R语言简介

R语言笔记:数据分析与绘图的编程环境

版本1.7

R Development Core Team

June 10, 2006

Contents

Chapter 1

绪论与基础

1.1 R语言环境

R是一套由数据操作、计算和图形展示功能整合而成的套件。包括：

- 有效的数据存储和处理功能，
- 一套完整的数组（特别是矩阵）计算操作符，
- 拥有完整体系的数据分析工具，
- 为数据分析和显示提供的强大图形功能，
- 一套（源自S语言）完善、简单、有效的编程语言（包括条件、循环、自定义函数、输入输出功能）。

在这里使用"环境"（environment）是为了说明R的定位是一个完善、统一的系统，而非其他数据分析软件那样作为一个专门、不灵活的附属工具。

R很适合被用于发展中的新方法所进行的交互式数据分析。由于R是一个动态的环境，所以新发布的版本并不总是与之前发布的版本完全兼容。某些用户欢迎这些变化因为新技术和新方法的所带来的好处；有些则会担心旧的代码不再可用。尽管R试图成为一种真正的编程语言，但是大家不要认为一个由R编写的程序可以长命百岁。

1.2 相关的软件和文档

R可以被当作S语言（由Rick Becker,John Chambers和AllanWilks在Bell实验室开发）的实现工具，或者S-Plus系统的基本形态。

S语言的发展变化可以参考John Chambers与其他人合作的四本书。对R来说,基本的参考书是*The New S Language: A Programming Environment for Data Analysis and Graphics*（*Richard A. Becker, John M. Chambers and Allan R. Wilks*）。 对于1991年发布的S (S version 3)可以参考*Statistical Models in S* （*edited by John M. Chambers and Trevor J. Hastie*）。更多的参考书目请查看本手册的相应部分。

此外，S-Plus的相关文档都可以用于R，只是要注意R与S执行工具之间的差别。

1.3 R与统计

在我们对R语言环境的介绍中并没有提到统计，不过很多人都把R作为一个统计系统来使用。我们倾向于把它当作环境，使得经典和现代统计技术在其中得到应用。一部分已经被内建在基本的R语言环境中，但是更多的是以包的形式提供的。由8个包是随着R一同提供的（称作标准包），其它的可以通过CRAN的成员网站获得（通过HTTP://CRAN.R-PROJECT.ORG）。

通过R可以使用绝大多数的经典或者最新的统计方法，不过用户需要花一些功夫来找出这种方法。

S（和R）与其他主流的统计系统在本质上有一个很重要的不同。在S中，统计分析通常由一系列的步骤完成，同时将交互的结果存储在对象中。所以，尽管SAS和SPSS在一个回归或者判别分析中会给出丰富的输出结果，R只是给出一个最小的输出，而将结果保存在一个适当的对象中由R函数进行后续查询。

1.4 R与视窗系统

使用R最便捷的方式是在一个运行视窗系统的图形工作站上。这份指南就是为拥有这项便利的用户准备的。尽管我们绝大部分的内容都是来讲R环境的一般应用，我们还是会时不时的提到R在X WINDOW系统下的应用。

与操作系统的直接互动对多数用户来说都是必要的。在这份指南中我们主要讨论在UNIX系统下的互动，所以WINDOWS下的R用户需要做出一些小的调整。

对工作站的定制是一项直接而有效但又单调乏味的过程，在这里我们并不会作更深入的讨论。如果您在这方面遇到了困难可以向你身边的专家寻求帮助。

1.5 R的交互使用

R程序在等待输入命令时会给出提示符，默认的提示符是>, 与UNIX的SHELL提示符是相同的。不过如果你愿意的话，我们可以轻松的更改R的提示符。在这里我们先假定UNIX的SHELL提示符是$。在UNIX下使用R可以按照下面的推荐步骤来做：1.创建一个独立的子目录来存储解决这个问题所用的数据文件,将目录命名为WORK.这个目录将作为你当前任务的工作目录.

```
$ mkdir work
$ cd work
```

2.启动R的程序

```
$ R
```

3.使用R的各种命令

4.退出R

```
> q()
```

此时您会被询问是否保存您在R任务中的数据。你可以回答YES,NO或CANCEL(使用缩略字符也可以)分别对应退出前保存数据，不保存数据退出或回到R任务中。被存储的数据在之后的R任务中可以继续使用。

之后的R任务就更简单了。1.令WORK成为工作目录，并启动R程序。

```
$ cd work
$ R
```

2.使用R，在任务结束时用 `q()` 来中止。

在WINDOWS下使用R的步骤与上面基本相同。创建一个文件夹作为工作目录，并将其设定R快捷方式的在"起始位置"中。然后双击图标启动R。

1.6 入门训练

我们非常推荐读者们在继续进行之前通过一个示例来获取在计算机上使用R的感觉。这个示例由示例训练给出。

1.7 获取函数和功能的帮助信息

和UNIX中的MAN命令一样，R拥有一个内建的帮助功能。对于任意一个指定的函数，例如SOLVE，命令是

```
>help(solve)
```

或者

```
>?solve
```

对于由特殊字符指定的功能，这些参数必须用单引号或双引号括起来，使之成为一个“字符串”：同时对于某些含有IF,FOR或者FUNCTION的合成词也要这样处理。

```
> help("[[")
```

不论是单引号还是双引号都可以包含在另一个中，例如字符串："IT'S IMPORTANT"。我们的惯例是使用双引号。

一般情况下帮助文档的HTML格式都是被安装了的，可以通过运行下面的命令

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

启动一个WEB浏览器(UNIX下是NETSCAPE 浏览器)来浏览包含超级链接的帮助页面。在UNIX下，后续的帮助请求回被发送到HTML为基础的帮助系统中。页面中'SEARCH ENGINE AND KEYWORDS'连接可以通过所包含的列表对各种函数进行非常有效的查询。这是你熟悉并且理解R提供的各种功能的好方法。命令HELP.SEARCH 允许我们用多种方式来搜索帮助信息：细节和例子可以用?HELP.SEARCH 来查询。与某个主题相关的例子通常可以用下面的命令得到

```
> example(topic)
```

WINDOWS版本的R还有另外可选的帮助系统，详细资料请用

```
> ?help
```

来查询。

1.8 R的命令、对大小写的敏感，等等

从技术角度来讲，R是一种表达式语言，它的语法是非常简单的。和大多数UNIX为基础的软件包一样，R对大小写是敏感的，也就是说A和A是不同的代号并且将代表不同的变量。R语言名称中可用的字符集由当前的操作系统决定(即由LOCALE决定)。正常情况下所有的字母和数字都是可用的(在某些国家包括重音字符)，还包括句点. ***NOTE 1***，但是要注意名称不能以数字开始。基本的命令由表达式或者赋值语句组成。如果一个表达式被作为一条命令给出，它将被求值、打印而表达式的值并不被保存。一个赋值语句同样对表达式求值之后把表达式的值传给一个变量，不过并不会自动的被打印出来。命令由分号(;)来分隔，或者另起新行。基本命令可以由花括号({和})合并为一组复合表达式。注释几乎可以被放在任何地方，只要是以井号(#)开始，到行末结

束的都是注释。如果一个命令在行莫仍没有结束，R将会给出一个不同的提示符，默认的是

```
+
```

在第二行和后续行R将继续读入，直到命令从语法角度讲已经输入完成了。这个提示符也可以由用户更改。一般情况忽略继续输入提示符就行。

1.9 对已输入命令的记忆和更改

在许多版本的UNIX和Windows下，R提供了对已输入命令记忆和再次执行的一种机制。使用垂直方向箭头的按键可以在命令记录中向前或向后滚动。当一个命令用这种方法定位之后，你就可以使用左右键改变光标的位置，对命令行进行编辑(用<DEL>删除或者按其他按键进行添加)。相信的资料在稍后提供：参见命令行编辑器。记忆和编辑功能在UNIX下是高度可定制的。你可以在readline 库的手册中找到具体的方法。作为选择，Emacs的文本编辑器提供了一种更为一般的支持机制(通过ESS，Emacs Speaks Statistics)来为R下面的交互工作服务。参见R和Emacs。

1.10 命令文件的执行和输出的转向到文件

如果命令存储于一个外部文件中，比如工作目录work中的commands.R，他们可以随时在R的任务中被执行

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

在Windows中Source也可以由File菜单执行。函数sink，

```
> sink("record.lis")
```

将把所有后续的输出由终端转向一个外部文件，record.lis。命令

```
> sink()
```

将把信息重新恢复到终端上。

1.11 数据的保持与对象的清除

R所创建、操作的实体是对象。对象可以是变量、数组、字符串、函数以及由这些元素组成的其它结构。在一个R的任务过程中，对象根据名称被创建和存储(我们将在下个训练中讨论这个过程)。下面的命令

```
> objects()
```

(作为选择，ls()可以)被用来显示目前存储在R中的对象的名字。而当前存储的所有对象的组合被称为workspace。

清除对象可以使用rm命令：

```
> rm(x, y, z, ink, junk, temp, foo, bar)
```

所有在一个R任务中被创建的对象都可以在文件中被永久保存，并在其它的R任务中被使用。在每个R任务结束时用户都有机会保存当前有效的所有对象。如果用户这样做的话，对象将被写入当前目录一个名为.RData ***Note3***。当R被再次启动时R会从这个文件中再载入workspace。同时相关的命令记录也被载入。

推荐大家在用R进行不同的分析时分别使用不同的工作目录。在分析过程中创建名为x或y的对象是很普通的。在一个单独的分析中它们的含义也会很清晰，但是如果几个不同的分析在同一个工作目录中被处理的话你将会发现分辨它们的含义是件很痛苦的事情。

Chapter 2

简单操作；数值与向量

2.1 向量与赋值

R对命名了的数据结构进行操作。最简单的数据结构是数字向量，数字向量是由一组有序数字组成的单个实体。下面的R命令将创建一个名为x，包含5个数字(10.4, 5.6, 3.1, 6.4 和21.7)的向量：

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

上面的就是一个使用函数c()的赋值过程，其中函数c()的作用是将参数中的数值向量以及向量的值首尾相接组成一个新的向量。一个数字形成的向量可以被看作长度为1。注意赋值操作符(¡-) 不是通常的= 操作符，= 是为另外一种目的保留的。赋值操作符由两个字符组成¡ (小于) 和- (减)组成，而且他们应当严格的遵守：¡和-是紧靠在一起的，符号指向的是接受表达式值的那个对象。***note 5*** 赋值的操作同样可以使用函数assign()。与上面等价的赋值方法是：

```
> assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))
```

通常使用的操作符¡- 可以被当作函数assign()的简写。赋值同样可以在另一个方向进行，改变赋值操作符的方向就可以了。所以同样的复制操作还可以被写成

```
> c(10.4, 5.6, 3.1, 6.4, 21.7) -> x
```

如果一个表达式被当作一个完整的命令，它的值将被打印到终端但不被储存。所以如果我们使用这个命令

```
> 1/x
```

五个值的倒数将被打印到终端上(还有x的值)。下面的这个赋值操作

```
> y <- c(x, 0, x)
```

将创建一个向量y，其中的11项包含两个x和中间位置的一个0。

2.2 向量运算

向量可以被用于算术表达式中，操作是按照向量中的元素一个一个进行的。同一个表达式中的向量并不需要具有相同的长度。如果它们的长度不同,表达式的结果是一个与表达式中最长向量有相同长度的向量。表达式中较短的向量会根据它的长度被重复使用若干次(不一定是整数次)，直到与长度最长的向量相匹配。而常数很明显的将被不断重复。所以在上面的赋值前提下命令

```
> v <- 2*x + y + 1
```

产生一个长度为11的新向量v，逐个元素的进行运算，其中2*x被重复2.2次，y被重复1次，常数1被重复11次。逐个元素进行运算的操作符包括+，-，*，/，^ 此外所有普通的运算函数都能够被使用。log，exp，sin，cos，tan,sqrt等等，而且意义并没有什么变化。max和min的作用是选出所给向量中最大的或最小的元素。range函数的值是一个长度为2的向量，即c(min(x),max(x))。length(x)返回了向量x中元素的个数，也就是x的长度。sum(x)给出了x中所有元素的总和，prod(x)给出x中所有元素的乘积。两个统计函数是mean(x)和var(x)，分别计算样本均值和样本方差，这两个函数分别相当于sum(x)/length(x)，sum((x-mean(x))\^2)/(length(x)-1)。如果var()的参数是一个n*p的矩阵，那么函数的值是一个p*p的样本协方差矩阵，认为每行是一个p变量的样本向量。sort(x)返回一个与x具有相同长度的向量，其中的元素按招升序排列。还有其他更灵活的排序功能(参见order()和sort.list())。需要注意不论参数中有几个向量，max和min给出的是所有向量的一个最大值或最小值。而平行的最大最小函数pmax和pmin将返回一个与最长的向量长度相等的向量，向量中的元素由参数中所有向量在相应位置的最大值(最小值)组成。绝大多数用户并不会关心一个数字向量中的数字究竟是整数、实数还是复数。在计算机中运算是按照双精度的实数或复数进行的。如果要使用复数，需要直接给出一个复数部分。因此

```
sqrt(-17)
```

将会返回NaN(无效数值)和一个警告，而

```
sqrt(-17+0i)
```

将按照复数进行运算。

2.3 产生规则的序列

R拥有很多产生常用数列的方法。例如1:30就是向量c(1,2,...,29,30)。在一个表达式中冒号(:)具有最高的优先级(即最先进行运算)，所以，比如2*1:15 就是向量c(2,4, ...,28,30)。令n<-10 然后比较一下这两个序列1:n-1 和1:(n-1)。30:1这样的构造可以用来产生一个递减的序列。函数seq()可以产生更为一般的序列。函数有5个参数，但并不是每次都要全部指定。如果给出第一个和第二个参数，那么它们将指定序列的首尾，也就是说在只给出两个参数时函数的作用相当于冒号。seq(2,10)也就相当于2:10。

函数seq()和其他许多R函数的参数可以指定名称的给出，此时它们出现的次序并没有什么关系。第一二个参数的名称是from=*value* 和to=*value*，因此seq(1,30), seq(from=1, to=30) 和seq(to=30, from=1)与1:30是相同的。seq()其后的两个参数分别是by=value和length=value，分别指定序列的步长与长度。如果这两个参数没有指定的话，默认的是by=1。

例如

```
> seq(-5, 5, by=.2) -> s3
```

在变量s3中产生向量c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0)。相似的

```
> s4 <- seq(length=51, from=-5, by=.2)
```

在s4中产生一个相同的向量。

第五个参数的名称是along=vector，它只能作为唯一的参数出现，产生一个序列一个相关的函数是rep()，这个函数可以用多种复杂的方法来复制一个对象。最简单的形式是

```
> s5 <- rep(x, times=5)
```

它将把x在s5种首尾相连的复制5次。

2.4 逻辑向量

与数字向量相同，R允许对逻辑向量进行操作。一个逻辑向量的值可以是TRUE, FALSE,和NA (not available的意思)。前两个通常简写为T 和F 。注意T 和F 仅仅是默认被指向TRUE和FALSE的变量，而不是系统的保留字，因此它们可以被用户覆盖。所以你最好还是用TRUE 和FALSE。逻辑向量是由条件给出的。比如

```
> temp <- x > 13
```

令temp 成为一个与x 长度相同，相应位置根据是否与条件相符而由TRUE或FALSE组成的向量。逻辑操作符包括<, <=, >, >=, 完全相等==和不等于 != 。此外，如果c1和c2时逻辑表达式，那么 c1 & c2 是它们的交集*("and")*，c1 | c2 是它们的并集*("or")*，而!c1 是c1的反面。逻辑向量可以在普通的运算中被使用，此时它们将被转化为数字向量，FALSE当做0而TRUE当做1。不过，有些情况下逻辑向量和它们对应的数值并不是等价的，相关的例子参见下部分。

2.5 缺失值

某些情况下一个向量的成分并不全是已知的。当某个元素或者数值从统计角度讲是"不可用"("not available")或者"缺失值"("missing value")时，它们在向量中的位置将被保留，同时被赋值为一个特殊值NA。一般来讲一个NA的任何操作都将返回NA。这条规则的出发点是如果一个操作的具体要求不够完整，是不能得出结果的，因此是无效的。函数is.na(x)返回一个逻辑向量，这个向量与x有相同的长度，并且由相应位置的元素是否是NA来决定这个逻辑向量相应位置的元素是TRUE还是FALSE。

```
> z <- c(1:3,NA);  ind <- is.na(z)
```

需要注意逻辑表达式x==NA与函数is.na(x)是不同的，因为NA并不是一个真实的值，而是一个无效量的标志。所以由于逻辑表达式本身是不完整的，x==NA是一个与x具有相同长度而其所有元素都是NA的向量。另外还有一种"缺失"值由数值运算产生，被称为*Not a Number*，NaN。例如

```
> 0/0
```

或

```
> Inf - Inf
```

都将返回一个NaN，因为结果无法被明确的定义。函数is.na(xx)对于NA和NaN值都返回TRUE。要想区分它们的话，函数is.nan(xx)只对NaN值返回TRUE。

2.6 字符向量

字符和字符向量在R中都被广泛的使用，比如图表的标签。在显示的时候，相应的字符串由双引号界定，e.g., "x-values", "New iteration results"。字符串在输入时可以使用单引号(')或双以号(")，但在打印时用双引号(有时不用引号)。R使用与C语言风格基本相同的转义符，即*backslash* ，所以输入\\打印的也是\\, 而引号" 在输入时应当写作\". 其它有用的转义符包括\n, 换行, \t, tab 和\b, 回格。字符向量可以通过函数c()连接；这样的例子会经常出现的。函数paste()可以接受任意个参数，并从它们中逐个取出字符并连成字符串，形成的字符串的个数与参数中最长字符串的长度相同。如果参数中包含数字的话，数字将被强制转化为字符串。在默认情况下，参数中的各字符串是被一个空格分隔的，不过通过参数sep=string 用户可以把它更改为其他字符串，包括空字符串。这个函数的功能最好通过例子来理解。例如

```
> labs <- paste(c("X","Y"), 1:10, sep="")
```

使变量labs成为字符变量

```
c("X1", "Y2", "X3", "Y4", "X5", "Y6", "X7", "Y8", "X9", "Y10")
```

请注意参数中比较短的字符串也要被被循环使用，因此|c("X","Y")|被重复了5次来匹配数列1:10 。

2.7 索引向量(index vector);数据集子集的选择与修改

选择一个向量中元素的子集可以通过在其名称后追加一个方括号中的索引向量来完成。更一般的，任何结果为一个向量的表达式都可以通过追加索引向量来选择其中的子集。这样的索引向量有四种不同的类型。

1. 逻辑的向量。在这种情况下索引向量必须与从中选取元素的向量具有相同的长度。在索引向量中返回值是TRUE 的元素所对应的元素将被选出，返回值为FALSE的值所对应的元素将被忽略。例如

   ```
   > y <- x[!is.na(x)]
   ```

   创建了一个名为y的对象，对象中包含x中的非缺失值，同时保持顺序。请注意如果x中包含缺失值，y的长度将小于x。

   ```
   > (x+1)[(!is.na(x)) & x>0] -> z
   ```

   上面的命令创建一个对象z，其中的元素由向量x+1中与x中的非缺失值和正数对应的向量组成。

2. 正整数的向量。这种情况下索引向量中的值必须在集合{1, 2, ...,length(x)}中。在返回的向量中包含索引向量中指定元素，并且在结果中按照索引向量中的顺序排列。索引向量的长度可以是任意的，返回的向量与索引向量由相同的长度。例如x[6] 是x的第六个元素，而

   ```
   > x[1:10]
   ```

   选取了x的前10个元素(假设x的长度不小于10)。而

   ```
   > c("x","y")[rep(c(1,2,2,1), times=4)]
   ```

   产生了一个字符向量，长度为16，由 "x", "y", "y", "x" 重复4次而组成。

3. 负整数的向量。这种索引向量的作用是把某些值派出而不是包括进来。因此

   ```
   > y <- x[-(1:5)]
   ```

   向量y取得了前5个元素以外的值。

4. 字符串的向量。这种可能型只存在于拥有names属性并由它来区分向量中元素的向量。这种情况下一个由名称组成的子向量起到了和正整数的索引向量相同的效果。

   ```
   > fruit <- c(5, 10, 1, 20)
   > names(fruit) <- c("orange", "banana", "apple", "peach")
   > lunch <- fruit[c("apple","orange")]|
   ```

由字母和数字组成的*名称(names)*比单纯的*数值索引*更好记是它最大的优点。这个功能在使用数据框的时候非常有用，我们在后面会看到。一个被索引的表达式。这个表达式应当有`vector[index_vector]`这样的形式，因为如果在`vector`的位置随便放上一个表达式并没有什么意义。

```
> x[is.na(x)] <- 0
```

replaces any missing values in x by zeros and

```
> y[y < 0] <- -y[y < 0]
```

has the same effect as

```
> y <- abs(y)
```

2.8 对象的其他类型

向量是R中最重要的对象类型，不过在后面的章节我们还会遇到其他的几种。

- *矩阵(matrices)* 或者更一般的说*数组* 是向量在多维情况下的一般形式。事实上它们是可以被两个或更多的指标索引的向量，并且以特定的方式被打印出来。参见数组和矩阵
- *因子(factors)* 提供了一种处理分类数据的更简介的方式。参见因子
- *列表(lists)* 是向量的一种一般形式，并不需要保证其中的元素都是相同的类型，而且其中的元素经常是向量和列表本身。列表为统计计算所返回的结果提供了一种便捷的方式。
- *数据框(data frames)*是一种与矩阵相似的结构，其中的列可以是不同的数据类型。可以把数据框看作一种数据"矩阵"，它的每行是一个观测单位，而且(可能)同时包含数值型和分类的变量。很多试验可以通过数据况进行描述:处理是分类的和应答是数值的。参见数据框
- *函数(functions)*是能够在R的workspace中存储的对象。我们可以通过函数来扩展R的功能。参见编写自己的函数。

Chapter 3

对象，模式和属性

3.1 固有属性：模式和长度

R所进行操作的实体是对象。比如由实数或复数、逻辑值、字符串组成的向量。由于它们的成分都是同一类型或者模式，所以它们可以被看作最基本的结构，分别称为数值(实值)型(numeric)，复值型(complex)，逻辑型(logical)以及字符型(character)。

向量中的值必须是相同模式的。因此任何给定的向量必定是逻辑型、数值型、复值型或字符型中的一种。唯一一个不那么重要特例就是对无效量的NA值。请注意即使一个空向量仍然有它的模式。比如一个空的字符串向量打印为character(0)，一个空的数值向量打印为numeric(0)。

R的操作对象还包括列表，它的模式是列表型(list)。列表是对象的有序序列，其中的元素可以是任意模式的。与最基本的结构不同的是，列表是一种"递归"的结构，也就是说它们中的元素也可以是列表。

递归结构还包括其他的模式——*函数(function)*和*表达式(expression)*。函数和表达式都是R系统的重要的组成部分，在后面的部分我们会深入的讨论函数，而表达式我们只是会在公式建模部分间接的讨论到。

通过模式我们可以确定对象的基本类型。模式是对象的一种特性。对象的另一种特性是它的长度。函数mode(object)和length(object)可以返回对象的模式和长度。

更深入的属性可以通过attributes(object)获得，参见取得和设置属性。因此，模式和长度也被称为对象的"固有属性"。

例如，令z是一个长度为100的复值向量，在表达式中mode(z)的值为"complex"，length(z)的值为100。

在允许的情况下(大多数情况都是允许的)，R可以完成各种模式的转换。例如

```
> z <- 0:9
```

我们可以通过命令

```
> digits <- as.character(z)
```

将`z`转化为字符向量`c("0", "1", "2", ..., "9")`后赋值给`digits`。我们还可以进一步的强制转化(coercion)，或称转化模式，将`digits`转化为数值向量:

```
> d <- as.integer(digits)
```

此时`d`和`z`是相同的。在R中存在很多类似的，形式为`as.something()`的函数，可以完成从一个模式向另一个模式的转化，或者是令对象取得它当前模式

不具有的某些属性。通过帮助文档，用户应当会对他们更加熟悉。

3.2 改变对象的长度

一个"空"对象具有模式。例如

```
> e <- numeric()
```

产生一个模式为数值型的空向量。同样的，character()是一个字符式的空向量等等。当一个对象被创建之后，不论对象的长度是多少，新添加的元素都将被赋予一个索引值，这个索引值将在原有索引值范围之外。因此

```
> e[3] <- 17
```

另e为一个长度为3的向量（此时，该向量的前两个元素都是NA）。这个规则对任何结构都是有效的，新增元素的模式与对象中第一个位置的元素相同。

这种对对象长度的自动调整是经常被使用的。例如使用scan()进行输入时。（参见）

相反的，如果要缩短（截断）一个对象的长度，只需要一个赋值命令。因此，若alpha是一个长度为10的对象，下面的命令

```
> alpha <- alpha[2 * 1:5]
```

使之成为一个长度为5，仅包含前五个索引值为偶数的元素的对象。变换后原有的索引值不被保留。

3.3 属性的获取和设置

函数attributes(object)将给出当前对象所具有的所有非基本属性（长度和模式属于基本属性）的一个列表。函数attr(object,name)可以被用来选取一个指定的属性。除了为某些特殊的目的创建新属性这样特殊的环境下，这些函数很少被用到，比如将一个R对象与创建日期或一个操作符相关联时。不过，这种概念是相当重要的。

在赋予或删除属性的时候需要慎重的作些检查。因为属性是R的对象系统不可分割的一部分。

当函数attr()用在赋值语句左侧时，既可以是将对象与一种新的属性关联，也可以是对原有属性的更改。例如

```
> attr(z,"dim") <- c(10,10)
```

另R将z作为一个10×10的矩阵看待。

3.4 对象的类别

对象的一个特别属性，*类别*，被用来指定对象在R编程中的风格。

例如，如果一个对象的类别是"data.frame"，它将按照一种特定的方式被打印出来，函数plot()在绘图时也会按照特定的方式来进行，其他被称作通用函数的函数，例如summary()在处理时也会对它的类别作出特定的反应。

如果要临时去除类别的作用，可以使用函数unclass()。例如，如果winter的类别为"data.frame"，那么

```
> winter
```

将按照数据帧的形式来打印它，和矩阵的形式差不多，而

```
> unclass(winter)
```

将把它按照一个普通的列表来打印。这个功能你只会在一些相当特殊的情况下才会用到，其中一个便是当你要学习类别和通用函数的概念时。

在sec10.9我们还会再讨论通用函数和类别，不过只是简要的。

Chapter 4

有序因子与无序因子

因子[1] 是一种向量对象，它给自己的组件指定了一个离散的分类（分组），它的组件由其他等长的向量组成。R提供了有序因子和无序因子。因子的“实际”应用是与模型公式相联系的（参见），我们在这里先看

4.1 一个特例

假设，我们有从澳大利亚来的30个税务会计作为样本他们各自所在省份由下面的字符向量指定

```
> state <- c("tas", "sa", "qld", "nsw", "nsw", "nt", "wa", "wa",
             "qld", "vic", "nsw", "vic", "qld", "qld", "sa", "tas",
             "sa", "nt", "wa", "vic", "qld", "nsw", "nsw", "wa",
             "sa", "act", "nsw", "vic", "vic", "act")
```

在这里提示读者：在一个字符向量中，“排序的”是值按照字母顺序被排序。

此时用函数`factor()`创建一个*因子*

```
> statef <- factor(state)
```

函数`print()`对因子的处理与其它对象略有不同。

```
> statef
[1] tas sa qld nsw nsw nt wa wa qld vic nsw vic qld qld sa
[16] tas sa nt wa vic qld nsw nsw wa sa act nsw vic vic act
Levels: act nsw nt qld sa tas vic wa
```

函数`levels()`可以用来观察因子中有多少不同的水平。

4.2 函数tapply()与ragged数组

继续前面的例子，假设我们拥有这些会计的收入数据

```
> incomes <-  c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42, 56,
                61, 61, 61, 58, 51, 48, 65, 49, 49, 41, 48, 52, 46,
                59, 46, 58, 43)
```

[1]此处的有序是指其因子水平顺序是指定的，而无序则表明因子的水平没有经过处理，一般是按照字母顺序排列的，所以称有序因子为定序因子似乎更恰当。

此时通过函数tapply()可以计算个省份会计收入的样本均值

```
> incmeans <- tapply(incomes, statef, mean)
```

包含所得均值的向量在显示时由其水平标记

```
  act    nsw     nt    qld     sa    tas    vic     wa
44.500 57.333 55.500 53.600 55.000 60.500 56.000 52.250
```

函数tapply()的作用是对它第一个参数的组件中所包含的每个组应用一个函数，本例中是对incomes应用函数mean()，而incomes的水平由tapply()的第二个参数statef定义。函数的结果是一个长度与因子水平数相等的结构。上面的例子是比较一般的情况，即incomes与statef是两个单独变量时tapply()的应用方法。更详细的资料读者可以通过帮助文档查询。

假设我们还要进一步的对各省税务会计收入均值的标准误进行计算。我们需要编写一个简单的R函数来计算任何给定向量的标准误。由于R内建一个计算样本方差的函数var()，所以我们要做的只是写一个一行的函数，并通过赋值语句指定函数的名称

```
> stderr <- function(x) sqrt(var(x)/length(x))
```

（函数的编写将在稍后的章节讲述，参见）赋值完成后我们就可以这样计算标准误了：

```
> incster <- tapply(incomes, statef, stderr)
```

所求得的值为

```
> incster
act    nsw  nt    qld     sa tas   vic     wa
1.5 4.3102 4.5 4.1061 2.7386 0.5 5.244 2.6575
```

作为一个练习，你可能还想得到收入均值95%的置信区间。可以通过下面的方法完成：先用tapply()应用函数length()得到样本长度，然后用函数qt()来获得t分布的分位点。

函数tapply()可以通过多类别的方法处理更复杂的向量索引。例如，我们可能像依据省份和性别来分割税务会计的数据，在一个简单的例子中（只有一个类别），我们的思路可以是这样的：根据类别中不同的项，向量中的值被分成不同的组，然后，函数被分别应用于每一个组。返回指是函数结果的向量，由类别的水平标记。

一个向量和一个标记用的因子合并有时会成为一个ragged array，因为子类别的大小可能是不规则的。当子类别的大小全都相同时，合并过程中会自动完成索引，而且这样显然会更有效率，正如我们在下一章将要看到的那样。

4.3 有序因子

因子的水平按照字母顺序存储，不过如果被明确的指定，他们将按照指定的顺序存储。有时因子的水平具有其原始的顺序，而且这种顺序可以在我们的统计分析中被用到，所以我们需要一定的方法来记录这种顺序。函数ordered() 可以创建这种有序因子，但是这种有序因子同*因子*是有差别的。在多数情况下，有序因子和无序因子的差别仅仅是前者在输出结果是其水平，不过在拟合线性模型时，两种因子是有实质差异的。

Chapter 5

数组和矩阵

5.1 数组

数组可以看成一个由递增下标表示的数据项的集合，例如数值。R语言对创建和处理数组及其特例矩阵提供了简单而方便的功能，尤其是对矩阵。

一个矩阵就是一个2维数组。维数向量中的值规定了下标k的上限。下限一般为1。如果一个向量需要在R中以数组的方式被处理，则必须含有一个维数向量作为它的`dim`属性。

假设，例如，`z`是一个由1500个元素组成的向量。下面的赋值语句

```
> dim(z) <- c(3,5,100)
```

使它具有`dim`属性，并且将被当作一个$3 \times 5 \times 100$的数组进行处理。

在更简单和一般化的赋值过程中还可以用到像`matrix()`和`array()`这样的函数。读者可以参考（）

当数据向量中的值被赋给数组中的值时，将遵循与FORTRAN相同的原则—”主列顺序”，即第一个下标变化的最快，最后的下标变化最慢。

例如，一个数组`a`的维数向量是`c(3,4,2)`，则它包含24个数据项，这些数据项在数据向量中的顺序是`a[1,1,1]`, `a[2,1,1]`,..., `a[2,4,2]`, `a[3,4,2]`。

5.2 数组的索引和数组的子块

正如上面所提到的，数组中的单个元素可以通过下标来指定，下标由逗号分隔，写在括号内。

更一般的，我们可以通过在下标的位置给出一个索引向量来指定一个数组的子块，不过如果*在任何一个索引位置上给出空的索引向量，则相当于选取了这个下标的全部范围*。

继续上面的例子，`a[2,,]`是一个4×2的数组，维数向量为`c(4,2)`，数据向量中包含下面这些值

```
c(a[2,1,1], a[2,2,1], a[2,3,1], a[2,4,1],
  a[2,1,2], a[2,2,2], a[2,3,2], a[2,4,2])
```

其顺序与给出顺序的相同。

`a[,,]`代表了整个数组，相当于省略所有下标，单独使用`a`。

对任意数组，比如说`Z`，其维数向量都可以用`dim()`来指代（在赋值语句的任意一侧）。

而且，如果一个给出的数组名称中只有一个*下标或单个索引向量*，那么，只有数据向量中的相应值会被用到；这种情况下维数向量是被忽略的。不过，如果给出的单个索引不是向量而是一个数组，就不在我们上面的讨论范围内了，而是我们下面将要讨论的。

5.3 索引数组

除了在任意一个下标位置使用索引向量之外，数组还可以在下标处使用*索引数组*，这样可以把数值向量中不规则的一组值赋值到数组中，也可以把数组中的一组不规则的值释放到一个向量中。

我们可以用一个矩阵的例子来解释这个过程，在有双下标索引的数组的情况下，一个索引矩阵将包含两列和所需的行数。索引向量中的项是双下标索引数组的行索引和列索引。假定我们有一个4×5的数组`X`，并且希望完成下面的工作

- 以向量的形式释放元素`X[1,3]`, `X[2,2]` 和 `X[3,1]`
- 将数组中的这些元素用`0`替换

此时，我们需要一个3×2的下标数组，请看下面的例子

```
> x <- array(1:20,dim=c(4,5)) # Generate a 4 by 5 array.
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> i <- array(c(1:3,3:1),dim=c(3,2))
> i
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
> x[i]
[1] 9 6 3
> x[i] <- 0 # Replace those elements by zeros.
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    0   13   17
[2,]    2    0   10   14   18
[3,]    0    7   11   15   19
[4,]    4    8   12   16   20
>
```

作为一个不太重要的例子，假定，我们要为一个由因子`blocks`(`b` levels) 和`varieties` (`v` levels) 定义的区组设计生成一个设计矩阵。进一步的，我们假设实验中包括`n`个计划（plot），我们可以进行如下操作：

```
> Xb <- matrix(0, n, b)
> Xv <- matrix(0, n, v)
```

```
> ib <- cbind(1:n, blocks)
> iv <- cbind(1:n, varieties)
> Xb[ib] <- 1
> Xv[iv] <- 1
> X <- cbind(Xb, Xv)
```

构建关联矩阵，比如叫N，我们可以使用

```
> N <- crossprod(Xb, Xv)
```

不过，构件这个矩阵，更简单直接的方法是使用函数table():

```
> N <- table(blocks, varieties)
```

5.4 函数array()

除了通过赋予一个向量以dim属性，我们还可以用函数array来从向量构建数组。函数形式为

```
> Z <- array(data_vector, dim_vector)
```

例如，若向量h包含24，或者更少个数值，那么命令

```
> Z <- array(h, dim=c(3,4,2))
```

将用h的数值在Z中创建一个$3 \times 4 \times 2$的数组。如果h的大小恰好是24，那么命令的效果等同于

```
> dim(Z) <- c(3,4,2)
```

不过，如果h的大小小于24，它的值将被重复使用直到凑足24个。（参见）作为一个极端但是很常见的例子

```
> Z <- array(0, c(3,4,2))
```

使E成为一个全零的数组。

此时，dim(Z)代表维数向量c(3,4,2)，Z[1:24]代表数据向量，Z[]和Z都代表整个数组。

数组可以在算数表达式中使用，结果也是一个数组，这个数组由数据向量逐个元素的运算后组成，，通常参与运算的对象应当具有相同的dim属性。而且这将作为最终结果的维数向量。所以如果A,B,C是相似的数组，那么

```
> D <- 2*A*B + C + 1
```

令D成为一个与数据向量相似的数组，而且很明显的，结果将是逐个元素进行运算后得出的。不过，涉及到向量和数组混合运算的法则还需要更进一步而且更精确的说明。

5.4.1 向量，数组的混合运算，重复使用规则

向量，数组混合运算的精确法则会让人感觉有些怪异，而且很难在参考书中找到。按照经验，我们发现下面的这些规则是值得信赖的一些参考。

- 表达式从左到右被扫描
- 参与运算的任意对象如果大小不足，都将被重复使用直到与其他参与运算的对象等长
- 有且只有较短的向量和数组在运算中相遇时，所有的数组必须具有相同的dim属性，或者返回一个错误。(As long as short vectors and arrays only are encountered, the arrays must all have the same dim attribute or an error results.)

- 如果有任意参与运算的向量比参与运算的矩阵或数组长，将会产生错误。
- 如果数组结构正常声称，并且没有错误或者强制转换被应用于向量上，那么得到的结果与参与运算的数组具有相同的dim属性。

5.5 两个数组的外积

对数组一项很重要的操作是外积，如果a,b是两个数值型数组，他们的外积是一个数组，其维数向量由二者的维数向量连接而成（顺序与结果有关），而其数据向量由a的数据向量与b的数据向量中所有元素的所有可能乘积组成，外积的操作符是%o%:

```
> ab <- a %o% b
```

或者

```
> ab <- outer(a, b, "*")
```

其中的乘法操作可以由任意一个双变量的函数替代，假设我们想对函数$f(x;y) = cos(y)/(1+x2)$求值，其中x, y的坐标值分别由R向量x,y提供，我们可以使用下面的方法：

```
> f <- function(x, y) cos(y)/(1 + x^2)
> z <- outer(x, y, f)
```

特别的，两个普通向量的外积是一个双下标的数组（即一个矩阵，秩至多为1）。注意，外积操作显然是不具有互换性的（即操作符左右两边互换会导致结果的变化）。在chapter10 我们会进一步的讨论如何定义你自己的R函数。

一个例子：2×2数字矩阵的行列式

作为一个人工不过相当精巧的例子，考虑一个2×2矩阵$[a, b; c, d]$的行列式，其中每项都是取值于$0 \sim 9$区间内的非负整数。

我们的目标是求得此区间上所有形如$ad - bc$的行列式的值。并通过密度图显示每个值出现的频率高低，这相当于寻找如果行列式中每个值都是独立均匀的选取时，其行列式值的概率分布。

一个灵巧的方法是使用函数outer()

```
> d <- outer(0:9, 0:9)
> fr <- table(outer(d, d, "-"))
> plot(as.numeric(names(fr)), fr, type="h",
       xlab="Determinant", ylab="Frequency")
```

请注意，在这里通过将频数表的names强制转化为数值型，我们的到了行列式值的区间。而处理这个问题更显而易见的方式是使用for循环，但是由于使用for循环的效率太低，并不适合用在这里。关于for循环我们会在chapter9 讲到。

5.6 数组的广义转置

函数aperm(a, perm)可以用来对一个数组进行排列，参数perm必须是整数$1, \ldots, k$的一个排列，此处k是a下标的个数。这个函数得到一个与a相同大小的数组，原有的维被perm[j]给定的新维取代。理解这个操作最简单的方式应当是把它来

当作一种广义的矩阵转置。如果A是一个矩阵（也就是一个双下标数组）由命令

```
> B <- aperm(A, c(2,1))
```

得到的B其实就是A的转置。对这个特定的操作，我们由一个更简单的函数t()，所以我们还可以用B <- t(A)。

5.7 专门的矩阵功能

如上所述，矩阵仅仅是一个双下标的数组。然而作为一个非常重要的特例，它需要一些单独的讨论。R包含许多针对矩阵（仅对矩阵有效）的操作符和函数。例如上面提到的矩阵转置函数t(X)。函数nrow(A)和ncol(A)分别返回矩阵A的行数和列数。

5.7.1 矩阵乘法

矩阵乘法的操作符为%*%。一个$n \times 1$或$1 \times n$的矩阵在适当的情况下显然可以作为一个n维向量来使用。相反的，在矩阵乘法表达式中出现的向量将被自动的转化为行向量*或者*列向量，而转化成何种形式取决于在表达式中哪种形式是可乘的。（不过在后面我们会看到，这个变换并不总是毫无歧义的）

例如：A,B是同样大小的方阵，命令

```
> A * B
```

是相应位置上元素乘积组成的矩阵，而

```
> A %*% B
```

是矩阵乘积。如果X是向量，那么

```
> x %*% A %*% x
```

是一个二次型。

函数crossprod()产生一个交叉乘积（cross product），即crossprod(X, y)与t(X) %*% y达成相同的效果，但crossprod()的效率更高。如果省略第二个参数，相当于第一个参数与自己做运算，即crossprod(X)等价于crossprod(X, X)。

函数diag()的作用取决于它的参数，如果v是向量，diag(v)返回一个由v的元素为对角元素的对角矩阵。若M为矩阵，diag(M)返回一个由M主对角元素组成的向量。这沿用了diag()在Matlab 中的传统用法。此外，多少有些让人迷惑，如果k只是一个数值，那么diag(k)是一个$k \times k$的单位矩阵。

5.7.2 线性方程和矩阵的逆

求解线性方程是矩阵乘法的逆运算。在

```
> b <- A %*% x
```

之后，若只有A和b被给出，则向量x是线性方程的解。在R中

```
> solve(A,b)
```

对线性方程求解。返回x的值（可能会有一些精度上的损失），请注意，人们很少通过solve(A)求A的逆A^{-1}，并在线性代数式$x = A^{-1}b$中使用。因为与solve(A,b)相比x <- solve(A) %*% b既不稳定又没有效率。

在多元计算中要使用的二次型$x'A^{-1}x$最好通过x %*% solve(A,x)来计算，而不是通过计算A的逆。

5.7.3 特征值和特征向量

函数eigen(Sm)返回一个对称矩阵的特征值和特征向量。这个函数的结果是由名为values和vectors的两部分组成的列表。操作

```
> ev <- eigen(Sm)
```

把这个列表赋值给ev。而ev$val和ev$vec分别是Sm的特征值向量和对应的特征向量组成的矩阵。如果只是需要特征值，我们可以使用：

```
> evals <- eigen(Sm)$values
```

此时evals包含特征值的向量，而第二部分则被弃用了。单独使用

```
> eigen(Sm)
```

将连同名称打印列表> eigen(Sm)的两部分。

5.8 奇异值分解与行列式

[1]

5.9 最小二乘拟合及QR分解

函数lsfit()返回由最小二乘拟合的结果组成的列表。形如

```
> ans <- lsfit(X, y)
```

的赋值操作返回最小二乘拟和的结果，其中y是观测值向量，X为设计矩阵，通过帮助功能可以获得更多的详细资料，还可以获得后续函数ls.diag()的信息，ls.diag()的功能包括回归诊断等等。请注意，一个总平均项已经默认包含在X中，所以不必在X中再增加这一列。

另一个密切联系的函数是qr()和与它的相关函数。考虑下面的赋值

```
> Xplus <- qr(X)
> b <- qr.coef(Xplus, y)
> fit <- qr.fitted(Xplus, y)
> res <- qr.resid(Xplus, y)
```

这些操作将分别计算y在x区间上的正交投影。

5.10 构建分区矩阵，cbind()和rbind()

正如我们在前面见到的，矩阵可以由其它向量和矩阵通过函数cbind()和rbind()构建。简单的说，cbind()按照水平方向，或者说按列的方式将矩阵连接到一起。rbind()按照垂直的方向，或者说按行的方式将矩阵连接到一起。

在赋值语句

```
> X <- cbind(arg 1, arg 2, arg 3, ...)
```

中，cbind()的参数必须是任意长度向量或具有相同列数的

5.11 连接函数c()，针对数组的应用

我们应当注意下面的问题：cbind()和rbind()是考虑dim属性的连接函数，而基础（更低级）的c()则不然，它会清除数值型对象的所有dim,dimnames属性。在某些情况下，我们可以利用这个特性。

[1] 奇异值分解的介绍可以参考http://mathworld.wolfram.com/SingularValueDecomposition.html

将一个数组强制转化为一个简单向量，比较正是的方法是使用as.vector()

不过我们可以通过使用单参数的c()来达到相似的效果：
两种方法之间有些细小的差异，但是使用中对二者的选择更多的只是习惯问题，多数倾向于前一种方法。

5.12 由因子生成频数表

请回忆一下，一个因子可以将一个区块定义为若干个组。相似的，一对因子可以定义一个二维的交叉分类，还可以有更多的因子。函数table()可以由等长的因子得出相应的频数表。如果有一个k类的参数，那么结果就是一个k维的频数数组。

例如，假定statef是一个数据向量中每一项的省份代码构成的向量，命令

将样本中各省份的频数表赋值给因子statefr，其中的频数按照类别的水平（level）标识并排序。

这个简单的例子与

等价，但更方便易用

我们进一步假定incomef是数据向量中每一项通过cut()函数生成的”income class”构成的因子

然后计算一个二维的频数表

由此可以扩展到更高维的频数表。

Chapter 6

列表和数据帧

6.1 列表

R列表是由称作组件的有序对象集合构成的对象。

对于组件，并未特定的要求他们是相同模式或类型。而且，一个列表可以同时包含数值向量，逻辑值，矩阵，复值向量，字符数组和函数等等。下面是构造列表的一个简单例子

```
> Lst <- list(name="Fred", wife="Mary", no.children=3,
child.ages=c(4,7,9))|
```

组件总是被*编*号的，并且可以通过编号指定。因此，如果Lst是一个有四个组件组成的列表，则其中的组件可以分别被指定为Lst[[1]], Lst[[2]], Lst[[3]] 和Lst[[4]]。更进一步的，如果Lst[[4]]是一个有下标的数组，Lst[[4]][1]就是它的第一项。

如果Lst是一个列表，那么函数length(Lst)返回列表（最高级别）组件的个数。列表的组件也可以是被命名的。这种情况下，组件既可以通过在双层方括号中的给出名称的方式来指定，也可以使用下面这种更方便的形式

¿ name$component name

来完成同样的功能。

这条规则非常有用，因为它可以让你在忘记编号的时候仍能轻松选取正确的组件。

所以在上面那个简单的例子中，
与Lst[[1]]相同，Lst$name代表字符串"Fred"
与Lst[[2]]相同，Lst$wife代表字符串"Mary"
与Lst[[4]][1]相同，Lst$child.ages[1]代表数字4。

此外，用户还可以在双层方括号中使用列表组件的名称，比如Lst[["name"]]与Lst$name等价。这种方式在要选定的组件名称存储于另一个变量中时特别有用，

```
> x <- "name"; Lst[[x]]
```

请注意区分Lst[[1]]和Lst[1]，'[[...]]'是选择单个元素时使用的操作符，而'[...]'是一个一般的下标操作符。因此，前者代表列表Lst中的第一个对象，而且如果列表已命名，对象的名称并不包含在所指定的对象里；后者是列表Lst的子列表，*仅包含列表的第一项。而如果列表已命名，其名称也包含到所指定的对象里*。

组件的名称可以缩写，可缩写的程度是只要能令组件被唯一的识别就可以了。因此Lst$coefficients 可以缩写到Lst$coe，而Lst$covariance可以缩写到Lst$cov，这是在可识别前提下最短的缩写了。

名称的向量也是列表的属性，自然也可以同样被处理。不仅是列表，其他结构也可以包含names属性。

6.2 构建和修改列表

新的列表可以通过`list()`函数从现有的对象中建立。型如

```
> Lst <- list(name_1=object_1, . . ., name_m=object_m)
```

的赋值将创建一个包含m个组件的列表，并根据参数中指定的名称为其命名。（其名称可以自由选取）。如果它们的名称被省略，组件将只是被编号。在创建列表的过程中，所使用的组件是被复制到新的列表中的，对原始对象没有影响。

同其他有下标对象一样，也可以通过指定额外组件的方式扩展列表。例如

```
> Lst[5] <- list(matrix=Mat)
```

6.2.1 连接列表

当连接函数`c()`的参数为列表时，其结果也是一个模式为列表的对象。由参数中的列表作为组件依次连接而成。

`> list.ABC <- c(list.A, list.B, list.C)` 请回忆一下连接函数以向量为参数时，也是将所有参数连接到一起形成一个单独的向量结构。在这种情况下，所有其它属性，例如dim属性，都失去了

6.3 数据帧

数据帧 是类别为"`data.frame`"的列表，下面对列表的限制对数据帧也有效。

- 组件必须是向量（数值型，字符形，逻辑型），因子，数值矩阵，列表，或其他数据帧
- 矩阵，列表，数据帧向新数据帧提供的变量数分别等于它们的列数，元素数和变量数
- 数值向量，逻辑值和因子在数据帧中保持不变，字符向量将被强制转化为因子，其水平是字符向量中所出现的值。
- 数据帧中作为变量的向量结构必须具有相同的长度，而矩阵结构应当具有相同的行大小

很多情况下，数据帧会被当作各列具有不同模式和属性的矩阵。数据帧按照矩阵的方式显示，选取的行或列也按照矩阵的方式来索引。

6.3.1 创建数据帧

那些满足对数据帧的列（组件）限制的对象可以通过函数`data.frame`来构建成为一个数据帧

```
> accountants <- data.frame(home=statef, loot=income, shot=incomef)
```

如果一个列表的组件与数据帧的限制一致，这个列表就可以通过函数`as.data.frame()`*强制转化*为一个数据帧。

创建数据帧最简单的方法应当是使用`read.table()`函数从外部文件中读取整个数据帧。这将在chapter 7中讨论。

6.3.2 attach()与detach()

标记$，如account$statef中的，在组件中的使用并不总是非常方便。一个有用的功能可以使列表或数据帧的组件暂时像变量一样可见。保持它们的组件名称，而无须每次都还引用列表名称。

函数attach()的参数既可以是一个directory name，也可以是一个数据帧。假定lentils是含有三个变量lentils$u,lentils$v,lentils$w 的数据帧，操作

```
> attach(lentils)
```

将数据帧置于搜索路径的位置2，而位置1上并没有变量u,v,w，这些来自数据帧的变量仅在他们自己的权限范围内有效。此时，类似

> u <- v+w 的赋值操作并不会替代数据帧中的组件u，而是由工作目录搜索路径位置1上的另一个变量u遮盖了。如果要对数据帧本身作永久的改变的话，最简单的方式是求助于$标记：

> lentils$u <- v+w 然而组件u的新值必须在数据帧卸载（detach）再重新挂接（attach）之后才可见。

卸载数据使用函数

```
> detach()
```

更精确的说，这个操作将搜索路径位置2的实体卸载了。因此当前内容中变量u,v,w不再可见。除非使用列表标记lentils$u这样的方式。存储在大于2的位置上的实体可以通过detach加上位置编号的方式卸载。不过，更安全的方法是使用名称，例如detach(lentils)或detach("lentils")

NOTE： 当前发行版的R可以在搜索路径中包含至多20个项目。避免多次挂接同一个数据帧。在结束对数据帧中变量的使用后尽快将其卸载。

NOTE： 当前发行版的R中，列表和数据帧只能在位置2或更靠后的位置上挂接。不能对一个挂接的列表或数据帧直接赋值。（因此，某种意义上将它们是静态的）

6.3.3 使用数据帧

在同一个工作目录下方便的处理多个不同问题，可以遵循下面的惯例。

- 将每个独立的，适当定义的问题所包含的所有变量收入同一个数据帧中，并赋予合适的、易理解、易辨识的名称；
- 处理问题时，当相应的数据帧挂接于位置2，同时在第1层工作目录下存放操作的数值和临时变量；
- 在结束一次工作之前，将你认为对将来有参考价值的变量通过$标记的形式添加到数据帧里面，然后detach()；
- 最后，将工作目录下所有不需要的变量剔除，并且尽量将剩下多余的临时变量都清除干净。

这样我们可以很简单的在同一个目录下处理多个问题，而且对每个问题都可以使用x,y,z这样的变量名。

6.3.4 挂接任意列表

attach()是具有一般性的函数，即它不仅能够将目录和数据帧挂接在搜索路径上，还能挂接其他类别的对象。特别是模式为"list"的对象可以通过相同的方式挂接：

```
> attach(any.old.list)
```

任何被挂接的对象都可以用detach来卸载，通过指定位置编号或者指定名称这样的方式。

6.3.5 管理搜索路径

函数search将显示目前的搜索路径。所以用来跟踪已挂接或已卸载的数据帧、列表（以及功能包）是非常有用的。

初始的结果是

```
> search()
[1] ".GlobalEnv" "Autoloads" "package:base"
```

此处，.GlobalEnv是工作区（workspace）[1]

当lentils被挂接后

```
> search()
[1] ".GlobalEnv" "lentils" "Autoloads" "package:base"
> ls(2)
[1] "u" "v" "w"
```

正如我们所见ls（或objects）命令可以用来检查搜索路径任意位置上的内容。最后，我们卸载数据帧，并且确认它们已被从搜索路径上删除。

```
> detach("lentils")
> search()
[1] ".GlobalEnv" "Autoloads" "package:base"
```

[1]关于第二项的含义请通过在线帮助查询autoload

Chapter 7

从文件中读取数据

通常大规模的数据对象都是从外部文件中读取值，而不是在R任务运行过程中从键盘录入。R的输入功能比较简单，其要求相当严格，很不灵活。R的设计者认为你完全可以通过其他的工具，例如文件编辑器和Perl [1]来修改输入文件，使之符合R的要求。一般来说，这还是很简单的。

如果变量可以通过数据帧存储，正如我们所建议的，我们就可以通过函数`read.table()`直接将完整的数据帧读出。还有一个更原始的输入函数`scan()`，可以直接读入。

关于R数据输入输出的详细信息请参考*R data Import/Export* 手册。

7.1 函数read.table()

要直接将整个数据帧读出，所用的外部文件需要符合特定的格式。

- 第一行应当提供数据帧中每个变量的名称
- 此外的每一行中包含一个行标号（必须在第一项的位置）和其它各变量的值

如果文件的第一行所包含的项目比第二行少，也被认为是有效的。所以一个数据帧的数据来源文件，其前几行可以是下面这样的。

包含名称和行标号的输入文件形式

	Price	Floor	Area	Rooms	Age	Cent.heat
01	52.00	111.0	830	5	6.2	no
02	54.75	128.0	710	5	7.5	no
03	57.50	101.0	1000	5	4.2	no
04	57.50	131.0	690	6	8.8	no
05	59.75	93.0	900	5	1.9	yes
...						

默认情况下，数值项（出了行标号）将被当作数值变量读入。非数值变量，如例子中的`Cent.heat`，将被作为因子读入。如果需要的话，这个规则可以改变。

之后，函数read.table()就可以直接读出一个数据帧了。

[1] 在Unix下可以使用Sed或Awk

```
> HousePrice <- read.table("houses.data")
```

经常的你会希望直接忽略行标号，使用系统默认的标号。在这个例子中，可以通过下面的方式忽略行标号这列：

包含名称和行标号的输入文件形式

Price	Floor	Area	Rooms	Age	Cent.heat
52.00	111.0	830	5	6.2	no
54.75	128.0	710	5	7.5	no
57.50	101.0	1000	5	4.2	no
57.50	131.0	690	6	8.8	no
59.75	93.0	900	5	1.9	yes
...					

此时，读取数据帧可使用：

```
> HousePrice <- read.table("houses.data", header=TRUE)
```

选项header=TRUE说明文件的第一行是表头行，由此可知文件的格式中不包括行标号。

7.2 函数scan()

假设需要平行读入等长的数据向量，进一步假定我们有3个向量，第一个的模式为字符型，其他两个为数值型，文件为'input.dat'，首先，使用scan()函数将三个向量以列表形式读入

```
> inp <- scan("input.dat", list("",0,0))
```

其中第二个参数是一个名义列表结构，用来确定要读取的三个向量的模式，存储于inp中的结果，是由三个读入的向量作为组件的列表。若要将其中的数据项分解为三个独立向量。可以通过下面的赋值操作

```
> label <- inp[[1]]; x <- inp[[2]]; y <- inp[[3]]
```

而且在名义列表中，我们可以直接命名组件，是存取已读入向量的过程更加方便。例如

```
> inp <- scan("input.dat", list(id="", x=0, y=0))
```

如果希望分别存取其中的变量，你既可以将变量重新赋值到工作帧中

```
> label <- in$id; x <- in$x; y <- in$y
```

也可以将列表附加到搜索路径的位置2上。（chapter 6）如果第二个参数是单个值而不是列表，那么函数只读入单个向量，而且所有组件都应当于名义变量保持相同的模式。

```
> X <- matrix(scan("light.dat", 0), ncol=5, byrow=TRUE)
```

R也拥有更复杂的输入功能，详细资料可以在提供的手册中查到。

7.3 内建数据集的存取

R本身提供超过50个数据集，同时在功能包（包括标准功能包）中附带更多的数据集。与S-Plus不同，这些数据即必须通过data函数载入。我们可以通过

```
data()
```

获得基本系统提供的数据集列表，然后通过形如

```
data(infert)
```

来载入名为infert的数据集。

多数情况下会有一个具有相同名称的R对象被载入，一般是一个数据帧。不过，少数情况下会由若干个对象被载入。此时最好察看关于这个对象的在线帮助，以确定可能发生的情况。

7.3.1 从其他R功能包中载入数据

要存取其他功能包中的数据，可以使用package参数，例如

```
data(package="nls")
data(Puromycin, package="nls")
```

如果某个功能包已经由函数library()挂接了，那么它的数据集也自动的包含在了搜索路径中。所以

```
library(nls)
data()
data(Puromycin)
```

将列出当前已挂接的所有数据集（至少包含**base**和**nls**）。并且从第一个能够找到Puromycin的功能包中载入这个数据集。

用户发布的功能包是一个丰富的数据集来源。

7.4 编辑数据

在使用一个数据帧或矩阵时，edit提供一个独立的工作表式编辑环境。对已读入的数据集做小改动，它还是很有用的。命令

```
> xnew <- edit(xold)
```

允许你对数据集xold进行编辑。并在完成时将改动后的对象赋值给xnew

使用

```
> xnew <- edit(data.frame())
```

可以通过工作表介面录入新数据。

Chapter 8

概率分布

8.1 R—作为一个统计表的集合

R一个很方便的用处是提供了一套完整的统计表集合。函数可以对累积分布函数$P(X \leq x)$，概率密度函数，分位函数（对给定的q，求满足$P(X \leq x) > q$的最小x）求值，并根据分布进行模拟。

Distribution	R name	additional arguments
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
chi-squared	chisq	df, ncp
exponential	exp	rate
F	f	df1, df1, ncp
gamma	gamma	shape, scale
geometric	geom	prob
hypergeometric	hyper	m, n, k
log-normal	lnorm	meanlog, sdlog
logistic	logis	location, scale
negative	binomial	nbinom size, prob
normal	norm	mean, sd
Poisson	pois	lambda
Student’s	t	t df, ncp
uniform	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

对于所给的名称，加前缀’d’代表密度(density)，’p’代表CDF，’q’代表分位函数，’r’代表模拟（随即散布）。这几类函数的第一个参数是有规律的，形为`dxxx`的函数为`x`，`pxxx`的函数为`q`，`qxxx`的函数为`p`，`rxxx`的函数为`n`（rhyper和rwilcox是特例，他们的第一个参数为nn）。目前为止，非中心参数(non-centrality parameter)仅对CDF和少数几个其他函数有效，细节请参考在线帮助。

所有pxxx和qxxx的函数都具有逻辑参数`lower.tail`和`log.p`，而所有的`dxxx`函数都有参数`log`，这个是我们可以直接通过

```
- pxxx(t, ..., lower.tail = FALSE, log.p = TRUE)
```

获取，比如说，累积失效函数（cumulative/integrated hazard function），$H(t) =$

$-\log(1-F(t))$，以及更精确的对数似然（通过`dxxx(..., log = TRUE)`）。

此外，对于来自正态分布，具有学生化样本区间的分布还有ptukey和qtukey这样的函数。下面是一些例子

```
> ## 2-tailed p-value for t distribution
> 2*pt(-2.43, df = 13)
> ## upper 1% point for an F(2, 7) distribution
> qf(0.99, 2, 7)
```

8.2 检测数据集合的分布

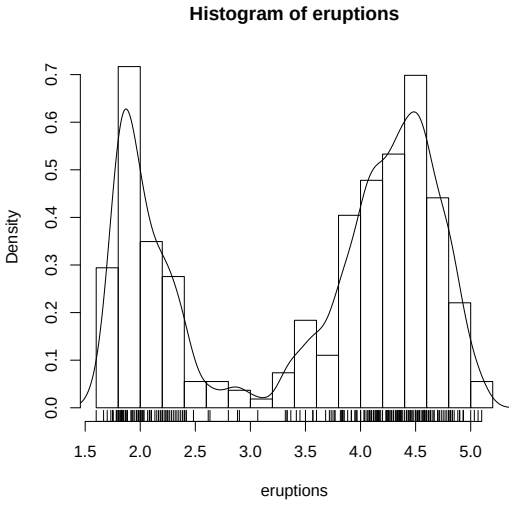
对于给定的（单变量）数据集合我们有许多方法检测其分布情况。最简单的方法是检测数值。函数`summary`和`fivenum`这两个函数可以分别给出两个有轻微差异的摘要，函数`stem`可以将数值显示出来（茎叶图"stem and leaf" plot）

```
> data(faithful)
> attach(faithful)
> summary(eruptions)
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.600 2.163 4.000 3.488 4.454 5.100
> fivenum(eruptions)
[1] 1.6000 2.1585 4.0000 4.4585 5.1000
> stem(eruptions)
The decimal point is 1 digit(s) to the left of the |
16 | 070355555588
18 | 000022233333335577777777888822335777888
20 | 00002223378800035778
22 | 0002335578023578
24 | 00228
26 | 23
28 | 080
30 | 7
32 | 2337
34 | 250077
36 | 0000823577
38 | 2333335582225577
40 | 000000335778888800223355557777 8
42 | 033355557788002333335555777 8
44 | 022223355577800000000233333577788 88
46 | 000023335770000002357 8
48 | 00000022335800333
50 | 0370
```

茎叶图看起来有些像直方图，R中绘制直方图的函数为hist。

```
> hist(eruptions)
## make the bins smaller, make a plot of density
> hist(eruptions, seq(1.6, 5.2, 0.2), prob=TRUE)
> lines(density(eruptions, bw=0.1))
> rug(eruptions) # show the actual data points
```

通过命令`density`我们可以获得更漂亮的密度图。在这个例子中我们还通过density绘制了一条密度曲线。由于默认方式平滑的有些过度（通常在“有趣”的密度中[1]），带宽`bw`是通过试错(trial-and-error)的方式确定的（在**MASS**和**KernSmooth**功能包中提供了自动选择带宽的方法和实现工具）。

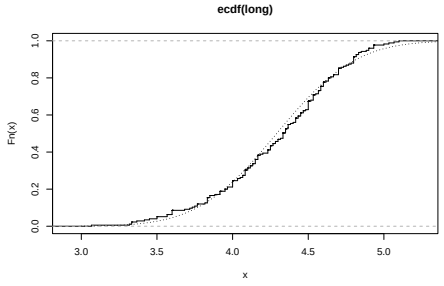


我们可以通过标准功能包**stepfun**中的函数`ecdf`绘制经验累积分布函数。

```
> library(stepfun)
> plot(ecdf(eruptions), do.points=FALSE, verticals=TRUE)
```

显然，这个分布与任何标准分布相距甚远。那么右边的情况怎么样呢，也就是eruption大于3的部分。我们拟和一个正态分布，覆盖拟合CDF。

```
> long <- eruptions[eruptions > 3]
> plot(ecdf(long), do.points=FALSE, verticals=TRUE)
> x <- seq(3, 5.4, 0.01)
> lines(x, pnorm(x, mean=mean(long), sd=sqrt(var(long))), lty=3)
```

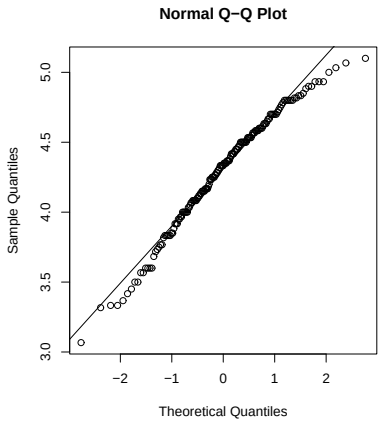


Quantile-quantile (Q-Q) 图有助于我们更细致的检测其分布形态

[1]原文括号中是it usually does for “interesting” densities

```
par(pty="s")
qqnorm(long); qqline(long)
```

可以看出这个分布与一个正态分布基本相符，但是右尾稍短，我们可以将它与t分布产生的模拟数据进行比较




```
x <- rt(250, df = 5)
qqnorm(x); qqline(x)
```

可以看到，这个随机样本的尾部要比正态数据所预期的长。我们可以通过

```
qqplot(qt(ppoints(250), df=5), x, xlab="Q-Q plot for t dsn")
qqline(x)
```

产生分布并绘制Q-Q图。

最后，我们可能希望一种更正式的检验来判断数据的正态性。功能包ctest中提供了Shapiro-Wilk检验：

```
> library(ctest)
> shapiro.test(long)

        Shapiro-Wilk normality test

data:  long
W = 0.9793, p-value = 0.01052
```

和Kolmogorov-Smirnov检验：

```
> ks.test(long, "pnorm", mean=mean(long), sd=sqrt(var(long)))

        One-sample Kolmogorov-Smirnov test

data:  long
D = 0.0661, p-value = 0.4284
alternative hypothesis: two.sided
```

（请注意，由于我们从相同样本中估计正态分布的参数，所以分布理论在此是无效的。）

8.3 单样本和两样本检验

目前我们已经将一个样本与一个正态分布进行了比较。一个更一般的操作是对两个样本的各方面进行比较。请注意，在R中，所有“经典”检验，包括我们在下面所用到的，都包含在功能包ctest中，为了使这些检验可用，有时需要通过library(ctest)从外部载入这个功能包。

考虑下面关于冰熔解放热(cal/gm)的数据，数据来自Rice(1995,P.490)

```
Method A: 79.98 80.04 80.02 80.04 80.03 80.03 80.04 79.97
          80.05 80.03 80.02 80.00 80.02
Method B: 80.02 79.94 79.98 79.97 79.97 80.03 79.95 79.97
```

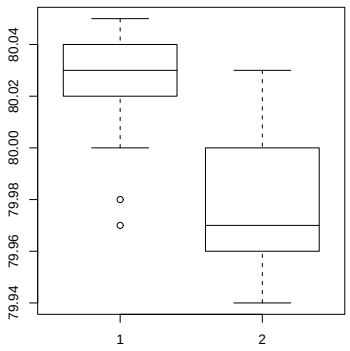
箱线图（盒子图box-plot）为两个样本提供了一个简单的图形比较方式：

```
A <- scan()
79.98 80.04 80.02 80.04 80.03 80.03 80.04 79.97
80.05 80.03 80.02 80.00 80.02

B <- scan()
80.02 79.94 79.98 79.97 79.97 80.03 79.95 79.97

boxplot(A, B)
```

这表明第一组的结果似乎高于第二组



要检验两样本的均值是否相等。我们可以使用不配对的t-检验

```
> t.test(A, B)

        Welch Two Sample t-test

data:  A and B
t = 3.2499, df = 12.027, p-value = 0.00694
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
```

```
 0.01385526 0.07018320
sample estimates:
mean of x mean of y
 80.02077  79.97875
```

在正态假设下，结果显示出显著的差异，在默认的情况下，R的函数并不假定两个样本是等方差的（与S-Plus中的t.test函数相反）我们可以再正态总体的假定下，使用F检验来判断样本方差是否相等

```
> var.test(A, B)

        F test to compare two variances

data:  A and B
F = 0.5837, num df = 12, denom df = 7, p-value = 0.3938
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1251097 2.1052687
sample estimates:
ratio of variances
         0.5837405
```

结果显示，没有充足的证据说明方差有显著的差异。所以我们可以使用假定方差相等的经典t-检验。

```
> t.test(A, B, var.equal=TRUE)

        Two Sample t-test

data:  A and B
t = 3.4722, df = 19, p-value = 0.002551
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.01669058 0.06734788
sample estimates:
mean of x mean of y
 80.02077  79.97875
```

所有的这些检验都基于两个样本的正态性。两样本的Wilcoxon（或Mann-Whitney）检验在零假设(null hypothesis)下仅仅假定分布是连续的。

```
> wilcox.test(A, B)

        Wilcoxon rank sum test with continuity correction

data:  A and B
W = 89, p-value = 0.007497
alternative hypothesis: true mu is not equal to 0
```

```
Warning message:
Cannot compute exact p-value with ties in: wilcox.test.default(A, B)
```

请注意警告信息，在样本中存在打结的现象，这意味着数据来自一个离散分布（可能由于截尾造成）。

通过图形方式比较两个样本由几种方法。我们已经见过一组箱线图的方式，下面的：

```
> library(stepfun)
> plot(ecdf(A), do.points=FALSE, verticals=TRUE, xlim=range(A, B))
> plot(ecdf(B), do.points=FALSE, verticals=TRUE, add=TRUE)
```

将显示两个经验CDF，而qqplot将显示一个两样本的Q-Q图。

Kolmogorov-Smirnov检验在一般连续分布的假设下计算两个ecdf之间垂直距离的最大值

```
> ks.test(A, B)

        Two-sample Kolmogorov-Smirnov test

data:  A and B
D = 0.5962, p-value = 0.05919
alternative hypothesis: two.sided

Warning message:
cannot compute correct p-values with ties in: ks.test(A, B)
```

Chapter 9

语句组、循环和条件操作

9.1 表达式语句组

R是一种表达式语言，也就是说其命令类型只有函数或表达式，并由它们返回一个结果。甚至，赋值也是一个表达式，只不过其结果是一个被赋值的值，只要可以使用表达式的地方都可以使用赋值；包括多重赋值。

命令可以通过花括号来分组，*expr_1; ...; expr_m*，此时这组语句的结果是组中最后一个能返回值的语句的结果。因此，这样一组语句也可以作为一个表达式，比如将其自身放在花括号中，作为一个更大的表达式的组成部分等等。

9.2 控制语句

9.2.1 条件执行：if语句

R语言中条件结构的形式为

```
> if (expr_1) expr_2 else expr_3
```

其中*expr_1*必须返回一个逻辑值，之后整个表达式的结果就是显而易见的了。

操作符`&&`和`||`经常被用于`if`语句的条件部分中，分别代表逻辑与和逻辑或。而`&`和`|`与`&&`，`||`的区别在于，`&`和`|`按照逐个元素的方式进行计算，`&&`和`||`对向量的第一个元素进行运算，只有在必需的时候才对第二个参数求值。

`if/else`结构的向量版本是函数`ifelse`，其形式为`ifelse (condition,a,b)`，产生函数结果的规则是：如果`condition[i]`为真，对应`a[i]`元素；反之对应的是`b[i]`元素。根据这个原则函数返回一个由a,b中相应元素组成的向量，向量长度与其最长的参数等长。

9.2.2 重复执行：for 循环，repeat 和while

loop循环结构的形式为

```
> for (name in expr_1) expr_2
```

其中*name*是循环变量，*expr_1*是一个向量表达式（通常是`1:20`这样的序列），而*expr_2*经常是一个表达式语句组，其中的子表达式仅仅以形式名称（dummy name）进行操作，和语句组外表达式的名称无关。*expr_2*随着*name*依次取*expr_1*结果向量的值而被多次重复运行。

作为一个示例，假定ind是个类别指示器向量，我们需要对x中的不同类别分别作y对x的图。可选的方法之一是使用coplot()，这个函数将产生一个图表的数组，对应因子的各个水平，我们会在后面讨论它。另一种选择是通过下面的方式将所有图表一次全部显示出来：

```
> xc <- split(x, ind)
> yc <- split(y, ind)
> for (i in 1:length(yc)) {
    plot(xc[[i]], yc[[i]]);
    abline(lsfit(xc[[i]], yc[[i]]))
}
```

（请注意函数split()将一个较大向量按照指定的类别分割成若干向量，并产生一个这些向量的列表。这个函数通常被用于连接箱线图。详细资料请使用help功能查询。）

警告：for()循环在R代码中的使用次数远少于在其他的编译语言中的。在R中使用'完整对象'（whole object）的观点来编写代码会更快更清晰。

其他循环功能包括

> repeat (*expr*)

语句和

> while (*condition*) *expr*

语句。

break语句可以用来中断任何循环，可能是非正常的中断。而且这是中止repeat循环的唯一方式。

next语句可以中止一个特定的循环，跳至下一个。

大多数情况下控制语句的应用都是与函数相关的。这方面的内容将在chapter 10讨论，在那里会有更多的例子出现。

Chapter 10

编写自己的函数

正如我们之前并不正式的见到的，R语言允许用户创建模式为*function*的对象，这些被创建的对象是真正的R函数，以特定的内在形式存储，可以在其他表达式中使用等等。通过创建函数，R语言在能力，易用性和易读性上都获得了极大的提高。学习编写有效的函数是令R的应用更舒适，更有创造性的主要途径。

还需要强调的一点是，绝大多数作为R系统组成部分的函数，例如`mean()`, `var()`,`postscript()`等等，都是由R语言编写的，也就是说与客户编写的函数并没有实质的差别。

函数的定义需要通过下面这种形式的赋值

```
> name <- function(arg_1, arg_2, ...) expression
```

*expression*是一个R表达式（通常是表达式语句组），并使用参数arg_i来计算出一个数值，表达式的值就是函数的返回值。

函数调用的形式通常都是name(*expr_1,expr_2,...*)，并且在任何一个可以进行函数调用的地方都可以出现。

10.1 简单示例

作为第一个例子，我们考虑创建一个计算两样本t统计量的函数。

这显然是一个人工的实力，因为我们可以通过其他更简易的方法得到相同的结果。

函数的定义

```
> twosam <- function(y1, y2) {
n1 <- length(y1); n2 <- length(y2)
yb1 <- mean(y1); yb2 <- mean(y2)
s1 <- var(y1); s2 <- var(y2)
s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
tst <- (yb1 - yb2)/sqrt(s2*(1/n1 + 1/n2))
tst
}
```

函数定义后，要进行一个两样本*t*检验可以使用Z

```
> tstat <- twosam(data$male, data$female); tstat
```

第二个例子，我们可以考虑创建一个模拟Matlab中`backslash`的命令，即返回向量y在矩阵X列空间上的正交投影的系数（者通常被称为回归系数的最小二

乘估计）。一般我们会用qr()函数完成这个任务，不过有些时候直接使用这个函数并不稳妥，我们可以用下面这个简单的函数使它用起来更安全些

因此对于给定的n维向量，y和$n \times p$矩阵X，Xy被定义为$(X'X)^{-1}X'y$，其中$(X'X)^{-1}$ 是$(X'X)$的广义逆

```
> bslash <- function(X, y) {
X <- qr(X)
qr.coef(X, y)
}
```

当这个对象被创建后，它就可以和任何对象一样用于例如

```
> regcoeff <- bslash(Xmat, yvar)
```

这样的语句中，等等。

经典的R函数lsfit()也能很好的完成这项工作，甚至更多[1]。它用qr()和qr.coef()函数，通过一种看起来有些不合常理的方法进行计算。

因此，如果使用很频繁的话，将这部分功能分离到一个简单易用的函数中还是有价值的。或者，我们会希望将它变成一个矩阵的二元操作符，以便更方便的使用。

10.2 定义新的二元操作符

我们可以给bslash()函数一个不同的名称，例如这种形式

```
%anything%
```

它可以在表达式中当作一个二元操作符使用，而非函数形式。假定，我们将其中的字符定为!，函数将按照下面的形式被定义

```
> "%!%" <- function(X, y) { ... }
```

（请注意引号的使用），之后，我们就可以通过x%!%y的方式来使用函数了。（在这里没有选择反斜线(backslash)是因为在这份手册中它用来针对某些特定的问题）

矩阵乘法运算符，%*%，和矩阵外积运算符%o%也是这种定义方式的例子。

10.3 指定的参数和默认值

正如我们在section 2.3所提到的，如果被调用函数的参数按照"name = obj"的形式给出，那么参数的次序可以是任意的。而且，参数序列可以在开始依次序给出，而将指定名称的参数置于后面。

因此，如果由一个函数fun1被定义为

```
> fun1 <- function(data, data.frame, graph, limit) {
[function body omitted]
}
```

那么函数可以有多种等价的使用方法，例如

```
> ans <- fun1(d, df, TRUE, 20)
> ans <- fun1(d, df, graph=TRUE, limit=20)
> ans <- fun1(data=d, limit=20, graph=TRUE, data.frame=df)
```

[1]参见chapter11种描述的方法

在很多情况下，参数都会被赋予一个普遍适用的默认值。有时，如果某些参数的默认值全是适用的，我们就可以把这些参数全部忽略。比如，令fun1被定义为

```
> fun1 <- function(data, data.frame, graph=TRUE, limit=20) { ... }
```

则函数可以这样被调用

```
> ans <- fun1(d, df)
```

此时与上面三种情况等价，也可以这样调用

```
> ans <- fun1(d, df, limit=10)
```

此时改变了一个默认值。

特别要注意的是参数可以是任意表达式，甚至可以是包含函数中其他参数的表达式；并不一定要像这里的例子一样是常量。

10.4 参数'...'

另外一个频繁出现的需要是允许一个函数将它的参数传递给另一个函数。例如，很多图形函数都会用到par()，而且plot()之类的函数允许用户通过par()来控制图形输出（参见）。可以通过包含一个额外的参数"..."来完成这个目的，将参数传递到另一个函数中。下面是一个简要的示例

```
fun1 <- function(data, data.frame, graph=TRUE, limit=20, ...) {
[omitted statements]

   if (graph)
   par(pch="*", ...)

[more omissions]
}
```

10.5 函数内的赋值

请注意，*在函数内完成的任何一般赋值操作都是局部的(local)和临时的(temporary)，当退出函数后就丢失了。* 因此X<-qr(x)这个赋值操作对被调用程序中的参数只是没有影响的。

如果要对管理R赋值范围的规则有一个彻底的了解，读者需要熟悉一下求值体系的理念。尽管这并不难，这仍是一个较深入的话题，在这里不会进行深入的讨论。

如果在函数内要进行全局(global)和永久(permanent)的赋值，那需要使用到“超赋值”操作符, <<- 或者函数assign()。详细内容参见帮助文件。S-PLUS用户请注意，<<- 在R中有不同的意思。这些在10.7节中有更深入的讨论。

10.6 更多高级示例

10.6.1 区组设计的效率因子(Efficiency factors)

作为一个更完整，不过有些乏味的函数的示例，考虑求解一个区组设计的效率因子（这问题的某些方面已经在sect 中讨论过）。

一个区组设计由两个因子定义，blocks(b levels)和varieties(v levels)。如果R与K分别是$v \times v$的复制矩阵(replications matrix)和$b \times b$的区组容量矩阵(block size matrix)，而N是$b \times b$的发生率矩阵(incidence matrix)，那么效率因子可以由矩阵的特征值定义

$E = I_v R^{1/2} N' K^{-1} N R^{1/2} = I_v A' A$
其中$A = K^{1/2} N R^{-1/2}$。下面给出了函数的一种编写方法。

```
> bdeff <- function(blocks, varieties) {
blocks <- as.factor(blocks) # minor safety move
b <- length(levels(blocks))
varieties <- as.factor(varieties) # minor safety move
v <- length(levels(varieties))
K <- as.vector(table(blocks)) # remove dim attr
R <- as.vector(table(varieties)) # remove dim attr
N <- table(blocks, varieties)
A <- 1/sqrt(K) * N * rep(1/sqrt(R), rep(b, v))
sv <- svd(A)
list(eff=1 - sv$d^2, blockcv=sv$u, varietycv=sv$v)
}
```

从计算的角度来说，这个例子中使用奇异值分解要比特征值的方法好。
函数的结果是一个列表。不仅在第一个组件中包含了效率因子，还给出了区组和变异的标准对比(block and variety canonical contrasts)，因为有时这些数值会给出一些有用的定性信息。

10.6.2 删除打引数组中的所有名称

一些大型的数组和矩阵，出于打印方面的目的，经常需要按照紧凑的区块形式，省略数组名称和编号打印。删除dimnames属性并不能达到这个效果，而应当令数组的dimnames属性全是空字串。例如打印一个矩阵X

```
> temp <- X
> dimnames(temp) <- list(rep("", nrow(X)), rep("", ncol(X))
> temp; rm(temp)
```

通过一个函数no.dimnames()来完成这个工作会方便些，而且正如下面所展示的，函数提供了一种整体化的"wrap around"的解决方法。这也说明了为什么某些很有效的用户函数可以相当短小。

```
no.dimnames <- function(a) {
## Remove all dimension names from an array for compact printing.
d <- list()
l <- 0
for(i in dim(a)) {
d[[l <- l + 1]] <- rep("", i)
}
dimnames(a) <- d
a
}
```

当这个函数被定义之后，一个数组就可以通过下面的命令按照紧凑格式打印了。

```
> no.dimnames(X)
```

当形态比数值更受关注时，对于大型的整数数组，这是个相当有用的函数。

10.6.3 递归的数值积分

函数可以是递归的，而且可以在函数内部再定义函数。不过需要注意的事，这类函数或变量是不会传递给调用它的更高层求值结构，除非它们在搜索路径中。

下面的例子—一维数值积分一种很初级的求法。被积函数在区间的末端和中点被求值，如果（one panel trapizum rule）的结果和tow panel trapizum的结果足够接近，就返回后者的值，否则就将这个过程应用于每个panel上。函数的结果是一个可适应的积分过程。这个过程中，被积函数在与一次曲线最远的区域被求值。不过这个过程的消耗太大，仅仅在被积函数光滑而且难以求值的时候才能勉强与其他算法的效率相比。

下面这个例子也可以当作R编程的一个小智力测验了

```
area <- function(f, a, b, eps = 1.0e-06, lim = 10) {
fun1 <- function(f, a, b, fa, fb, a0, eps, lim, fun) {
## function 'fun1' is only visible inside 'area'
d <- (a + b)/2
h <- (b - a)/4
fd <- f(d)
a1 <- h * (fa + fd)
a2 <- h * (fd + fb)
if(abs(a0 - a1 - a2) < eps || lim == 0)
return(a1 + a2)
else {
return(fun(f, a, d, fa, fd, a1, eps, lim - 1, fun) +
fun(f, d, b, fd, fb, a2, eps, lim - 1, fun))
}
}
fa <- f(a)
fb <- f(b)
a0 <- ((fa + fb) * (b - a))/2
fun1(f, a, b, fa, fb, a0, eps, lim, fun1)
}
```

10.7 范畴(scope)

在这节进行的讨论与这份文档的其他部分相比要更偏技术一些。不过这部分对S-PLUS和R之间的一处主要区别进行了详细的描述。

函数主体内出现的标识(SYMBOL)可以被分为3类；正式参数、局部变量和自由变量。函数的正式参数就是出现在函数参数列表中的参数，他们的值由实际的函数参数与正式参数的绑定（BINDING）过程决定。局部变量是在参数主体中，由表达式求值过程决定的变量。既不是正式参数又不是局部变量的变量被称为自由变量。赋值之后自由变量成为局部变量。请观察下面的函数定义

```
f <- function(x) {
y <- 2*x
print(x)
print(y)
print(z)
}
```

这个函数中，x是正式参数，y是局部变量，z是自由变量。

在R中，自由变量的值由函数被创建的环境中与其同名的第一个变量值决定，这种方式被称为词汇式范畴(LEXICAL SCOPE)，首先我们定义一个名为cube的函数。

```
cube <- function(n) {
sq <- function() n*n
n*sq()
}
```

函数sq中的变量n并不是这个函数的参数，因此它是一个自由变量，需要使用范畴规则来确定这个变量将与哪个值绑定。在静态范畴(STATIC SCOPE)（S-PLUS使用的规则）下，这个值将与一个名为n的全局变量绑定。在词汇式范畴(R) 下，由于函数被sq定义的时候，n与函数cube参数的绑定是被激活的，所以n对应的就使函数sq的参数。R与S-PLUS求值的区别就在于S-PLUS寻找一个名为n的全局变量，R首先寻找函数cube被调用时所创建的环境中一个名为n的变量。

```
## first evaluation in S
S> cube(2)
Error in sq(): Object "n" not found
Dumped
S> n <- 3
S> cube(2)
[1] 18
## then the same function evaluated in R
R> cube(2)
[1] 8
```

词汇式模式可以用来赋予函数*可变状态*(MUTABLE STATE)。在下面的例子中我们会展示如何将R模拟成一个银行账户。一个函数式的银行账户应当有余额和总计，进行存、取款的函数和显示当前余额的函数。要完成这个目标，我们可以在account下创建三个函数，然后返回包含着三个函数的一个列表，当account被调用时，它需要一个数值型的参数total，然后返回包含那三个函数的一个列表，因为这些函数是在包含total的环境中被定义的，它们可以直接存取total的值。

而特殊的操作符<<-可以用来改变与total绑定的值，这个操作符返回周围的环境，寻找一个包含标识total的环境。当找到这个环境时用操作符右侧的值替换环境中的值[2]。如果在更高级环境或全局中都没有找到标识total的话，R就会创建一个变量，并在其所在环境中对它赋值。对绝大多数用户来说<<- 将创建一个全局变量，并将右侧的值赋给这个变量。只有当<<-被用于一个函数中，并作为令一个函数的值被返回的时候，我们在这里所描述的特性才会发生

```
open.account <- function(total) {
list(
deposit = function(amount) {
if(amount <= 0)
stop("Deposits must be positive!\n")
```

[2]从某种意义上讲，这模拟了S-PLUS的方式，因为在S-PLUS中这个操作符通常创建或者赋值到一个全局变量

```
total <<- total + amount
cat(amount, "deposited. Your balance is", total, "\n\n")
},
withdraw = function(amount) {
if(amount > total)
stop("You don't have that much money!\n")
total <<- total - amount
cat(amount, "withdrawn. Your balance is", total, "\n\n")
},
balance = function() {
cat("Your balance is", total, "\n\n")
}
)
}
ross <- open.account(100)
robert <- open.account(200)
ross$withdraw(30)
ross$balance()
robert$balance()
ross$deposit(50)
ross$balance()
ross$withdraw(500)
```

10.8 定制环境

用户可以通过几种不同的方式定义它们的环境。R包含一个基础初始文件(SITE INITIALIZATION FILE)，而且每个目录下可以有它们自己特定的初始文件。最后还可以使用特殊的函数.First和.Last。

基础初始文件的位置是由R_PROFILE环境变量的之所决定的。如果这个变量没有被设定。那么R根目录下的子目录'etc'将被使用。这个文件中包含了你希望每次R启动时都会被执行的命令。此外，一个名为'.Rprofile'[3] 的个人配置文件可以被置于任意一个目录下。如果R在该目录下被调用，这个配置文件就会被使用。这个文件是每个用户可以对他们的工作区进行控制，并允许不同工作目录下不同的启动进程。如果启动目录下没有找到'.Rprofile'文件，那么R就在用户的根目录下搜索'.Rprofile'，并使用这个文件（如果这个文件存在的话）。

在两个配置文件之中或者'.RData'映像中，任何名为.First()的函数都具有特殊的意义。它将在一个R任务的开始被自动执行，可以被用来初始化并为任务的其他部分设定很多有用的东西。

被执行文件序列是'Rprofile.site', '.Rprofile','.RData'，之后是.First()。后面文件的定义会覆盖前面文件中的定义。

```
> .First <- function() {
options(prompt="$ ", continue="+\t") # $ is the prompt
options(digits=5, length=999) # custom numbers and printout
x11() # for graphics
par(pch = "+") # plotting character
source(file.path(Sys.getenv("HOME"), "R", "mystuff.R"))
```

[3]在UNIX下是隐藏的

```
# my personal package
library(stepfun) # attach the step function tools
}
```

相似的，如果定义函数.Last()，它将在每个任务的结尾被执行。下面是一个例子

```
> .Last <- function() {
graphics.off() # a small safety measure.
cat(paste(date(),"\nAdios\n")) # Is it time for lunch?
}
```

10.9 类别，通用函数和对象定位

一个对象的类别(CLASS)决定了他会如何被通用函数(GENERIC FUNCTION)处理。通用函数对参数执行的任务或动作取决于其参数自身的类别。如果参数本身没有任何类别属性，或者其类别在特定问题中并不满足通用函数的要求，通常会有一个默认的动作被执行。

例子会令问题更清晰。类别机制使用户可以为特定的目的设计和编写通用函数。通用函数中包括以图形方式显示对象的plot()，对各种分析进行摘要的summary(),比较统计模型的anova()等等。

可以对一种类别进行特定处理的通用函数是非常多的。例如，可以对''data.frame''类型的对象进行操作的函数包括：

```
[    [[<-   any    as.matrix
[<-  model  plot   summary
```

使用method()函数可以获得相应的完整列表

```
> methods(class="data.frame")
```

而一个通用函数所能处理的类别也是非常多的。例如，对于''data.frame''
,''density'',''factor''— 或其他类别的对象，plot()函数拥有一个默认的方式和多种变化。函数method()可以提供一个完整的列表

```
> methods(plot)
```

对这种机制的完整讨论，读者可以参考各种官方资料。

Chapter 11

R的统计模型

这部分中我们假定读者对统计方法已经有了一些了解，特别是回归分析和方差分析部分。之后我们会有更深入的一些假定，例如广义线性模型和非线性回归。

拟合统计模型所需要的条件已经被很好的定义了，这使得我们可以对广泛的一系列的问题构建一般性的通用的工具。

R提供了一套连锁的功能，使统计模型的拟合非常简单。正如我们在绪论中提到的，基本的输出是最精简的，用户可以通过调用释放函数来寻求更详细的资料。

11.1 定义统计模型；公式

一个统计模型的基本模版是一个独立等方差的线性回归模型

$y_i = \sum_{j=0}^{p} \beta_j x_{ij} + e_i, \qquad e_i \sim NID(0, \sigma^2), \qquad i = 1, \ldots, n$

矩阵形式可以写作

$y = X\beta + e$

其中y是响应向量，X是模型矩阵或设计矩阵，矩阵的列$x_0, x_1, \ldots, x_p$是决定变量。x_0经常是一列1，用来定义一个截距项。

Examples

在给出正式定义之前，几个例子有助于我们建立印象。

假定`y,x,x0,x1,x2,...`是数值变量，`X`是一个矩阵，`A,B,C,...`是因子。下面左侧的公式定义了右侧描述的模型。

`y~x`

`y~1+x`

二者指定的模型是相同的，即y对x的简单线性模型。第一个公式的截距项是隐含的。第二个的截距项是明确给定的。

`y~0+x`

`y~-1+x`

`y~x-1`

y对x过原点的简单线性回归模型（无截距项）

`log(y)~x1+x2`

进行变换了的因变量log(y)对x1和x2的多重回归（包括一个隐含的截距项）

`y~poly(x,2)`

`y~1+x+I(x^2)`

y对x的二次多项式回归。第一种形式使用正交多项式。

`y~X+poly(x,2)`

y的多重回归。模型矩阵包含矩阵X和x的二次多项式项

`y~A`

y的单因素方差分析模型，类别由A决定。

`y~A+x`

y的单因素协方差分析模型，类别由A决定，协变量为x。

`y~A*B`

`y~A+B+A:B`

`y~B%in%A`

`y~A/B`

y对A和B的两因素非附加(non-additive)模型。前两个指定了相同的交叉分类(crossed classification)，后面的两个指定了相同的嵌套分类(nested classification)。抽象一点儿讲，这四个公式指定了相同的模型子空间。

`y~(A+B+C)^2`

`y~A*B*C-A:B:C`

三因素试验，模型包含主效应和两因素之间的交互作用。两个公式指定的是相同的模型。

`y~A*x`

`y~A/x`

`y~A/(1+x)-1`

y对x在A水平单独的简单线性回归模型，几种形式的编码不同。最后一种形式产生与A中水平数一样多的截距和斜率估计值。

`y~A*B+Error(C)`

试验包含两个处理因子A、B以及由因子C决定的误差层(error strata)。比如一个分组试验中，整个组（当然包括子分组），都由因子C决定。

在R中，由操作符~定义一个模型公式，对一个一般的线性模型，其形式为

response ~ op_1 term_1 op_2 term_2 op_3 term_3 ...

其中

response 是一个矩阵或向量（或一个值为矩阵、向量的表达式），由它定义响应变量。

op_i 是一个操作符，+或−中的一个，指定模型中的某项是被包含还是被排除，（公式第一项可以省略操作符）。

term_i 可以是

- 一个向量或矩阵表达式，或1，
- 一个因子，或
- 一个由因子，向量，矩阵组成，由公式操作符连接的公式表达式。

在任何情况下，公式中每项都定义了一个列的组合，这个组合或者加入或者被移出模型矩阵。而1代表一个截距项的列，除非指明删除，否则这列默认包含在模型矩阵中。

公式*操作符*的作用与Wilkinson和Rogers在Glim 和Genstat中所使用的符号效果相似。不过由于句号是R的名称字符，所以不可避免的将'.'换作':'。

下面是这些符号的摘要（基于C&H 1992,p29）

Y~M

Y按照M建模

M_1+M_2

包括M_1和M_2。

M_1-M_2

包含M_1，不包含M_2。

M_1:M_2

M_1和M_2的张量积。如果两项都是因子，那么是“子类别”(subclasses)因子。

M_1 %in% M_2

与M_1和M_2相似，只是编码不同。

M_1 * M_2

M_1+M_2+M_1:M_2

M_1/M_2

M_1 + M_2 %in% M_1

M^n

M中的所有项以及其n阶交互作用。

I(M)

隔离M。这项出现在模型矩阵中，M中的所有操作符仍有它们正常的算术意义。

请注意括号通常都用于隔离函数参数，使所有操作符都具有正常的算术意义。函数`I()`是一个恒等式函数，一般仅用来允许使用算术操作符定义模型公式中的项。

特别要注意的是，模型公式指定的是*模型矩阵的列*，并且暗含对参数的指定。在其他情况下情况有所不同，比如用来指定非线性模型。

11.1.1 对比(contrasts)

对于模型公式如何指定模型矩阵的列，我们至少还要有些概念。如果我们拥有连续变量，事情就比较简单。每个变量就向模型矩阵提供一列（如果模型中包含截距，它将向模型矩阵提供一列1）。

但是对于一个k水平的因子A，答案就会因为因子是有序的还是无序的而有差别。对无序因子，对应因子的第2,...,k水平将生成$k-1$列（因此默认的参数化也就是将各个水平的响应值和初值进行对比）。而对有序因子，$k-1$则是对1,...,k的正交多项式，忽略常数项。

尽管这个答案已经挺复杂了，不过还不是全部。首先，如果在一个含有因子项的模型中省略常数项，那么第一个这样的项就会被编码为k列，对应所有的水平。此外，通过更改contrasts的选项(options)，可以改变处理的方式。在R中默认的设置是

```
options(contrasts = c("contr.treatment", "contr.poly"))
```

提到这点主要是因为R和S在处理无序因子的时候使用不同的默认设置，S使用Helmert Contrasts。因此，如果你需要将你用R得到的结果与使用S-PLUS的论文或教材得到的结果作比较的话，你需要设置

```
options(contrasts = c("contr.helmert", "contr.poly"))
```

这是一个谨慎的差别，因为TREATMENT CONTRAST（R的默认值）被认为对新手来说比较容易理解。

到这里仍然没有结束，因为通过函数contrasts和C还可以对模型中的每一项设定对比的方式。

而且我们还没有考虑交互项：由他们的组件引用，生成列的积。

尽管细节比较复杂，事实上R的模型公式一般都会生成一个统计专家所期望的那个公式，提供这种扩展是为了以防万一。比如拟合一个有交互作用但是没有对应的主效应的模型，一般结果会比较出人意料，这仅仅是为专家准备的。

11.2 线性模型

拟合普通多重模型的基本方程是lm()，调用方式为

```
> fitted.model <- lm(formula, data = data.frame)
```

例如

```
> fm2 <- lm(y ~ x1 + x2, data = production)
```

将拟合Y对X1和X2的多重回归模型（包含截距项）。

其中一个重要的不过技术上可选的参数data = production 指定构建模型所需的所有变量都来自数矩帧production。这种情况下并不在乎数据帧production是否已经挂接到搜索路径上。

11.3 用于释放模型信息的通用函数

函数lm()的值是一个拟和模型的对象；即类别为"lm"，由结果组成的一个列表。关于拟和模型的信息可以由那些适合"lm"类别对象的通用函数显示、释放和绘图。这些函数包括

```
add1    coef        effects   kappa   predict residuals
alias   deviance    family    labels  print   step
anova   drop1       formula   plot    proj    summary
```

下面是最常用几个函数的简要描述

anova(object 1, object 2) 将一个子模型与一个外部模型做比较，并生成一个方差分析表。

coefficients(object) 释放回归系数（矩阵）。缩略形式：COEF(OBJECT)。

deviance(object) 残差平方和，如果需要的话返回加权值。

formula(object) 释放模型公式。

plot(object) 生成四个图表，显示残差，拟合值和某些诊断。

predict(object, newdata=data.frame) 提供的新数据帧必须与原数据帧中的变量具有相同的标签。函数返回值是与DATA.FRAME中的决定变量对应的预测值向量或矩阵。

print(object) 打印对象的一个精简版本，经常暗含在其他操作中。

residuals(object) 释放残差（矩阵），如果需要的话返回加权值。缩略形式：RESID(OBJECT)。

step(object) 通过增减项来选择一个适当的模型，，并保持层级(HIERARCHIES)。在逐步搜索中，具有最大AIC值的模型将被返回。

summary(object) 打印回归分析结果的一个综合摘要。

11.4 方差分析与模型比较

模型拟合函数`aov(formula, data=`*data.frame*`)`最简单的操作与函数LM()和SECT11.3所列的大多数通用函数非常相似。

需要注意的是，此外AOV()允许进行多误差层(MULTIPLE ERROR STRATA)的模型分析，例如分组试验(SPLIT PLOT EXPERIMENTS)，包含组间信息的平衡不完全区组设计(BALANCED INCOMPLETE BLOCK DESIGNS WITH RECOVERY OF INTER-BLOCK INFORMATION)。模型公式

RESPONSE MEAN.FORMULA + ERROR(STRATA.FORMULA)

指定一个多层试验，误差层由*strata.formula*定义，在最简单的情况下，*strata.formula*只是一个因子，它定义了一个两层试验，即因子的水平内和水平间。

例如，已知全部决定变量因子，一个这样的模型公式：

```
> fm <- aov(yield ~ v + n*p*k + Error(farms/blocks), data=farm.data)
```

将用来描述均值模型为`v + n*p*k`，包含三个误差层的试验，这三个层分别是"农田间"，"农田内，区块间"，"区块间" ("BETWEEN FARMS", "WITHIN FARMS, BETWEEN BLOCKS" AND "WITHIN BLOCKS")。

11.4.1 方差分析表(ANOVA tables)

同时需要注意的是方差分析表也是拟合模型的一个结果。

11.5 更新拟合模型

函数UPDATE()允许在之前一个拟合模型的基础上仅仅增减几项来拟合一个新的模型。其形式为

```
> new.model <- update(old.model, new.formula)
```

在*new.formula*中含有句号'.'的特定标志，只能被用于代表"与旧模型公式对应的部分"。例如，

```
> fm05 <- lm(y ~ x1 + x2 + x3 + x4 + x5, data = production)
> fm6 <- update(fm05, . ~ . + x6)
> smf6 <- update(fm6, sqrt(.) ~ .)
```

分别拟合一个5个变元的多重回归，变量（假定）来自数据帧PRODUCTION；拟合另一个模型包含第六个回归元变量；拟合一个响应变量做了平方根变换的模型变量。

此外，特别要注意的是如果参数data=在原模型拟合函数调用中被指定的话，这个信息将随拟合模型对象被传递到update()和相关项目中。

标志'.'也可以用于其他情况，不过意义有些微小差别。例如

```
> fmfull <- lm(y ~ . , data = production)
```

将拟合一个模型，其中响应变量为y，*回归元变量为数据帧*production*中所有其它变量*。

用于探索模型增加结果的其他函数有add1(),drop1()和step()。这些函数的名称提示了它们的用途，不过完整的资料还是要参考在线帮助。

11.6 广义线性模型

广义线性模型是线性模型的发展，以便通过一种简明清晰的方式适应非正态响应分布合线性变换。描述一个广义线性模型应当基于以下的假设。

- 存在响应变量y，刺激变量$x_1, x_2, \ldots$,的值影响响应变量的分布
- 刺激变量，仅通过一个简单线性方程来影响y的分布。这个线性方程被称为线性预测元(LINEAR PREDICTOR)，通常写作$\eta = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$,

  因此当且仅当$\beta_i = 0$ 时，x_i对y的分布没有影响。
- y的分布形如$f_Y(y;\mu,\varphi) = \exp[\frac{A}{\varphi} y\lambda(\mu) - \gamma(\lambda(\mu)) + \tau(y,\varphi)]$ 其中φ 为*尺度参数*（可能已知），所有观测都是常量，A代表权重，并假定权是已知的，只是可能随观测的不同而改变，μ 是y 的均值。也就是说，假定了y的分布由它的均值或一个尺度参数来决定。
- 均值μ 是线性预测元的一个平滑可逆函数$\mu = m(\eta)$ $\eta = m^{-1}(\mu) = l(\mu)$ 而其中的反函数$l()$成为连接函数(LINK FUNCTION)。

这些假定对于包括一大类在统计实践中被广泛使用的模型是足够宽松的了，而对于发展一门统一的估计和预测方法又是足够严密的，至少是大致上的。读者可以参考这方面的任何一本参考书来获取详细的资料。

11.6.1 族(families)

R支持的广义线性模型类别包括高斯，二项，泊松，逆高斯和伽马响应分布以及响应分布无需明确指定的准似然(*quasi-likelihood*)分布，在后者中，方差函数必须作为均值的一个函数给定，而其他情况下，这个方程已经由响应分布确定了。

每种响应分布都包含若干种连接函数来连接均值函数和线性预测元。下表中给出了自动可用的连接函数。

广义线性模型的族包含一个响应分布，连接函数以及用于完成模型构建的其他各种信息。

11.6.2 函数glm()

由于响应分布仅仅由刺激变量(STIMULUS VARIABLES)通过一个线性函数决定，所以用于线性模型的相同机制可以被用于指定一个广义模型的线性部分。模型的族需要用另一种方式指定。

用于拟合一个广义线性模型的R函数为`glm()`，形为

```
> fitted.model <- glm(formula, family=family.generator, data=data.frame)
```

唯一的新特性是FAMILY.GENERATOR，其作用是描述模型的族，它其实是一个函数的名称，这个函数将产生一个函数与表达式列表用于定义和控制建模、估计过程。尽管刚看到时可能会显得有些复杂，其实使用很简单。

标准的、支持的族生成器(FAMILY GENERATORS)名称在SECT 11.6的表中"FAMILY NAME"下给出。当选定一个连接时，XXXXXXXXX

高斯分布族

如下的调用

```
> fm <- glm(y ~ x1 + x2, family = gaussian, data = sales)
```

与下面这个调用是等价的

```
> fm <- lm(y ~ x1+x2, data=sales)
```

但是效率要低得多。需要注意高斯分布族并没有自动的提供连接函数的选项，因此不用提供参数。如果一个问题要用高斯分布族和一个非标准的连接函数。那么通常都是通过`quasi`族来完成的。我们稍后可以看到。

二项分布族

让我们来考虑一个人工的小例子。（来自SILVEY）

在爱琴海的KALYTHOS岛上，男性居民受到一种先天眼病的困扰，而且年龄越大影响越重。对不同年龄的男性岛民样本做了失明的检测，并记录了结果。下面是获得的数据：

AGE	20	35	45	55	70
NO.TESTED	50	50	50	50	50
NO.BLIND	6	17	26	37	44

我们的问题是对这个数据拟合LOGISTIC和PROBIT模型，并对每个模型估计LD50，即男性居民失明概率为50的年龄。

如果y是x岁的失明人数，n是测试数，两个模型的形式都是

$y \sim B(n, F(\beta_0 + \beta_1 x))$ 其中，对PROBIT模型$F(z) = \Phi(z)$是标准正态分布，对LOGIT模型（默认值），$F(z) = e^z/(1 + e^z)$。两种情况下LD50都是

$$\text{LD50} = -\beta_0/\beta_1$$

也就是分布函数参数为零的点。
首先建立数据为数据帧

```
> kalythos <- data.frame(x = c(20,35,45,55,70), n = rep(50,5),
                         y = c(6,17,26,37,44))
```

使用GLM()拟合一个二项模型，有两种可能的响应变量：

- 如果响应变量是向量形式，那么它应当完全由0/1 的二元数据构成。
- 如果响应变量是由两列构成的矩阵，那么它应当在第一列保存试验成功数，第二列保存试验失败数。

这里我们使用第二条惯例，所以我们向数据帧中添加一个矩阵：

```
> kalythos$Ymat <- cbind(kalythos$y, kalythos$n - kalythos$y)
```

用下面的命令拟合模型

```
> fmp <- glm(Ymat ~ x, family = binomial(link=probit), data = kalythos)
> fml <- glm(Ymat ~ x, family = binomial, data = kalythos)
```

在第二个调用中，由于LOGIT连接函数是默认值，参数可被忽略。要察看每个拟合的结果可以使用

```
> summary(fmp)
> summary(fml)
```

两个模型都拟合得不错。要得到LD50的估计值，我们可以使用一个简单函数：

```
> ld50 <- function(b) -b[1]/b[2]
> ldp <- ld50(coef(fmp)); ldl <- ld50(coef(fml)); c(ldp, ldl)
```

从数据中得到的估计值分别是43.663岁和43.601岁。

泊松分布模型

泊松分布族默认的连接函数是LOG，在实际应用中，这个族主要用于拟合频数数据的替代对数线性模型(SURROGATE POISSON LOG-LINEAR MODELS)，因为这些数据的分布通常都是多项分布。这个主题的内容相当多，而且很重要，我们在这里就不深入讨论了。它甚至是构成非正态广义模型应用的主要部分。

泊松分布数据偶尔会出现在实际应用中，过去通常是将它进行对数或平方根变换后作为一个正态数据分析。作为平方根变换的一个不错的替代，可以向下面的例子展示的那样，拟合一个泊松广义线性模型：

```
> fmod <- glm(y ~ A + B + x, family = poisson(link=sqrt),
              data = worm.counts)
```

准似然模型(Quasi-likelihood models)

对所有的族，响应的方差都依赖于均值，并将尺度参数作为乘数。方差对均值依赖性的形式是响应分布的一个特性；例如泊松分布就是$Var[y] = \mu$。

对准似然估计和预测来说，准确的响应分布并未给出，而仅给出连接函数和方差函数对均值的依赖形式。由于准似然估计使用与正态分布形式相同的技术，因此这个族为拟合非标准的连接函数和方差函数提供了一种选择。

例如，考虑拟合非线性回归

$y = \frac{\theta_1 z_1}{z_2 - \theta_2} + e$

这个方程也可以写作

$y = \frac{1}{\beta_1 x_1 + \beta_2 x_2} + e$

其中，$x_1 = z_2/z_1$,$x_2 = -1/x_1$,$\beta_1 = 1/\theta_1$和$\beta_2 = \theta_2/\theta_1$。假设，存在一个合适的数据帧，我们可以这样拟合这个非线性回归

```
> nlfit <- glm(y ~ x1 + x2 - 1,
               family = quasi(link=inverse, variance=constant),
               data = biochem)
```

更深入的信息，读者可以参考手册或帮助文档。

11.7 非线性最小二乘和最大似然模型

特定形式的非线性模型可以由广义线性模型(Generalized Linear Models)(`glm()`)拟合，但是大多数情况下，我们需要通过非线性优化来解决非线性曲线拟合的问题。R中非线性优化通常使用`nlm()`，相当于S-Plus中的`ms()`和`nlmin()`。我们通过使某些拟合不足（lack-of-fit）指标最小化来寻找参数值，`nlm()`则通过迭代的方式试验不同的值。与线性回归不同。我们无法保证这个过程会收敛于令我们满意的估计上。`nlm()`需要一个初始猜测以便尝试参数值，而收敛性依赖于初始猜测的质量。

11.7.1 最小二乘

拟和非线性模型的一种方法使通过是参差平方和最小化。这种方法在观测误差来自正态分布时是有意义的。

下面是来自Bates & Watts (1988), page51. 数据是：

```
> x <- c(0.02, 0.02, 0.06, 0.06, 0.11, 0.11, 0.22, 0.22, 0.56, 0.56,
         1.10, 1.10)
> y <- c(76, 47, 97, 107, 123, 139, 159, 152, 191, 201, 207, 200)
```

要拟合的模型是

```
> fn <- function(p) sum((y - (p[1] * x)/(p[2] + x))^2)
```

要进行拟合，我们需要参数的初始估计，寻找初始值的一种方法是通过数据的图形，猜测某些参数值，并使用这些值添加模型曲线。

```
> plot(x, y)
> xfit <- seq(.02, 1.1, .05)
> yfit <- 200 * xfit/(0.1 + xfit)
> lines(spline(xfit, yfit))
```

我们可以做得更好些，不过初始值200和.1应该足够了。现在进行拟合：

```
> out <- nlm(fn, p = c(200, 0.1), hessian = TRUE)
```

拟合之后，out$minimum是SSE，out$estimates是参数的最小二乘估计。通过下面的操作可以得到大致的估计标准差：

```
> sqrt(diag(2*out$minimum/(length(y) - 2) * solve(out$hessian)))
```

上式的2代表参数的个数。参数估计值$+_1$.96SE就是一个95%的置信区间。我们可以在新的图上添加最小二乘的拟合：

```
> plot(x, y)
> xfit <- seq(.02, 1.1, .05)
> yfit <- 212.68384222 * xfit/(0.06412146 + xfit)
> lines(spline(xfit, yfit))
```

标准功能包**nls**提供了更多更广泛的功能来对非线性模型做最小二乘拟合。我们刚刚拟合的是Michaelis-Menten 模型，所以我们可以用

```
> df <- data.frame(x=x, y=y)
> fit <- nls(y ~ SSmicmen(x, Vm, K), df)
> fit
Nonlinear regression model
  model:  y ~ SSmicmen(x, Vm, K)
   data:  df
         Vm            K
212.68370749   0.06412123
 residual sum-of-squares:  1195.449
> summary(fit)

Formula: y ~ SSmicmen(x, Vm, K)

Parameters:
    Estimate Std. Error t value Pr(>|t|)
Vm 2.127e+02  6.947e+00  30.615 3.24e-11 ***
K  6.412e-02  8.281e-03   7.743 1.57e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 10.93 on 10 degrees of freedom

Correlation of Parameter Estimates:
      Vm
K 0.7651
```

11.7.2 最大似然

最大似然方法可以拟合非线性模型，即使误差不是正态的。这种方法通过对数似然最大化寻找参数值，或者等价的，使负对数似然最小化。这里的例子来自Dobson (1990),pp.108-111。这个例子是对剂量-反应数据拟合logistic模型，显然也可以通过glm()拟合。数据如下：

```
> x <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113,
         1.8369, 1.8610, 1.8839)
> y <- c( 6, 13, 18, 28, 52, 53, 61, 60)
> n <- c(59, 60, 62, 56, 63, 59, 62, 60)
```

最小化负对数似然：

```
> fn <- function(p)
    sum( - (y*(p[1]+p[2]*x) - n*log(1+exp(p[1]+p[2]*x))
            + log(choose(n, y)) ))
```

我们挑一个差不多的初值，然后拟合：

```
> out <- nlm(fn, p = c(-50,20), hessian = TRUE)
```

拟合之后，`out$minimum`是负对数似然，`out$estimates`是参数的最大似然估计。要得到大致的标准误估计，我们用：

```
> sqrt(diag(solve(out$hessian)))
```

参数估计值±1.96SE是95%的置信区间。

11.8 一些非标准的模型

作为这章的结束，我们对R提供的其他功能作一个简单介绍，这些功能主要针对特殊的回归和数据分析问题。

- **混合模型(Mixed models)** 用户发布的功能包**nlme**为线性和非线性混合效应模型提供了函数`lme()` 和`nlme()` 这些模型是系数对应随机效应的线性或非线性回归。这些函数要是用大量公式来指定模型。

- **局部逼近回归(Local approximating regressions)** 函数`loess()`通过局部加权回归来拟合一个非参数回归。这类回归。这类回归有助于突现某些杂乱无章数据的趋势，或是简化一个大的数据集。

  函数`loess` 与投影追踪回归（PROJECTION PURSUIT REGRESSION)的代码一同包含在标准功能包**modreg**中。

- **稳健回归(Robust regression)** 有些函数可以用以下的方式拟合模型—这些方法可以抵抗数据中野点的影响。标准功能包**lqs**函数`lqs`为高抵抗性的拟合提供了STATE-OF-ART 算法。一些抵抗性不是那么强，但是统计上更有效的方法可以在用户发布的功能包中找到，比如功能包**MASS**中的函数`rlm`。

- **Additive models**

- **Tree based models** 与寻找一个全局线性模型不同，TBM试图通过递归的方式在决定性变量的关键点将数据分开，是最终数据的分组达到组内尽可能相似，组建尽可能相异。得到的结果通常能够比其他方法得到的结果更深入。模型通过一般线性模型的形式指定。模型拟合函数是`tree()`，许多其它通用函数都可以用来显示TBM的结果，例如`plot()`和`text()`。

  TREE MODELS 包含在用户发布的功能包**rpart**和**tree**中。

Chapter 12

图形过程

图形功能是R环境一个重要且富于变化的组件。通过它可以显示多种统计图形或创建全新的图形类别。

图形功能可以通过交互模式或批处理模式使用，不过大多数情况下交互方式更有效。由于R在启动时就初始化一个图形*设备驱动*，为交互图形的显示打开一个特殊的*图形窗口*，所以使用交互交互模式也比较方便。尽管这个过程是自动的，最好还是了解一下，所使用的命令在UNIX下是`X11()`，在Windows下是`windows()`，在MacOS 8/9下是`macintosh()`。

当设备驱动运行后，R绘图语句就可以用来生成一系列图形显示或者创建全新的显示方式。

绘图语句分为三个基本类别：

- **高级(High-level)** 绘图函数在图形设备上创建一个新图形，通常包括坐标轴，标签，标题等等。
- **低级(Low-level)** 绘图函数在已有图形上添加更多信息，例如额外的点、线和标签。
- **交互(Interactive)** 图形函数允许用户通过鼠标一类的指点设备向已有图形交互的增加信息，或者从中释放信息。

此外，R包含一系列可以用来定制图形的*图形参数*。

12.1 高级绘图命令

高级绘图函数，由函数参数提供数据生成一幅完整的图形。其中适当的坐标轴，标签和标题都自动生成了（除非你另外指定了）。高级绘图命令每次都生成一幅新的图形，清除当前的图形（如果需要的话）。

12.1.1 函数plot()

在R中最经常被用到的一个绘图函数是`plot()`函数。这是一个*通用*函数：生成图形的类型取决于第一个参数的类型或类别(class)。

plot(x,y)

plot(xy)

如果x，y是向量，plot(x,y)生成一幅y对x的散点图。用包含两个元素x，y的一个列表或一个两列的矩阵作为一个参数（第二种形式那样的）也能达成相同的效果。

plot(x)

如果x是一个时间序列，这个命令生成一个时间序列图，如果x是一个数值型向量，则生成一个向量值对它们向量索引的土，而如果x是一个复向量，则生成一个向量中元素的虚部对实部的图。

plot(f)

plot(f,y)

*f*是一个因子对象，*y*是一个数值型向量。第一种形式生成一个*f*的条形图；第二种形式生成*y*对应于*f*各个水平的箱线图。

plot(df)

plot(expr)

plot(y expr)

*df*是一个数据帧，*y*是任意对象，*expr*是由对象名称组成的列表，分隔符'+'(例如，a + b + c)。前两种形式生成分布式的图形，第一种是数据帧中的变量，第二种是一系列被命名的对象。第三种形式生成*y*对*expr*中每个对象的图。

12.1.2 显示多元数据

R为展示多元数据提供了两个非常有用的函数。如果X是一个数值矩阵或数据帧，下面的命令

```
> pairs(X)
```

生成一个配对的散点图矩阵，矩阵由X中的每列的列变量对其他各列列变量的散点图组成，得到的矩阵中每个散点图行、列长度都是固定的。

当问题涉及到三、四个变量时，使用*coplot*更好些。如果a 和b 是数值向量，c 是数值向量或因子对象（全都是相同长度的），下面的命令

```
> coplot(a ~ b | c)
```

对应c的某些给定值生成数个a对b的散点图。当c是一个因子时，这个命令也就是对c的每个水平生成a对b的散点图。当c是数值向量的时候，这个数值向量被分为一系列的条件区间（*conditioning intervals*），对应c的每个区间生成一个a对b的散点图。区间的数量和位置可以通过coplot()的参数given.values来控制，函数co.intervals()也可以用来选择区间。我们也可以使用两个*给定*变量通过命令

```
>coplot(a ~ b | c+d)
```

对c和d的每个联合条件区间生成a对b的散点图。

函数coplot()和pairs()都可以使用参数panel=，这个参数可以用来定制我们得到的图形类型。默认的是points()函数，生成一个散点图，不过通过在参数panel=中提供某些其它的低级图形函数，我们可以生成需要的各种图形。一个很有用的函数例子是panel.smooth()。

12.1.3 显示图形

其他高级图形函数生成不同类型的图形。下面是一些例子：

qqnorm(x)

qqline(x)

qqplot(x,y)

分布比较图。第一种形式生成向量x对期望正态分数（一个正态记分图），第二个在上面的图上添加一条穿过分布分位点和数据分位点的直线。第三个命令生成x的分位点对y分位点图，用于分别比较它们的分布。

hist(x)

hist(x,nclass=n)

hist(x,breaks=b, ...)

生成数值变量x的直方图。通常会自动选定一个合理的类别数，不过可以通过`nclass=`参数来指定一个推荐值。或者通过参数`breaks=`来指定分界点。如果给定了`probability=TRUE`参数，那么条形图代表相对频数而不是累计数。

dotchart(x, ...)

创建一个x中数据的点图(dotchart)。点图中y轴给出x中数据的标签，x轴给出它们的值。它允许对落入某一特定区间的所有数据项方便的进行可视化选择。

image(x,y,z, ...)

contour(x,y,z, ...)

persp(x,y,z, ...)

生成三个变量的图。函数`image`是用不同的颜色绘制一些矩形方格来展示z的值，函数`contour`通过绘制等高线来展示z的值，函数`persp` 绘制一个3D 面。

12.1.4 高级绘图函数的参数

高级图形函数可以使用一系列的参数，如下所示：

add=TRUE 强制函数按照低级图形函数的方式操作，将图形置于当前图形上（仅对某些函数有效）。

axes=FALSE 暂时禁止坐标轴的生成—以便使用`axis()`函数添加你自己定制的坐标轴。默认情况是axes=TRUE，即包含坐标轴。

log="x"

log="y"

log="xy" 令x,y或者两者全都对数化。这个参数对许多函数都有效，不过不是全部。

type= 参数type=控制所生成图形的类型：

type="p" 绘制单独的点（默认值）

type="l" 绘制线

type="b" 绘制由线连接的点（*both*）

type="o" 将点绘在线上

type="h" 绘制从点到零轴的垂线（*high-density*）

type="s"

type="s" 阶梯式图。第一种形式中，点由垂线的顶部定义；第二种形式里用底部定义。

type="n" 不绘制。不过坐标轴是绘出的（默认情况）而且要根据数据绘出坐标系统。用来给后续的低级图形函数创建图形作基础。

xlab=string

ylab=string

x轴或y轴的标签。使用这些参数来改变默认的标签，通常的默认值是调用高级绘图函数时所使用对象的名称。

main=string

图表标题，位于图形的顶部，大字体显示。

sub=string

子标题，位于x轴下面，用较小的字体显示。

12.2 低级绘图命令

有些时候高级绘图函数并不能很精确的生成我们想要的图形。这种情况下，我们可以通过低级绘图命令在当前图形上添加信息（例如，点、线或文本）。

下面列出了一些比较有用的低级绘图函数：

points(x,y)

lines(x,y)

在当前图形上添加点或线。函数plot()的参数type=也可以用于这些函数（默认的是"p"代表points()和"l"代表lines()）。

text(x,y,labels, ...)

给定点坐标x,y，在该点添加文本。通常labels 是一个整数或字符向量，其中labels[i]出现在点(x[i],y[i])。默认值是1:length(x)。

Note: 这个函数通常用于这样的序列中 `> plot(x, y, type="n"); text(x, y, names)`

图形参数type="n"阻止了点的生成，但是建立了坐标轴，由函数text()提供字符向量names所指定的特定字符。

abline(a, b)

abline(h=y)

abline(v=x)

abline(lm.obj)

在当前图上添加一条斜率为b，截距为a的直线。h=y在图形指定的高度上绘制一条贯穿图形的水平线，同样的，v=x 在x轴的指定位置绘制一条贯穿的垂线。而*lm.obj*是一个包含coefficients组件的列表，该组件的长度为2，分别当作截距和斜率。

polygon(x, y, . . .)

绘制一个多边形，其顶点由(x,y)指定。同时还（可选的）可以加上阴影线，如果图形设备允许的话还可以将多边形填充。

legend(x, y, legend, . . .)

这当前图形的指定位置添加图例。绘制的字符，线条类型，颜色等等由字符向量legend指定。除此之外至少还要给出一个参数v，与绘图单元的相应值，分别有：

legend(, fill=v)

填充方框的颜色

legend(, col=v)

绘制点线的颜色

legend(, lty=v)

线条类型

legend(, lwd=v)

线条宽度

legend(, pch=v)

绘制字符(字符向量)

title(main,sub)

在当前图形的顶部用大字题添加一个标题main，在底部用较小的字体添加子标题sub。

axis(side, . . .)

在当前图形的指定边上添加坐标，在哪个边上由第一个参数指定（1到4，从底部按照顺时针顺序）。其他参数控制坐标的位置—在图形内或图形外，以及标记的位置和标签。适合在调用参数为axes=FALSE的函数plot()后添加定制的坐标轴。

低级绘图函数通常都需要一些位置信息（例如，x,y坐标）来决定在哪里添加新的元素。坐标以*用户坐标(user coordinates)*的形式给出，这个坐标系是根据所提供的数据由之前的高级绘图语句定义的。

需要x,y参数的地方还可以选用一个单独的参数，即一个由名为x,y的元素组成的列表。相似的，一个两列的矩阵也可以。像locator()（后面会提到）这样的函数也可以按照这种方式交互的指定图形中的位置。

12.2.1 数学注释

某些情况下需要在图形中加入数学符号或公式。在R中可以通过在text, mtext,axis或title中指定一个表达式来实现。例如，下面的代码绘制了二项概率函数的公式：

```
> text(x, y, expression(paste(bgroup("(", atop(n, x), ")"),
                              p^x, q^{n-x})))
```

更多的信息，包括其特性的一个完整列表可以在R中通过下面的命令得到：

```
> help(plotmath)
> example(plotmath)
```

12.2.2 Hershey 矢量字体

当时用函数text和contour时可以使用Hershey矢量字体来修饰文本。使用Hershey字体通常基于以下三点原因：

- 对旋转的或者小字体，Hershey字体可生成更好的输出，尤其是在电脑屏幕上。
- Hershey字体能够提供标准字体中所没有某些符号。尤其是一些天文学符号和制图符号。
- Hershey字体提供了斯拉夫语和日（平假名和片假名）语字符。

包括Hershey字符表的更多信息可以在R中通过下面的命令得到：

```
> help(Hershey)
> example(Hershey)
> help(Japanese)
> example(Japanese)
```

12.3 图形的交互

R提供了函数，使用户可以通过鼠标从图形中释放信息或添加信息。这些函数中最简单的是locator()函数：

locator(n,type)

> 等待用户使用鼠标左键在当前图形上选择位置。直到n(默认是512)个点都选完或者点击了鼠标另一个键（Unix,Windows)，在Mac下用鼠标点击图形窗口外的部分也可以结束。参数type的效果和在高级绘图函数中使用时是一样的，即在选择的点绘制指定的图形。默认情况是不绘制图形。函数locator()将所选点的坐标返回到一个列表中，列表包含x,y两个组件。

通常locator()在调用的时候并不带参数。这个功能在为图例或标签这样的图形元素选择位置时比较有用，尤其是这些位置不好通过计算得到的时候。举个例子，如果要在一个野点附近添加一些信息，可以使用下面的命令

```
> text(locator(1), "Outlier", adj=0)
```

函数locator()在当前设备不支持鼠标的情况下也可以工作；此时系统会要求用户提供x,y坐标。

identify(x, y, labels)

允许用户在x,y（使用鼠标左键）定义的点附近绘制相应的labels的组件（如果没有给出labels就给出该点的序号），突显该点。当鼠标的另一个键被点击时（Unix,Windows）返回被选点的序号，在Mac下通过点击图形窗口外来实现这个效果。

有时我们更想确定图形中的点，而非它们的位置。例如，我们可能希望从图示中选出某些感兴趣的观测点，然后进行某些操作。通过两个数值向量x,y 给定一系列坐标(x, y)，我们可以使用identify()函数：

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

函数identify()本身不绘图，但是允许用户移动鼠标，在某点附近点击左键。离鼠标指针最近的点将被突显，并标出其序号（也就是它在向量x/y中的位置）。或者使用identify()的labels参数，把某些信息（例如案例名称）作为突显的标志；或者通过plot=FALSE参数取消所有突显标志。当选点的过程结束后，identify()返回所选点的序号；用户可以使用这些序号从x和y中释放所选的点。

12.4 使用图形参数

当创建图形时，尤其是为出版和演讲这样的目的，R并不是总能很精确的生成我们所需要的图形。不过你可以通过*图形参数*定制图形显示的几乎所有方面。R包含大量的图形参数，可以控制的包括线条类型，颜色，图标排列，文本对齐等等。每个图形参数都包括一个名称（例如'col'，控制颜色）和一个值（例如一个颜色数）。

每个被激活的设备都包含一个自己的图形参数列表，每个设备在初始化时都有一套默认的参数设置。图形参数的设定有两种方式：一种是持续性的，即使用当前设备的所有图形函数都要受到影响；另一种是临时的，仅对一个图形函数的调用有影响。

12.4.1 持续性变更(Permanent changes): par()函数

函数par()用于存取和修改当前图形设备的图形参数列表。

par() 不带任何参数，返回当前设备所有图形参数和它们的值的列表。

par(c("col","lty"))

参数为一个字符向量，仅返回参数中指定的图形参数（也是作为一个列表）。

par(col=4, lty=2)

带指定参数（或一个列表型参数），设定指定图形参数的值，并将参数的原始值作为一个列表返回。

通过par()函数设定图形参数的值会持续性的更改参数的值，也就是说这之后（在当前设备上）所有对图形函数的调用都受到新值的影响。你可以这么想，像默认值这样的值都是通过这种方式设定的，所有图形函数都会使用到这些设定，直到给出另外的值。

需要注意的是par()*通常*影响的是图形参数的全局值，即便par()是在函数内调用。这通常不是我们想要的结果—一般我们想做的是设定某些图形参数，

绘制一些图形，然后恢复原来的设定，这样就不会影响用户的R任务。你可以在改变设定时先通过`par()`保存原来的设置，在绘图后恢复原来的设置。

```
> oldpar <- par(col=4, lty=2)     ... plotting commands ...     > par(oldpar)
```

12.4.2 临时性变更：图形函数的参数

图形参数也可以作为（几乎）所有图形函数的命名参数。这种方式的效果和用于par()函数的效果是一样的，只不过这种改变只在函数调用的区间内有效。比如：

```
> plot(x, y, pch="+")
```

生成一个以加号作为绘图符号的散点图，而不改变后续图形的默认的绘图符号。

12.5 图形参数列表

下面的章节给出一些常用图形参数的详细信息。函数par()的R帮助文档提供了一个比较简洁的提要；这里为大家提供的应该是一个比较详细的选择。

图形参数按照下面的形式展示：

name=value

参数效果的描述。name是参数的名称，也就是用在`par()`函数和图形函数中的参数名称。value是设定参数可能用到的一个比较典型的值。

12.5.1 图形元素

R图表由点、线、文本和多边形（填充区）组成。下面的图形参数控制了*图形元素*的绘制：

pch="+" 用来绘点的字符。这个默认值随不同的图形驱动是不同的，不过通常都是'o'。除非使用"."作为绘图字符，否则绘制的点都会比适当的位置高一点或者低一点，而不是恰好在指定位置。

pch=4 当给定一个0到18的整数时，会生成一个特殊的绘图符号。通过下面的命令可以看这些符号都有什么。

```
> legend(locator(1), as.character(0:18), pch=0:18)
```

lty=2 线条类型。并不是所有图形设备都支持多种线条类型（在那些支持的设备上也不全一样），不过线条类型1始终是实线，2及以上的是点、划线或者它们的组合。

lwd=2 线条宽度。所需的线条宽度，是"标准"线条宽度的倍数。对`line()`等函数绘制的线条和坐标轴都有效果。

col=2 点、线、文本、填充区和图像使用的颜色。每种图形元素都有其可用的颜色列表，这个参数的值就是颜色在列表中的序号。显然，这个参数值对有限的一类设备有效。

font=2 指定文本所使用字体的一个整数。如果可能的话，设备驱动会把1对应普通文本，2对应粗体，3对应斜体，4对应粗斜体。

font.axis

font.lab

font.main

font.sub 这几个参数分别指定坐标轴注释，x,y轴的标签，主、副标题所用的字体。

adj=-0.1 文本对齐和绘图位置有关。0代表左对齐，1代表右对齐，0.5代表水平的中间位置。当前的值使会图位置到左端距离的比例，所以-0.1在文本和绘图位置之间留10%的空白。

cex=1.5 字符缩放。这个值是所需文本字符（包括绘图字符）的大小，与默认文本大小相关。

12.5.2 坐标轴和标记

很多R的高级图形都有坐标轴，你可以使用低级图形函数axis()自己创建坐标轴。坐标轴包含三个主要组件：轴线axis line（线条类型由参数lty控制），标记tick mark（沿着轴线划分单元），标号tick label（用来标出这些单元）。这些组件可以用下面这些参数定制。

lab=c(5,7,12)

前两个数字分别是x和y轴上所要划分的区间数。第三个数字是坐标轴标签的长度，用字符数来衡量（包括小数点）。参数的值如果选得太小可能导致所有标号都聚在一起。

las=1 坐标轴标签的方向。0代表总是和坐标轴平行，1代表总是水平的，2代表总是垂直于坐标轴。

mgp=c(3,1,0)

坐标轴组件的位置。第一个组件是坐标轴标签到坐标轴的距离，单位是文本行(text lines)。第二个组件是到标号的距离，最后一个是轴的巍直到轴线的距离（一般都是0）。正数代表绘图区域外，负数代表区域内。

tck=0.01 标号的长度，绘图区域大小的一个分数作单位。当tck比较小时（小于0.5），就强制x和y轴上的标记为相同大小。tck=1就是生成网格线。取负值时标记画向绘图区域外。内部标记可以使用tck=0.01和mgp=c(1,-1.5,0)。

xaxs="s"

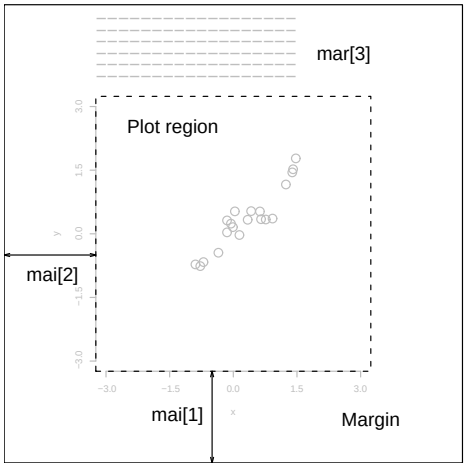
yaxs="d" 分别是x、y轴的类型。如果是s(standard)或e(extended)类型，那最大和最小的标记都始终在数据区域之外。如果有某个点离边界非常近，那么扩展型(extended)的轴会稍稍扩展一下。这种类型的轴有时会在边界附近留出大片空白。而i(internal)或r(默认值)类型的轴，标记始终在数据区域内，不过r类型会在边界留出少量空白。

如果这个参数设为d，就锁定当前轴，对之后绘制的所有图形都用这个轴（直到参数被重新设定为其他的那几个值）。这个参数适用于生成一系列固定尺度的图。

12.5.3 图边缘(Figure margins)

在R中一个单独图形，图（`figure`），包含一个绘图区(*plot region*)，以及环绕着这个区域的边缘（其中可能含有坐标轴标签、标题等等），（通常）这两部分以轴为边界。

一个典型的图是



控制图的样式的图形参数包括：

mai=c(1,0.5,0.5,0)

分别是底部，左侧，顶部，右侧的宽度，单位是英寸。

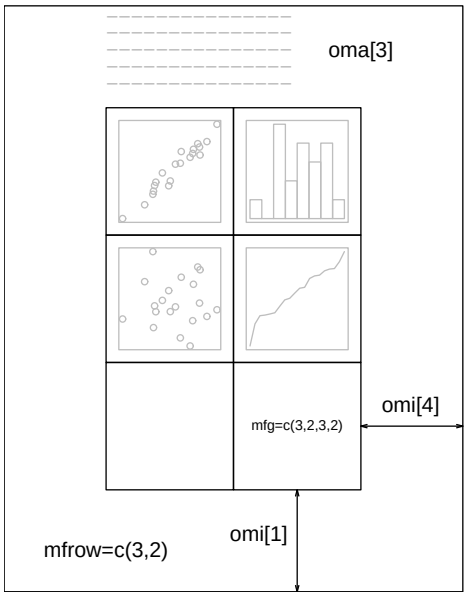
mar=c(4,2,2,1)

与`mai`相似，只是单位是文本行(text lines)。

由于更改一个就会改变另一个的值，所以在这个意义上，`mai`和`mar`是等价的。这个参数的默认值通常都太大了；右侧边缘很少用到，如果没标题，顶部边缘也不需要，左侧和底部的边缘应当足够大，以容纳坐标轴和标号。此外，默认值并没有考虑到设备表面的大小：比如，使用`postscript()`驱动，`height=4`参数时，除非用`mar`，`mai`另外设定，否则图的边缘就要占上大约50%。使用多图环境时（后面会提到）。边缘会减少一半，不过这在多图共用一页的时候可能还不够。

12.5.4 多图环境

R允许在一页上创建一个$n \times m$的图的阵列。每个土豆由自己的边缘，图的阵列还有一个可选的外部边缘，如下图所示。



与多图环境相关的图形参数有：

mfcol=c(3, 2)

mfrow=c(2, 4)

设定多图阵列的大小。第一个值是行数，第二个值是列数。这两个参数唯一的区别是`mfcol`把图按列排入，`mfrow`把图按行排入。上图所示的版式可用`mfrow=c(3,2)`创建；上图显示的是绘制四幅图后的情况。

mfg=c(2, 2, 3, 2)

当前图在多图环境下的位置。前两个数字是当前图的行、列数；后两个是其在多图阵列中的行列数。这个参数用来在多图阵列中跳转。你甚至可以在后两个数中使用和真值(true value)不同的值，在同一页上得到大小不同的图。

fig=c(4, 9, 1, 4)/10

当前图在页面的位置，取值分别是左下角到左边界，右边界，下、上边界的距离与对应边的百分比数。给出的例子是一个页面右下角的图。这个参数可以设定图在页面的绝对位置。

oma=c(2, 0, 3, 0)

omi=(0, 0, 0.8, 0)

外部边缘的大小。与mar和mai相似，第一个用文本行作单位，第二个以英寸作单位，从下方开始按照顺时针顺序指定。

外部边缘对页标题这类东西很有用。文本可以通过带`outer=TRUE`参数的`mtext()`函数加入外部边缘。默认情况下是没有外部边缘的因此必须通过`oma`或`omi`指定。

函数`split.screen()`和`layout()`可以对多个图形作更复杂的排列。

12.6 设备驱动

R几乎可以在任何一种类型的显示器和打印设备上生成（不同质量的）图形。不过，在这之前，需要告诉R要处理的是哪一种设备。这通过启动一个设备驱动来完成。设备驱动的作用是把R的图形指令(例如，"画一条线")转化成某个特定设备可以明白的形式。

设备驱动通过调用设备驱动函数来启动。每个设备驱动都有一个这样专门的函数。用`help(Devices)`可以得到所有设备驱动启动函数的一个列表。比如

```
> postscript()
```

把所有后续的图形输出都以PostScript格式发送到打印机。某些常用的设备驱动有：

X11() 使用X11视窗系统

postscript() 在PostScript打印机上打印或者创建PostScript图形文件

pictex() 生成一个LaTeX文件

当一个设备使用完之后，可以通过下面的命令终止设备驱动

```
> dev.off()
```

这个命令可以确保设备已经结束；例如，在某些硬拷贝的设备中，这个命令可以保证每页都已经完成，并且都被传送到打印机了。

12.6.1 文本文档的PostScript图表

通过给`postscript()`函数带上`file`参数，我们可以把图形以`PostScript`格式存储到文件中。如果没有给出horizontal=FALSE参数，图形是横向的，你可以通过width和height参数控制图形的大小（图形会自动适应）。例如，命令

```
> postscript("file.ps", horizontal=FALSE, height=5, pointsize=10)
```

为一个五英寸的图生成一个包含PostScript代码的文件，可以放在文当中。需要注意的是，如果命令中的指定的文件名已经存在，将会被覆盖。即便这个文件是刚刚在相同的R任务创建的，也会被覆盖。

12.6.2 多重图形设备

在R的高级应用里，经常需要同时使用若干个图形设备。当然，每次只能有一个图形设备接收图形命令，这个设备就是*当前设备*。当多重设备打开后，它们形成一个序列，可以分别用序号和名称来指定。

操作多重设备的主要命令和它们的含义如下：

X11() [Unix]

windows() [Windows]

Macintosh() [MacOS 8/9]

postscript()

pictex()

... 每个对设备驱动的新调用都会打开一个新的图形设备，在设备列表中加入新的一项。这个设备就成为当前设备，图形输出就传送到这个设备。（某些平台下有更多的可用设备）

dev.list() 返回所有活动中设备的序号和名称。在列表位置1的设备始终是空设备(*null device*)，这个设备不接收任何图形命令。

dev.next()

dev.prev() 分别返回当前设备的后一个和前一个设备的序号和名称。

dev.set(which=k) 用来把当前设备更改为设备列表中位置k的那个。返回设备的序号和标签。

dev.off(k) 终止图形列表位置k的那个图形设备。对于某些设备，比如`postscript`，这个命令会立刻打印文件或者正常结束文件，具体怎样处理取决于设备是怎样初始化的。

dev.copy(device,...,which=k)

dev.print(device,...,which=k)

建立一个设备k的拷贝。其中device是一个设备函数，例如postscript，如果需要的话可以在'...'中指定其它的参数，dev.print效果相似，不过复制的设备会立刻关闭，所以打印硬拷贝这样的终止操作也会被立即执行。

graphics.off() 终止列表中的所有图形设备，空设备除外。

12.7 动态图形

R（目前）没有一个内建的动态图形函数，执行例如旋转一片点和交互突显图形这类的动作。不过在Swayne,Cook,Buja的XGobi系统中有相当多的动态图形功能，可以从

`http://www.research.att.com/areas/stat/xgobi/`

得到，在R中通过xgobi功能包来存取。

目前XGobi可以在Unix和Windows这样的X Windows系统下运行，在每个平台下都有可用的R 界面。