2 + \alpha$, 我们可以使用公式`y ~ poly(x, 2)` (而非`y ~ x + x^2`). 但是, 为了对变量进行一定的转换, 可以在公式中包含一些函数，如前面的杀虫剂效价分析的方差分析公式.

对于方差分析, `aov()` 用一个特别的语法规则定义随机效应. 例如, `y ~ a + Error(b)` 表示固定项`a`和随机项`b`的相加效应.

5.3 泛型函数

和许多统计编程语言不同的是, R函数将输入对象的属性作为输入参数. 类是最应该关注的的一个属性.R统计函数常常返回一个类名与函数名相同的对象(如, `aov` 返回类`"aov"`的对象, `lm` 返回类`"lm"`的对象). 我们用来解析结果的函数对特定的类对象有特定的行为. 这些函数被称为*泛型*(generic)[19].

例如, 最常用的解析统计分析结果的R函数是`summary`. 它可以用来显示较为细致的结果. 无论作为参数的对象可能是`"lm"` 类(线性模型) 或者`"aov"`类(方差分析), 显示的信息显然是不一样的. 泛型函数的优势在于一个函数对所有类的使用格式都是一样的[20].

[19]译者注:在Java, C++等面向对象语言中, 泛型有更为详细的介绍.这里, 我是借用了它们的概念.此外, 我觉得R里面的泛型,更像Java里面的接口.

[20]译者注:这里和Java的接口定义非常的相似.

一个包含分析结果的对象常常是一个列表对象, 而它的结果展示方式由它的类定义所决定. 前面的例子中已经体现这种思想，就是一个函数的行为由输入参数的对象类型决定[21]. 这是R [22]的一个重要性质. 下面的列表列出一些用于提取分析结果对象的信息的主要泛型函数. 这些函数的典型使用方式为:

```
> mod <- lm(y ~ x)
> df.residual(mod)
[1] 8
```

`print`	返回简单的汇总信息
`summary`	返回较为详细的汇总信息
`df.residual`	返回残差的自由度
`coef`	返回被估计的系数(有时还包括他们的标准差)
`residuals`	返回残差
`deviance`	返回方差
`fitted`	返回拟合值
`logLik`	计算对数似然值和返回参数数目
`AIC`	计算Akaike 信息准则(Akaike information criterion, AIC)(依赖于`logLik()`)

像`aov` 或者`lm`之类的函数返回一个保存各分析结果的列表. 以前面对数据集`InsectSprays`进行的方差分析为例, 我们可以看一下`aov` 返回对象的结构:

```
> str(aov.spray, max.level = -1)
List of 13
 - attr(*, "class")= chr [1:2] "aov" "lm"
```

另外一种查看对象结构的方法是显示列表对象各个元素的名字:

```
> names(aov.spray)
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "contrasts"
[10] "xlevels"       "call"          "terms"
[13] "model"
```

这些看到的元素都可以提取出来:

```
> aov.spray$coefficients
(Intercept)      sprayB      sprayC      sprayD
```

[21] 译者注:函数多态性.
[22] 在R 里面有超过100个泛型函数.

```
  3.7606784   0.1159530  -2.5158217  -1.5963245
     sprayE      sprayF
 -1.9512174   0.2579388
```

在aov()例子中的summary()函数实际上也创建了一个简单的列表对象:

```
> str(summary(aov.spray))
List of 1
 $ :Classes anova  and 'data.frame':    2 obs. of  5 variables:
  ..$ Df     : num [1:2] 5 66
  ..$ Sum Sq : num [1:2] 88.4 26.1
  ..$ Mean Sq: num [1:2] 17.688  0.395
  ..$ F value: num [1:2] 44.8   NA
  ..$ Pr(>F) : num [1:2] 0 NA
 - attr(*, "class")= chr [1:2] "summary.aov" "listof"
> names(summary(aov.spray))
NULL
```

泛型函数很少对对象进行任何操作: 它们调用自变量所属类的对应函数. 在R的术语里面, 泛型调用的函数称为*方法*(method). 简略地说, 一个方法的构建方式是*generic.cls*, 其中*cls* 是对象的类.例如,以summary 为例, 我们可以显示对应的方法:

```
> apropos("^summary")
 [1] "summary"               "summary.aov"
 [3] "summary.aovlist"       "summary.connection"
 [5] "summary.data.frame"    "summary.default"
 [7] "summary.factor"        "summary.glm"
 [9] "summary.glm.null"      "summary.infl"
[11] "summary.lm"            "summary.lm.null"
[13] "summary.manova"        "summary.matrix"
[15] "summary.mlm"           "summary.packageStatus"
[17] "summary.POSIXct"       "summary.POSIXlt"
[19] "summary.table"
```

通过下面的简单的例子, 我们可以发现这些泛型函数在线性回归和方差分析中的行为是不同的:

```
> x <- y <- rnorm(5);
> lm.spray <- lm(y ~ x)
> names(lm.spray)
 [1] "coefficients"  "residuals"     "effects"
```

```
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
> names(summary(lm.spray))
 [1] "call"          "terms"         "residuals"
 [4] "coefficients"  "sigma"         "df"
 [7] "r.squared"     "adj.r.squared" "fstatistic"
[10] "cov.unscaled"
```

下面的表格中显示了一些可以对分析结果对象做一些补充分析的泛型函数, 主要参数一般都是分析结果对象, 但是有些情况下,如泛型函数如predict 或update 需要一些额外的参数.

add1	连续测试所有可以加入模型的元素项
drop1	连续测试所有可以从模型中移除的元素项
step	通过AIC (调用add1 和drop1)选择一个模型
anova	计算一个或多个模型的方差/残差分析表
predict	通过拟合的模型计算一个新的数据集的预测值
update	用新的数据或者公式拟合一个模型

还有很多各种各样用于从模型对象或者公式中提取信息的效用函数, 如函数alias 可以用来查找一个特定公式拟合的线性模型中的线性依赖项.

最后,还有许多图形函数, 如plot 显示各种各样的诊断图, 或者termplot (见上面的例子), 尽管后面一个函数不是泛型函数但它调用泛型函数predict.

5.4 包

下表将列出随R基本安装环境发布的*标准* 包. 其中一些包在R启动时就直接调入内存; 这些可用通过函数search 显示:

```
> search()
[1] ".GlobalEnv"        "package:methods"
[3] "package:stats"     "package:graphics"
[5] "package:grDevices" "package:utils"
[7] "package:datasets"  "Autoloads"
[9] "package:base"
```

其他包需要在载入后才能使用:

```
> library(grid)
```

一个包中可以使用的函数可以通过下面的方式显示:

```
> library(help = grid)
```

或者直接访问网页格式的帮助文档. 任何函数的相关信息可以用(7)页描述的方法访问.

包	描述
base	基本R函数
datasets	基本R数据集
grDevices	基本的或grid图形的设备函数
graphics	基本图形函数
grid	grid图形
methods	用于R对象和编程工具的方法和类的定义
splines	样条回归函数和类
stats	统计函数
stats4	基于S4标准定义的统计函数
tcltk	R 和Tcl/Tk 图形接口元素的交互函数
tools	包开发和管理的工具
utils	R 工具函数

许多*捐献* 包都丰富了R的统计方法列表. 它们各自发布，需要安装和载入R. 可以在CRAN 网站上[23]得到捐献包的完整列表以及相关描述. 其中的一些包标明是*推荐*使用的, 因为它们包括了一些在数据分析中常用的统计方法. 推荐包常常和R的基本安装环境捆绑发布. 在下面的表中将会对它们进行简单的描述.

[23] http://cran.r-project.org/src/contrib/PACKAGES.html

包	描述
boot	抽样和bootstraping 方法
class	分类方法
cluster	聚类方法
foreign	读取各种格式(S3, Stata, SAS, Minitab, SPSS, Epi Info)的外部数据
KernSmooth	核密度拟合方法(包括双变量核)
lattice	grid图
MASS	Venables & Ripley 著的“Modern Applied Statistics with S”中的配套库, 包含很多有用的函数,工具和数据集
mgcv	广义的可加模型
nlme	线性和非线性混合效应模型
nnet	神经网络和多项对数线性模型
rpart	递归分割
spatial	空间分析(“kriging”, 空间协方差, . . .)
survival	生存分析

还有另外两个重要的R资源库: Omegahat项目着力于网络应用程序的统计计算[24] 并且定义R语言和一般编程语言之间的接口, Bioconductor 项目[25] 专注于生物信息类的软件(特别是生物芯片数据的分析).

一个包的安装方式决定于操作系统以及R是直接从源码编译安装还是用编译好的二进制代码安装. 在后面一种情况下,推荐采用安装CRAN 网站已经编译好的包. 在Windows 系统中, 程序Rgui.exe 有一个“Packages”菜单, 它允许通过网络或者本地的压缩文件直接安装.

如果R是本地编译的, 包可以通过源码(常常是以‘.tar.gz’作为扩展名的文件) 编译安装. 例如，如果我们想安装包gee, 首先下载文件gee_4.13-6.tar.gz (数字4.13-6 表明包的版本号; 在CRAN, 一个包常常只有一个版本号). 然后, 在系统控制台(不是R控制台) 键入如下命令:

```
R CMD INSTALL gee_4.13-6.tar.gz
```

有几个非常有用的包管理函数, 如`installed.packages`, `CRAN.packages`, 或者`download.packages`. 定期键入如下命令是非常有用的:

```
> update.packages()
```

这个命令将会比较系统安装包的版本与目前CRAN可以获得的包的版本(在Windows 系统中, 这个命令可以从“Packages” 菜单中调用). 用户可以用这个命令更新自己电脑上已经安装的包.

[24] http://www.omegahat.org/R/
[25] http://www.bioconductor.org/

6 R编程实践

自此, 我们已经对R 的功能有个全面的了解, 下面将从统计语言和编程角度来说明. 我们将会了解一些在R编程实践中采用的简单的概念.

6.1 循环和向量化

相比下拉菜单式的程序, R的一个优势在于它可以把一系列连续的操作简单的程序化. 这一点上和所有其他计算机编程语言是一致的, 但R有一些特性使得非专业人士也可以很简单的编写程序.

和其他编程语言一样, R 有一些和C语言类似的*控制结构*. 假定我们有一个向量x, 对于向量x 中值为b的元素, 我们我们把0赋给另外一个等长向量y的对应元素, 否则赋1. 我们首先创建一个与向量x等长的向量y:

```
y <- numeric(length(x))
for (i in 1:length(x)) if (x[i] == b) y[i] <- 0 else y[i] <- 1
```

几个指令可以放在一个大括弧里面:

```
for (i in 1:length(x)) {
    y[i] <- 0
    ...
}

if (x[i] == b) {
    y[i] <- 0
    ...
}
```

另外一种可能的情况是当条件为真的时候一条指令才执行:

```
while (myfun > minimum) {
    ...
}
```

但是, 由于R的一些特性, 在很多情况下循环和控制结构可以避免: *向量化*. 向量化使得循环暗含在表达式中, 前面的例子中已经多次采用了. 比如, 两个向量之和:

```
> z <- x + y
```

这种加和可以通过循环结构来实现, 就像很多编程语言采用的策略一样:

```
> z <- numeric(length(x))
> for (i in 1:length(z)) z[i] <- x[i] + y[i]
```

在这个例子中, 由于要用到向量的下标系统, 有必要预先创建一个向量z. 显然, 这种显式的循环仅仅用于向量x 和y 等长的情况: 如果不是这样程序代码需要改变, 而第一种表达方式在任何情况下都成立[26].

条件语句(if ... else) 可以用逻辑索引向量代替; 同样上面的例子:

```
> y[x == b] <- 0
> y[x != b] <- 1
```

除了让代码更简洁, 向量化的表达式在计算上效率更高, 特别是大数据量的数据集.

有多个'apply' 形式的函数用于避免使用代码的显式循环结构. apply 作用于矩阵的行或者/和列, 它的语法规则是apply(X, MARGIN, FUN, ...), 其中X是一个矩阵, MARGIN 表明是对行(1) 还是列(2), 或者二者(c(1, 2))进行操作, FUN 是一个函数(或操作符, 但是这种情况下操作符要在一个括弧里面指定[27]) , ... 是函数FUN可能的参数. 一个简单的例子如下.

```
> x <- rnorm(10, -5, 0.1)
> y <- rnorm(10, 5, 2)
> X <- cbind(x, y)  # 矩阵X的列名保持 "x" 和 "y"
> apply(X, 2, mean)
        x         y
-4.975132  4.932979
> apply(X, 2, sd)
        x         y
0.0755153 2.1388071
```

lapply() 可以用于一个列表对象: 它的语法类似apply 并且返回一个列表对象.

```
> forms <- list(y ~ x, y ~ poly(x, 2))
> lapply(forms, lm)
[[1]]
```

[26]译者注:如果对R的向量循环使用方式不了解的话,这里要小心一点.

[27]译者注:我没有用过这种情况,原文为"or an operator, but in this case it must be specified within brackets"

```
Call:
FUN(formula = X[[1]])

Coefficients:
(Intercept)            x
     31.683        5.377

[[2]]

Call:
FUN(formula = X[[2]])

Coefficients:
(Intercept)  poly(x, 2)1  poly(x, 2)2
     4.9330       1.2181      -0.6037
```

sapply()是函数lapply()一个更为灵活的变种, 它可以接受向量或者矩阵作为主要参数, 并且返回形式更为友好的结果, 常常是表格方式.

6.2 用R写程序

一般情况下, 一个R程序以ASCII格式保存,扩展名为'.R'. 如果一个工作要重复好多次, 用R程序是一个不错的选择. 在我们的第一个例子中, 我们想对三个不同种属的鸟绘制一样的图, 而且数据在三个不同的文件中. 我们将一步一步的演示看R用不同的方式去完成这个简单的问题.

首先, 我们凭直觉连续键入一系列命令, 而且预先分割图形设备.

```
layout(matrix(1:3, 3, 1))                #分割图形界面
data <- read.table("Swal.dat")           #读入数据
plot(data$V1, data$V2, type="l")
title("swallow")                         #增加标题
data <- read.table("Wren.dat")
plot(data$V1, data$V2, type="l")
title("wren")
data <- read.table("Dunn.dat")
plot(data$V1, data$V2, type="l")
title("dunnock")
```

字符'#' 用于在程序中添加注释行: R 会自动跳过注释行.

第一个程序的问题是在我们加入另外一个物种数据时, 它过长. 此外, 一些命令多次执行, 因此它们可以放在一起,在执行的时候仅仅修改一些参数. 这里的策略是把参数放到一个字符型的向量中去, 然后用下标去访问这些不同的值.

```
layout(matrix(1:3, 3, 1))                    # 分割图形界面
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
for(i in 1:length(species)) {
    data <- read.table(file[i])              # 读入数据
    plot(data$V1, data$V2, type="l")
    title(species[i])                        # 增加标题
}
```

注意代码`read.table()`里面的参数`file[i]`上面没有双引号, 因为这个参数是字符型.

现在的代码比较紧凑. 它比较容易加入新的物种, 因为设置物种名字和数据文件的向量都程序的前端.

如果扩展名为'.dat'的数据文件在R的工作目录下面, 程序可以正常运行, 否则用户要设置工作目录, 或者设置绝对路径(例如:`file <- "/home/paradis/data/Swal.dat"`). 如果程序保存在文件Mybirds.R 中, 可以通过键入如下命令执行:

```
> source("Mybirds.R")
```

和所有以文件作为输入对象的函数一样, 如果该文件不在当前工作目录下面, 用户需要提供该文件的绝对路径.

6.3 编写你自己的函数

大多数R的工作是通过函数来实现, 而且这些函数的输入参数都放在一个括弧里面. 用户可以编写他们自己的函数, 并且这些函数和R里面的其他函数有一样的特性.

编写自己的函数可以让你有效的,灵活的与合理的使用R. 我们再次使用前面读数据并且画图的例子. 如果我们想在其他情况下进行这样的操作, 写一个函数是一个不错的想法:

```
myfun <- function(S, F) {
    data <- read.table(F)
    plot(data$V1, data$V2, type="l")
    title(S)
}
```

执行时, 这个函数必须载入内存.当然, 这有多种方式实现. 和所有其他命令一样, 函数的各行可以直接通过键盘键入,或者从一个文本编辑器里面拷贝粘贴.如果函数保存在一个文本文件中, 可以命令`source()`载入. 如果用户期望一些函数在R启动时就被载入, 可以把它们保存在工作目录下面的文件`.RData`中. 另外一种方式是, 配置文件'.Rprofile' 或'Rprofile' (详见`?Startup` for details). 最后, 还可以创建一个包, 但是这里不想多讨论(见手册"编写R扩展").

一旦函数载入后, 我们就可以键入一条命令以读入数据和画出我们想要的图, 如`myfun("swallow", "Swal.dat")`. 因此, 现在我们的程序有第三个实现的版本了:

```
layout(matrix(1:3, 3, 1))
myfun("swallow", "Swal.dat")
myfun("wren", "Wrenn.dat")
myfun("dunnock", "Dunn.dat")
```

我们还可以用`sapply()` 实现程序的第四个版本:

```
layout(matrix(1:3, 3, 1))
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
sapply(species, myfun, file)
```

在R里面, 没有必要在一个函数里面进行变量声明. 当一个函数执行时, R用一种称为*词汇作用域*(lexical scoping)的规则决定一个对象的作用域相对一个函数是局部还是全局.为了理解这种机制, 我们可以认真研究一下下面的一个简单函数:

```
> foo <- function() print(x)
> x <- 1
> foo()
[1] 1
```

`x` 不是为了在函数`foo()`里面创建对象, 因此R 将会在*封装*环境中搜索是否有个名为`x` 的对象, 和打印它的值(否则一条错误信息将会显示, 执行中断).

如果`x`是我们定义的函数中一个对象的名字, 全局环境中变量`x` 值将会被采用.

```
> x <- 1
> foo2 <- function() { x <- 2; print(x) }
> foo2()
[1] 2
> x
[1] 1
```

此时, `print()` 使用在它所在的环境中定义的`x`, 即环境`foo2` 中的`x`.

前面提及的"*封装*"是关键所在.在前面两个演示函数中, 有*两个*环境: 全局环境和函数`foo` 或`foo2`的局部环境. 如果有三个或者更多的嵌套环境, 对象搜索将逐层搜索直到全局环境.

有两种方式指定一个函数的参数: 通过它们的定义时的位置或者名字(又称为*标签参数*). 例如, 假定一个函数有三个参数:

```
foo <- function(arg1, arg2, arg3) {...}
```

`foo()` 在执行时可以不用名字`arg1`, ..., 如果相应的参数对象放在相应的位置上, 如: `foo(x, y, z)`. 但是, 如果使用了参数的名字, 位置信息将会失效, 如`foo(arg3 = z, arg2 = y, arg1 = x)`. R函数的另外一个特性是函数可能采用定义时的默认设置. 例如:

```
foo <- function(arg1, arg2 = 5, arg3 = FALSE) {...}
```

命令`foo(x)`, `foo(x, 5, FALSE)` 和`foo(x, arg3 = FALSE)` 将会得到一样的结果. 使用一个函数的默认设置非常有用, 特别在使用标签参数的时候, 如`foo(x, arg3 = TRUE)` 仅仅改变一个默认设置.

在结束本章前, 我们来看另外一个例子. 尽管这个例子不是纯的统计学例子, 但是它很好地展示了R 语言的灵活性. 假定我们想研究一个非线性模型的行为: 这个模型（Ricker 模型）的定义如下:

$$N_{t+1} = N_t \exp\left[r\left(1 - \frac{N_t}{K}\right)\right]$$

这个模型广泛地用于种群动态变化（population dynamics）的研究里面,特别是鱼类的种群变化. 我们想用一个函数去模拟这个模型关于增长率r和初始群体大小N_0的变化情况(承载能力K 常常设定为1且以这个值作为默认值)的影响; 结果将以种群大小相对时间的图表示. 我们还将设定一个可选项允许用户只显示最后若干步中种群大小(默认所有结果都会被绘制出来). 下面的函数就是做Ricker模型的数值模拟的.

```
ricker <- function(nzero, r, K=1, time=100, from=0, to=time) {
    N <- numeric(time+1)
    N[1] <- nzero
    for (i in 1:time) N[i+1] <- N[i]*exp(r*(1 - N[i]/K))
    Time <- 0:time
    plot(Time, N, type="l", xlim=c(from, to))
}
```

你可以试一试下面的代码:

```
> layout(matrix(1:3, 3, 1))
> ricker(0.1, 1); title("r = 1")
> ricker(0.1, 2); title("r = 2")
> ricker(0.1, 3); title("r = 3")
```

7 R 相关的文献

手册. 在目录R_HOME/doc/manual/下面有几份R安装时分发的手册:

- *R*导论[28] [R-intro.pdf],
- *R 安装和管理* [R-admin.pdf],
- *R 数据导入/导出* [R-data.pdf],
- *编写R扩展* [R-exts.pdf],
- *R 语言定义* [R-lang.pdf].

这些文件可能以不同的格式(pdf, html, texi, ...)显示, 这取决于你采用的安装方式.

FAQ. 在目录R_HOME/doc/html/ 还有一个FAQ (*常见问题集, Frequently Asked Questions*) .R-FAQ 的版本会在CRAN网站定期更新:

http://cran.r-project.org/doc/FAQ/R-FAQ.html

在线资源. CRAN 网站拥有许多文档, 参考文献和其他有用的R 网页链接. 它还有有关R或者统计方法的出版物(书或者文章)的清单[29] 以及一些R用户编写的文档和手册[30].

邮件列表. R有四种形式的邮件讨论列表; 订购, 发布信息以及阅读所有档案记录请参阅: http://www.R-project.org/mail.html.

一般的讨论列表是'r-help'.对于R用户来说, 这是一个非常有意思的资源(其他三个列表主要面向开发用户和新版本发布).许多用户向'r-help'咨询的函数或程序在档案文件中都已有回答. 如果在使用R的过程中碰到困难, 你首先应该根据下面的建议处理一下, 然后才考虑是否给'r-help'发咨询信息:

1. 仔细阅读在线帮助文档(可能需要搜索引擎);
2. 阅读R-FAQ;
3. 搜索'r-help'档案文件, 或者使用一些网站提供的搜索引擎[31];
4. 在你提交问题前阅读"公告指南"[32].

[28] 译者注:已经有中文版发布.用google 搜索即可得到.
[29] http://www.R-project.org/doc/bib/R-publications.html
[30] http://cran.r-project.org/other-docs.html
[31] 这些网站的地址清单在http://cran.r-project.org/search.html
[32] http://www.r-project.org/posting-guide.html

R 新闻. 电子杂志*R News* 的目的是弥补电子讨论列表和传统科学文献的鸿沟. 第一期是在2001年一月发布的[33].

在文献中引用R. 最后, 如果你想在你的文章中引用R, 你必须采用下面参考文献:

> R Development Core Team (2005). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL: http://www.R-project.org.

[33] http://cran.r-project.org/doc/Rnews/