

R 导论

Next: [Preface](#), Previous: [\(dir\)](#), Up: [\(dir\)](#)

R 导论

本文档是对 R (“ GNU S ”) 的入门介绍。R 是一种为统计计算和图形显示 而设计的语言及环境，它和贝尔实验室 (Bell Laboratories) John Chambers 等人开发的 S 系统相似。它提供了一系列统计和图形显示工具 (线性和非线性模型，统计检验，时间序列分析，分类，聚类，.....)。

该文档提供了有关 R 的数据类型，编程原理，统计建模和图形显示等方面的信息。

文档的当前版本是 0.0.1 (2005-06-05)。该文档译自 R-2.1.0 文档。

本翻译文档还包括一个对 R 简要介绍的 [PowerPoint](#) 幻灯片。非常适合第一次使用 R 的朋友看看。

本翻译文档还提供 [CHM](#) 版本下载。

提供最新的 pdf 文档 <http://www.biosino.org/R/R-doc/>。

丁国徽 (ghding@gmail.com) 译。

ISBN 3-900051-12-7

[Preface](#): 序言

[Introduction and preliminaries](#): 绪论

[Simple manipulations numbers and vectors](#): 简单的算术操作和向量运算

[Objects](#): 对象

[Factors](#): 因子

[Arrays and matrices](#): 数组和矩阵

[Lists and data frames](#): 列表和数据框

[Reading data from files](#): 从文件中读取数据

[Probability distributions](#): 概率分布

[Loops and conditional execution](#): 循环和条件控制

[Writing your own functions](#): 编写函数

[Statistical models in R](#): R 中的统计模型

[Graphics](#): 图形显示

[Packages](#): 包

[A sample session](#): 一个简单的会话

[Invoking R](#): 调用 R

[The command-line editor](#): 命令行编辑器

[Function and variable index](#): 函数和变量索引

[Concept index](#): 概念索引

[References](#): 参考文献

丁国徽 (ghding@gmail.com) 译。

R 导论

Next: [Preface](#), Previous: [\(dir\)](#), Up: [\(dir\)](#)

R 导论

本文档是对 R (“ GNU S ”) 的入门介绍。R 是一种为统计计算和图形显示 而设计的语言及环境，它和贝尔实验室 (Bell Laboratories) John Chambers 等人开发的 S 系统相似。它提供了一系列统计和图形显示工具 (线性和非线性模型，统计检验，时间序列分析，分类，聚类，.....)。

该文档提供了有关 R 的数据类型，编程原理，统计建模和图形显示等方面的信息。

文档的当前版本是 0.0.1 (2005-06-05)。该文档译自 R-2.1.0 文档。

本翻译文档还包括一个对 R 简要介绍的 [PowerPoint](#) 幻灯片。非常适合第一次使用 R 的朋友看看。

本翻译文档还提供 [CHM](#) 版本下载。

提供最新的 pdf 文档 <http://www.biosino.org/R/R-doc/>。

丁国徽 (ghding@gmail.com) 译。

ISBN 3-900051-12-7

[Preface](#): 序言

[Introduction and preliminaries](#): 绪论

[Simple manipulations numbers and vectors](#): 简单的算术操作和向量运算

[Objects](#): 对象

[Factors](#): 因子

[Arrays and matrices](#): 数组和矩阵

[Lists and data frames](#): 列表和数据框

[Reading data from files](#): 从文件中读取数据

[Probability distributions](#): 概率分布

[Loops and conditional execution](#): 循环和条件控制

[Writing your own functions](#): 编写函数

[Statistical models in R](#): R 中的统计模型

[Graphics](#): 图形显示

[Packages](#): 包

[A sample session](#): 一个简单的会话

[Invoking R](#): 调用 R

[The command-line editor](#): 命令行编辑器

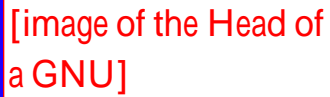
[Function and variable index](#): 函数和变量索引

[Concept index](#): 概念索引

[References](#): 参考文献

丁国徽 (ghding@gmail.com) 译。

Texinfo - The GNU Documentation System



[image of the Head of
a GNU]

Texinfo is the official documentation format of the [GNU project](#). It was invented by Richard Stallman and Bob Chassell many years ago, loosely based on Brian Reid's Scribe and other formatting languages of the time. It is used by many non-GNU projects as well.

Texinfo uses a single source file to produce output in a number of formats, both online and printed (dvi, html, info, pdf, xml, etc.). This means that instead of writing different documents for online information and another for a printed manual, you need write only one document. And when the work is revised, you need revise only that one document. The Texinfo system is well-integrated with [GNU Emacs](#).

Help wanted: I would be very grateful for help maintaining Texinfo. Standalone Info in particular could be maintained more or less independently of the other programs. I also have a long list of bug reports and feature requests for makeinfo, the Texinfo language, etc. For an overview of some of them, see the [TODO](#) file in the distribution. Please [write me](#) if you can help.

Basics

[Download](#) the latest official release as a gzipped tar file (also on all [GNU mirrors](#)). Or [browse the unpacked release](#) (also on all [CTAN mirrors](#)).

The [Texinfo manual](#), [Info manual](#), and [standalone Info reader manual](#) from the latest official release, in various formats.

The [latest texinfo.tex macro file](#) used for converting Texinfo source to DVI or PDF files for printing and viewing, using the [TeX typesetting system](#). The latest texinfo.tex is also available via the [Savannah gnulib project](#) and [tug.org](#), also [gzip-compressed](#) and [bzip2-compressed](#). (texinfo.tex is updated much more frequently than full Texinfo releases are made; please use the latest version.)

If available, the [latest pretest by ftp](#) ([alternate location via http](#)). Pretests are inherently unstable, may not work, etc.

The [development sources via CVS](#), from [savannah.gnu.org](#).

[Old releases](#), for historical and archeological purposes.

The [NEWS](#) file listing notable changes by release, the [ChangeLog](#) file detailing all changes, and the [TODO](#) file with future projects large and small, awaiting volunteers.

Mailing lists

bug-texinfo@gnu.org

The main address for bug reporting and developer discussion ([web archive](#), [plain text archive](#)).

help-texinfo@gnu.org

For Texinfo authoring help and discussion ([archive](#), [plain text archive](#)).

texinfo-pretest@texinfo.org

To discuss the latest test releases and development issues ([archive](#)).

texinfo-cvs@texinfo.org

To get mail with every CVS commit to the repository ([archive](#)).

Anyone is welcome to subscribe to any or all of these. Do so by sending a message to the corresponding -request address, for example bug-texinfo-request@gnu.org. A single line

`subscribe yourusername@yourdomain.dom`

in both the body and the subject works best. The plain text archives may be especially useful for Texinfo, since the web archiver considers many Texinfo @-commands as email addresses and therefore mangles them.

More Texinfo

Documentation:

The [official GNU documentation page](#) contains much more information about Texinfo and GNU manuals.

You can [search all GNU documentation](#) (among other interesting things).

An [introductory article](#) written by Arnold Robbins.

[Other Texinfo converters](#), a page maintained by kama.

Programs:

[a2ps](#), for generating printable files from plain text, has a Texinfo style option.

[docbook2x](#), conversion from Docbook to Texinfo.

[gdoc](#), for extracting Texinfo (and other format) documentation from source files.

[help2man](#), for generating good man pages from --help output.

[ltxindex](#), for typesetting indices of LaTeX documents with Texinfo's texindex program.

[man2texi](#), generate Texinfo from man pages.

[my-bib-macros.texi](#), a start at bibliography support via Texinfo macros. For a start at integration with BibTeX, see the [Web2C manual sources](#).

[pinfo](#), a curses-based Info reader with color support and a lynx-like interface.

[sgmltexi](#), Texinfo authoring via SGML tools.

[Scheme Scribe](#), which can convert Texinfo to Scribe and generate Info files.

[texi2html](#), an (actively maintained) alternative translator for generating HTML, roff, and chm.

[texi2latex](#), converting Texinfo to LaTeX via XML.

[texi2roff](#), an old standalone program that converts to troff format; it is not maintained and not up to date.

[tkinfo](#), a Tk-based Info reader.

If you know of free software or documentation that would be useful to add here, please email bug-texinfo@gnu.org.

Return to [GNU's home page](#).

Please send comments on GNU web pages in general to webmasters@www.gnu.org (and bug-texinfo@gnu.org for this page in particular), and send other FSF & GNU questions and inquiries to gnu@gnu.org. [Other ways to contact the FSF](#).

Copyright (C) 2004 Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111, USA

Verbatim copying and distribution of this entire article is permitted in any medium, provided this notice is preserved.

Updated: \$Date: 2006/10/10 15:04:09 \$ \$Author: karl \$

Next: [Introduction and preliminaries](#), Previous: [Top](#), Up: [Top](#)

序言

该文档改自 Bill Venables 和 David M. Smith (Insightful 公司) 描述 S 和 S-Plus 开发环境的讲义。我们只做了一些很小的修改以反映 R 和 S 的差异，并且扩充了一些材料。

非常感谢 Bill Venables 和 David Smith 允许我们以这种方式发布该讲义的修改版本 以及他们一直以来对 R 不断的支持。

各种评论和校正可以通过电子邮件 R-core@R-project.org 联系我们。

对读者的建议

大多数 R 的新手可以从附录 A 中的引导性的会话 (session) (see [A sample session](#))入手，从而对 R 会话 (R sessions) 有一些简单的认识。更为重要的是，新手将会从这些 R 会话中对一些即时的命令获得实时的反馈。

有些用户仅仅关心 R 的绘图功能。在这种情况下，可以马上跳到关于绘图功能的章节([Graphics](#)) 而没有必要等到所有前面的章节 都看完。

下面是译者对还没有安装 R 程序的用户的一个简单的提示。新用户可以去<http://cran.r-project.org/mirrors.html>，随便找一个可以访问的地址，点击进入，下载 “ Precompiled Binary Distributions ” 栏中的软件。如果是 Windows 用户，可以点击 “ Windows (95 and later) ”，进入 “ base ”，下载 “ rwxxx.exe ” (如 rw2010.exe)，然后就像一般的 Windows 软件安装即可。OK，你可以试试附录 A 中的引导性的会话(see [A sample session](#))了。

[Copyright](#): 版权声明

[Word from the Translator](#): 译者前言

[Introduction and preliminaries](#): 绪论

Previous: [Copyright](#), Up: [Preface](#)

译者前言

经过几周的努力，我终于完成了《R 导论》的翻译。这是一本关于 R 环境入门的教材，同时也是 R 官方文档中面向大众用户的教材。一般用户只要看完这本《R 导论》，并且运行完上面的例子，就可以解决大部分问题了。我没有把文档叫做 R 入门，因为这份文档中有些内容已经超出一般用户的要求，但是把 R 作为一个系统看，那些确实又是最基本的。过一些时候，我会列出一个适合一般统计用户看的内容清单。

我是在做第一轮 ROTATION 的时候接触 R 的，后来就一直用上了这个软件。虽然我还摸过 Stata，MATLAB，SPSS 和 SAS 等统计计算方面的软件，但是 R 依然是我的首选。我选择 R 的理由是：

R 是自由软件。它不会向你收取任何费用，但是它的功能不会比任何同类型商业软件差。从功能相似的角度来说，R 和 MATLAB 最像的。

通过 R 你可以和全球一流的统计计算方面的专家合作讨论，它是全世界统计学家思维的最大集中。我加入了 R 的邮件列表，每天都会收到五六十份关于 R 的资讯邮件。

它是彻底的面向对象的统计编程语言。对于我这种生长在面向对象编程模式年代里的人可是非常容易理解和使用的。

R 和其他编程语言/数据库之间有很好的接口。代码整合的时候感觉 R 为你提供了一系列对象，你用其他语言只要调用这些对象就可以了。这对数据整合工作非常有用。

这里我要说明的是，虽然 R 是自由软件，但是我们要记得感激所有为 R 贡献智慧的奉献者。所以，大家对 R 的支持可以体现在行动上，有力出力有钱出钱或者至少会这样想，这也是出于对知识劳动的尊重。具体可以访问<http://www.r-project.org/foundation/about.html>。

整个文档的翻译是直接对 Texinfo 文档进行的。由于 Texinfo 的中文 PDF 转换问题还没有完全解决，所以我先用 HTML 格式(字符集是 UTF-8)发布我的文档。不过，我相信这个问题可以解决的（最笨的方案就是把 Texinfo 转换成 LaTeX）。PDF 版本的文档会很快和大家见面。另外，我同时发布简体中文版和繁体中文版的译文以方便使用繁体的朋友。

由于时间比较紧张，翻译过程中可能有不少错误。有些数学术语也可能斟酌不够。因此，希望各个兄弟姐妹发现错误后给我发 email（ghding@gmail.com），我会尽快修改。现在还只是 版，在大家的砖头锤炼后发布正式的版本。那时候，肯定 PDF 版本的也搞定了。当然，也要提交给 R 核心开发小组了。

我的目标是利用业余时间翻译 R 的五份关键文档《An Introduction to R》（本册），《R Data

Import/Export》, 《The R language definition》, 《Writing R Extensions》, 《R Installation and Administration/R FAQ》。每天在中午 和黄昏的时候翻译一两部分这些文档还是一件不错的事情。我会尽快发布这些文档。本册在 这一套文档中地位是统领性的, 也是一般读者需要看的, 所以我提前发布这个文档了。其他的 文档有点偏向高级用户或者开发用户, 如果不想深入了解 R, 可以不看。

“ 工欲工欲善其事, 必先利其器 ”, 我常常用 R, 所以学好它是应该的。而这个翻译的过程中, 我学到了很多東西。

非常感谢东京工业大学的 Shigeru MASE 以及身边的各位朋友。

丁国徽

Email : ghding@gmail.com

2005年6月3日

Next: [Simple manipulations numbers and vectors](#), Previous: [Preface](#), Up: [Top](#)

1 绪论

[The R environment](#): R 环境

[Related software and documentation](#): 相关软件和文档

[R and statistics](#): R 和统计

[R and the window system](#): R 和桌面系统

[Using R interactively](#): 交互式使用 R

[Getting help](#): 寻求帮助

[R commands; case sensitivity etc](#): R 命令 , 大小写敏感等

[Recall and correction of previous commands](#): 重新调用和修正先前的命令

[Executing commands from or diverting output to a file](#): 批处理文件和结果重定向

[Data permanency and removing objects](#): 永久数据和对象删除

Next: [Related software and documentation](#), Previous: [Introduction and preliminaries](#), Up: [Introduction and preliminaries](#)

1.1 R 环境

R 是一组数据操作，计算和图形显示工具的整合包。相对其他同类软件，它的特色在于：

- 有效的数据处理和保存机制，
- 拥有一整套数组和矩阵的操作运算符，
- 一系列连贯而又完整的数据分析中间工具，
- 图形工具可以对数据直接进行分析和显示，可用于多种图形设备，
- 一种相当完善，简洁和高效的程序设计语言(也就是 'S')。它包括条件语句，循环语句，用户定义的递归函数以及输入输出接口。（实际上，系统提供的大多数函数都是用 S 写的）。

和其他数据分析软件的一样，术语“环境”（environment）是用来描述一个充分设计、连贯性的系统，而不是一个非常专一并且难以扩充的工具群。

R 是一个开发新的交互式数据分析方法的工具。它的开发周期短，而且有大量的扩展包（packages）可以使用。不过，大多数用 R 开发的程序仅仅是为了处理一些特定的数据，因此很快就被淘汰了。

Next: [R and statistics](#), Previous: [The R environment](#), Up: [Introduction and preliminaries](#)

1.2 相关软件和文档

R 可以看作贝尔实验室 (Bell Laboratories) 的 Rick Becker , John Chambers 和 Allan Wilks 开发的 S 语言的一种实现。当然 , S 语言也是 S-Plus 的基础。

关于 S 语言的发展过程可以参考 John Chambers 及其合著者们编写的四本书。对于 R , 最主要的参考书是 Richard A. Becker , John M. Chambers 和 Allan R. Wilks 著的 *The New S Language: A Programming Environment for Data Analysis and Graphics*。另外 , John M. Chambers 和 Trevor J. Hastie 编的 *Statistical Models in S* 覆盖了 1991 发布的 S 3 版本的一些新特征。 methods 包中的方法 (method) 和类 (class) 则是基于 John M. Chambers 著的 *Programming with Data*。具体参考书目 See [References](#)。

现在已经有很多关于如何用 R 进行数据分析和统计的书籍。 S/S-Plus 方面的文档都可以直接用于 R , 不过要注意 R 在 S 实现上的差异。 See [What documentation exists for R? \(The R statistical system FAQ\)](#)。

Next: [R and the window system](#), Previous: [Related software and documentation](#), Up: [Introduction and preliminaries](#)

1.3 R 和统计

我们对 R 环境的介绍没有提到 *统计*，但是大多数人用 R 就是因为它的统计功能。不过，我们宁可把 R 当作一个内部实现了许多经典的时髦的统计技术的环境。部分的统计功能是整合在 R 环境的底层，但是大多数功能则以 *包* 的形式提供。大约有25个包和 R 同时发布（被称为“标准”和“推荐”包），更多的包可以通过网上或其他地方的 CRAN 社区 (<http://CRAN.R-project.org>) 得到。关于包更多的细节将在后面的章节叙述 (see [Packages](#))。

大多数经典的统计方法和最新的技术都可以在 R 中直接得到。终端用户只是需要花点精力去找到一下就可以了。

S (因此也包括 R) 和其他主要的统计系统在观念上有着重要的差异。在 S 语言中，一次统计分析常常被分解成一系列步骤，并且所有的中间结果都被保存在对象 (object) 中。因此，尽管 SAS 和 SPSS 为回归和判别分析提供了丰富的屏幕输出内容，但 R 给出屏幕输出却很少。它将结果保存在一些合适的对象中以便于用 R 里面的函数做进一步的分析。

Next: [Using R interactively](#), Previous: [R and statistics](#), Up: [Introduction and preliminaries](#)

1.4 R 和桌面系统

最容易的方法就是在一个桌面系统的图形工作站（graphics workstation）上运行 R。这个提示主要是对有这样便利的读者说的。大多数情况下我们不会特指在什么 R 环境下使用 R，但我们偶尔会提到在 X window 系统上使用 R。

许多用户发现有必要在他们的计算机系统上和程序进行实时的交流。在这本手册里面，我们主要讨论在 UNIX 操作系统中的交互式操作。如果你在 Windows 或者 MacOS 上运行 R，你可能需要做少量的调整。

为充分利用 R 的个性化特征，直接设置图形工作站是最直接的办法。不过，这种方法有点乏味。我们不准备讨论这个问题。用户如果遇到这方面的问题可以向你身边高手寻求帮助。

Next: [Getting help](#), Previous: [R and the window system](#), Up: [Introduction and preliminaries](#)

1.5 交互式使用 R

当一个 R 程序需要你输入命令时，它会显示命令提示符。默认的提示符是 `>`。UNIX 系统中可能会和 shell 的命令提示符一致。它还提示了当前环境没有任务运行。但是，正如我们所期望的一样，你很容易设定你想要的不同于默认值的 R 命令提示符。在接下来的文档中，我们将假定 UNIX 的 shell 命令的提示符是 `$`。

如果你是第一次在 UNIX 系统使用 R，我们推荐的操作步骤如下：

1. 创建一个独立的子目录 work 来保存你要在这个系统上用 R 分析的数据文件。当你用 R 处理这些数据时，这将是你的工作目录。

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

2. 用命令启动 R 程序。

```
$ R
```

3. 此时，可以键入 R 的命令 (见后面的内容)。
4. 退出 R 程序的命令是

```
> q()
```

这个时候，R 会话会问你是否需要保存数据。有些系统会弹出一个会话框，另外一些系统则会给出文本命令提示。对于文本命令提示，你可以键入 yes，no 或者 cancel（首字母就行了）以表示在退出前保存数据，不保存数据就退出，和重新返回 R 会话。被保存的数据可以被将来的 R 会话调用。

再一次进行 R 会话是比较简单的。

1. 创建工作目录 work，和前面一样的步骤启动程序：


```
$ cd work  
$ R
```

2. 使用 R 程序，用 q() 命令结束 会话。

在 Windows 系统使用 R 的操作流程在本质上是一样的。创建一个工作目录，将该目录设置为 R 桌面快捷方式的起始位置¹，双击快捷图标以启动 R。

1.6 一个引导性的 R 会话

如果读者现在极力想在自己的电脑上体验一下 R 的魅力，那就迅速把附录 [A sample session](#) 中给出的一个引导性 R 会话做完。这个引导性的 R 会话非常值得推荐。

Footnotes

[1] 译者注：右键点击 R 桌面快捷方式，出现一个菜单，选择“属性”，然后弹出会话框，点击“快捷方式”，就可以看到“起始位置”。填上你所创建的工作目录，就OK^_^)

Next: [R commands; case sensitivity etc](#), Previous: [Using R interactively](#), Up: [Introduction and preliminaries](#)

1.7 通过函数和特征寻求帮助

R 有一个和UNIX的帮助命令 `man` 类似的内嵌帮助工具。为了得到任何特定名字的函数的帮助，如 `solve`，可以使用如下命令

```
> help(solve)
```

还有一种办法就是

```
> ?solve
```

对于有特殊含义的字符，可以加上双引号或者单引号，即“字符串”：这同样适用于有语法含义的关键字 `if`，`for` 和 `function`。

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

任一种引号都可用于逃逸（escape）另外一种，如字符串 `"It's important"`¹。习惯上，一般优先使用双引号。

在大多数 R 平台中，你可以通过运行下面的命令得到 HTML 格式的帮助。

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

它会启动一个网页浏览器，允许你通过超链访问帮助页。在 UNIX 系统中，后续的帮助要求可以发送给基于 HTML 的帮助系统。再 `help.start()` 启动的浏览页上，‘搜索引擎和关键词’（Search Engine & Keywords）链接特别有用，因为它通过搜索可以使用的函数而提供一个高层次的概念列表。这会让你很快认清自己所处的位置和理解 R 所提供的函数能力范围。

`help.search` 命令² 允许你用任何方式搜索帮助文档：键入 `?help.search` 看一下这个命令的详细信息和例子。

关于某个帮助主题的例子，可以用下面的命令查看。

```
> example(topic)
```

Windows 版本的 R 还有其他可选的帮助工具：可以利用下面的命令得到更多的信息。

> ?help

Footnotes

[1] 译者注：如果采用 'It's important' , R 就会告诉你语法错误！

[2] 译者注：我非常喜欢这个命令，除?外它可能是我用的最多的一个

Next: [Recall and correction of previous commands](#), Previous: [Getting help](#), Up: [Introduction and preliminaries](#)

1.8 R 命令，大小写敏感等

技术上来说，R 是一种语法非常简单的表达式语言 (expression language)。它大小写敏感，因此 A 和 a 是不同的符号且指向不同的变量。可以在 R 环境下使用的命名字符集依赖于 R 所运行的系统和国家 (就是系统的 *locale* 设置)。通常，数字，字母，`.` 和 `_` 都是允许的 (在一些国家还包括重音字母)。不过，一个命名必须以 `.` 或者字母开头，并且如果以 `.` 开头，第二个字符不允许是数字。

基本命令要么是表达式 (expressions) 要么就是赋值 (assignments)。如果一条命令是表达式，那么它将会被解析 (evaluate)，并将结果显示在屏幕上，清空该命令所占内存。赋值同样会解析表达式并且把值传给变量但结果不会自动显示在屏幕上。

命令可以被 `(;)` 隔开，或者另起一行。基本命令可以通过大括弧 (`{` 和 `}`) 放在一起构成一个复合表达式 (compound expression)。注释几乎可以放在任何地方^[1]。一行中，从井号 (`#`) 开始到句子收尾之间的语句就是注释。

如果一条命令在一行结束的时候在语法上还不完整，R 会给出一个不同的提示符，默认是

+

该提示符会出现在第二行和随后的行中，它持续等待输入直到一条命令在语法上是完整的。该提示符可以被用户修改。在后面的文档中，我们常常省略延续提示符 (continuation prompt)，以简单的缩进表示这种延续。

Footnotes

[1] 不要放在字符串之中，也不要放在一个函数定义的参数列表中

Next: [Executing commands from or diverting output to a file](#), Previous: [R commands; case sensitivity etc](#),
Up: [Introduction and preliminaries](#)

1.9 重新调用和修正先前的命令

在大多数 UNIX 版本和 Windows 系统上，R 提供了一种 调用和重新执行先前用过的命令的机制。键盘上的上下键 可以使光标在 *命令的历史记录* (command history) 中前翻或者 后退。一旦找到某条具体的命令，光标可以通过 左右键移动，而且 可以插入其他字符或者用键删除字符。更具体的内容可以参考：see [The command-line editor](#).

UNIX 下面以前用过的命令的调用和编辑的键设置是可以修改的，具体内容可以参考 readline 库的指南。

特别提一下，Emacs 文本编辑器¹对交互使用 R 提供了更一般的机制 (主要是 ESS 和 *Emacs Speaks Statistics*)。See [R and Emacs \(The R statistical system FAQ\)](#).

Footnotes

[1] 译者注：当然Vi，UltraEdit也不错。我就用UltraEdit写 R 代码做蒙特卡罗模拟的。在<http://www.ultraedit.com/index.php?name=Content&pa=showpage&pid=40>下载 R 的 wordfile (“ R Scripting - 02/18/2003 ”，具体可能有些不同)，贴到你的 UltraEdit 程序根目录下的 wordfile.txt 文件中就行了。这时候，自动会对 .R 文件语法高亮显示。

Next: [Data permanency and removing objects](#), Previous: [Recall and correction of previous commands](#),
Up: [Introduction and preliminaries](#)

1.10 批处理文件和结果重定向

如果一批命令保存在工作目录 work 下一个叫 commands.R 的文件中，可以用下面的命令在 R 会话中执行这个文件。

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

在 Windows 版本的 R 控制台中，可以在文件（File）菜单中选择载入（Source）来实现。对于函数 sink，

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

可以把所有后续的输出结果从控制台重定向到外部文件¹ record.lis 中。命令

```
> sink()
```

让你输出流重新定向到控制台。

Footnotes

[1] 译者注：此时，控制台可以看见你输入的命令，但是看不见输出结果。因为结果都定向到外部文件中去了。

Previous: [Executing commands from or diverting output to a file](#), Up: [Introduction and preliminaries](#)

1.11 永久数据和对象删除

R 创建和控制的实体 (entity) 被称为 *对象*。它们可以是变量，数组，字符串，函数，或者其他通过这些实体定义的更一般性的结构 (structures)。

在 R 会话过程中，对象是通过名字创建和保存的(我们将在后面的 话题中讨论这个过程)。R 的命令(或者ls())

```
> objects()
```

是用来显示当前 保存在 R 环境中的对象名字 (可能不是全部名字)。保存当前对象的地方被称为*工作空间* (workspace)。可以通过命令 rm 删除对象：

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

R 会话中创建的所有对象可以永久地保存在一个文件中 以便于以后的 R 会话调用。在每一次 R 会话结束的时候，你可以保存所有当前可用的对象。如果你想这样做，这些对象将会 写入当前目录下下一个叫 .RData^[1] 的文件中，并且所有在这次会话中用过的命令行 都会被保存在一个叫 .Rhistory 的文件中。

当 R 再次在同一目录下启动时，这些对象将从这个文件中 重新导入工作空间。同时，相关的历史命令文件也会 被导入。

如果采用 R 做分析，你最好用相对独立的工作目录。在分析过程中，将对象命名为 x 和 y 是一件非常常见的事情。在一次独立的分析中，这样的命名是有特定含义的，但若几个分析同时在一个目录下进行时，区别它们的涵义 可能是一件非常困难的事情。

Footnotes

[1] 文件名起始的“点”可能会让文件在常规的 UNIX 文件列表中不可见，即隐藏文件。

Next: [Objects](#), Previous: [Introduction and preliminaries](#), Up: [Top](#)

2 简单的算术操作和向量运算

[Vectors and assignment](#): 向量和赋值

[Vector arithmetic](#): 向量运算

[Generating regular sequences](#): 正规序列的产生

[Logical vectors](#): 逻辑向量

[Missing values](#): 缺省值

[Character vectors](#): 字符向量

[Index vectors](#): 索引向量

[Other types of objects](#): 其他类型的对象

Next: [Vector arithmetic](#), Previous: [Simple manipulations numbers and vectors](#), Up: [Simple manipulations numbers and vectors](#)

2.1 向量和赋值

R 是在指定的 **数据结构** (data structure) 上起作用的。其中，最简单的结构就是由一系列有序数值构成的数值 **向量** (vector)。假如我们要创建一个含由五个数值的向量 x，且这五个值分别为 10.4，5.6，3.1，6.4 和 21.7，R 中的命令为

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

这是一个用函数 c() 完成的赋值语句。这里的函数 c() 可以有任意多个参数，而它的值则是一个把这些参数首尾相连形成的一个向量¹。

在 R 环境里面，即使是表达式中的标量也是被当作长度为1的向量。

注意一下赋值符号 (<-)，它实际上包括两个字符，即 < (“ 小于号 ”) 和 - (“ 负号 ”)。这两个字符在方向上要求严格一致²并且 `指向`被表达式赋值的对象。在许多情况下，= 可以代替使用。赋值也可以用函数 assign() 实现。下面的命令和前面的赋值命令等价：

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

我们常用的赋值符 <- 可以看作是该命令一个语义上的缩写。

当然，还可以从另外一个方向上赋值。用下面的语句，可以完成上面同样的赋值工作

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

如果一个表达式是一个完整的命令，那么它的值将会被显示在屏幕上 **并且不能被别的对象访问**³。因此，如果我们运行语句

```
> 1/x
```

五个数的倒数将会在终端显示 (注意，x 的值没有改变)。

进一步的赋值

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

会创建一个含有11个元素的向量 `y`，其中包括两份 `x` 拷贝和位于中间的一个0。

Footnotes

[1] 除了向量格式的参数，对于 list 模式的参数，函数 `c()` 的作用可能会有点差异。具体参见 [Concatenating lists](#).

[2] 译者注：`->`和`<-`是一致的，但`-<`与`>-`就不一致了。

[3] 实际上，在其他命令运行前，它是保存在变量 `.Last.value` 中。译者注：当然你还可以直接在屏幕上拷贝表达式的输出值。

Next: [Generating regular sequences](#), Previous: [Vectors and assignment](#), Up: [Simple manipulations numbers and vectors](#)

2.2 向量运算

在算术表达式中使用向量将会对该向量的每一个元素都进行同样算术运算。出现在同一个表达式中的向量最好是一样的长度。如果他们的长度不一样，该表达式的值将是一个和其中最长的向量等长的向量。表达式中短的向量会被循环使用 (recycled) (可能是部分的元素) 以达到最长的向量的长度。对于一个常数就是简单的重复。利用前面例子中的变量，命令

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

将产生一个新的长度为11的向量 v。它由 2*x 重复2.2次，y 重复一次 1 重复11次得到的向量相加而成。

基本的算术运算符就是常用的 +, -, *, / 和做幂运算的 ^。另外还包括常用的数学函数。如 log, exp, sin, cos, tan, sqrt 等等。这些在教科书上都有所定义。max 和 min 分别给出一个向量的最大值和最小值。函数 range 得到的是一个长度为2的向量，即 c(min(x), max(x))。length(x) 给出向量 x 的元素个数，sum(x) 给出 x 中元素的累加和，以及 prod(x) 得到它们的乘积。两个统计函数就是计算均值的 mean(x) (等价于 sum(x)/length(x)) 和计算样本方差的 var(x)。var(x) 等价于

```
sum((x-mean(x))^2)/(length(x)-1)
```

如果 var() 的参数是一个 n × p 的矩阵，则将该矩阵行与行之间看作是相互独立的 p-变量的样本向量，从而得到一个 p × p 的样本协方差矩阵。

sort(x) 返回一个和 x 长度一样但元素以升序排列的向量；当然，还有其他功能更强大的排序函数 (如可以随意排列的 order() 和 sort.list() 等)。注意 max 和 min 将会给出它们参数向量中的最大和最小值。在同时给予多个向量的情况下，这两个函数会把参数向量合并成一个向量处理。并行 (parallel) 求解最大和最小值的函数 pmax 和 pmin 将会返回一个和最长的参数长度一致的向量。该向量每一个元素就是同一位置上的所有输入向量 (即函数的参数) 元素的最大 (最小) 值。大多数情况下，用户并不关心一个数值向量中的“数值”到底是整数，实数，还是复数。R 环境内部的计算是以双精度的实数或者双精度的复数 (在输入数据是复数的情况下) 实现的。

如果要处理复数，应该给出明确的复数部分。因此

```
sqrt(-17)
```

将会给出 NaN 和一个警告，但是

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

就会以复数形式计算。

[Generating regular sequences](#): 正则序列的产生

Next: [Logical vectors](#), Previous: [Vector arithmetic](#), Up: [Simple manipulations numbers and vectors](#)

2.3 正则序列的产生

R 有一系列产生常用数列的工具。如 1:30 等价于向量 `c(1, 2, ..., 29, 30)`。在表达式中，冒号优先级最高，因此 `2*1:15` 等价于 `c(2, 4, ..., 28, 30)`。读者可以认真将 `n <- 10`，`1:n-1`，`1:(n-1)` 相互比较一下。

`30:1` 形式的句法（construction）可用来产生一个逆向的数列。

函数 `seq()` 是产生数列中最为常用的工具。它有五个参数，仅仅部分参数需要每次都设定。起始的两个参数，表示一个数列的首尾。如果只是给定这两个值，则和冒号运算符的结果完全一样了。如 `seq(2,10)` 等价于 `2:10`。

`seq()` 和其他许多 R 函数的参数一样都可以用命名方式给定。在这情况下，参数的顺序可以是任意的。这样，前两个参数就可以用 `from=value` 和 `to=value` 方式设定；因此 `seq(1,30)`，`seq(from=1, to=30)` 和 `seq(to=30, from=1)` 同 `1:30` 完全一样。`seq()` 接下来的两个参数是 `by=value` 和 `length=value`；它们分别表示这个数列的步长和长度。如果二者没有设定，那么默认值就是 `by=1`（步长为1）。

例如

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

将向量 `c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0)` 赋给 `s3`。类似的是，

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

将在 `s4` 中产生一样的向量。

第五个参数是 `along=vector`。使用这个参数时，必须是唯一的¹，它可以创建数列 `1, 2, ..., length(vector)`，或者是空数列（在向量 `vector` 为空时）。

还有一个相关的函数是 `rep()`。它可以用各种复杂的方式重复一个对象。最简单的方式就是

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

这种方式先把 `x` 的完整拷贝五次，保持 `x` 的数列顺序，逐一放在 `s5` 中。另一种有用的方式是

```
> s6 <- rep(x, each=5)
```

这种放松把 x 中的每个元素都重复五次，然后将重复五次的元素逐一放入。

Footnotes

[1] 译者注：我加过其他参数，也运行好好的。问题的关键在于，最终产生的数列长度和length(*vector*)一致。

Next: [Missing values](#), Previous: [Generating regular sequences](#), Up: [Simple manipulations numbers and vectors](#)

2.4 逻辑向量

和数值向量一样，R 允许操作逻辑向量。逻辑向量元素可以被赋予的值有 TRUE，FALSE 和 NA (“不可得到”，见下一小节)。前两个值可以分别简写为 T 和 F。注意 T 和 F 仅仅默认设置为 TRUE 和 FALSE 的变量，因此不是系统保留字 (reserved word)，可以被用户重写。正因为这样，你应该尽量使用 TRUE 和 FALSE。逻辑向量可以由 条件式 (conditions) 产生。例如

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

temp 是一个长度和 x 一致的向量。它的元素 FALSE 表示 x 的元素 不吻合控制条件而 TRUE 则相反。

R 逻辑运算符是 <，<=，>，>=，以及判断严格相等的 == 和判断不等的 !=。此外，如果 c1 和 c2 是逻辑不等式，那么 c1 & c2 是它们的交集运算 (“与”)，c1 | c2 是并集运算 (“或”)，!c1 是 c1 的非运算。在常规的算术运算中采用逻辑变量，它们将会被 强制转换成数值变量，FALSE 变成 0，TRUE 变成 1。但是有些情况下，逻辑向量和它们强制转换的数值向量不等价，具体的例子可以看下一小节。

Next: [Character vectors](#), Previous: [Logical vectors](#), Up: [Simple manipulations numbers and vectors](#)

2.5 缺省值

在一些情况下，向量的元素可能有残缺。当一个元素或者值在统计的时候“不可得到”（not available）或者“值丢失”（missing value），相关位置可能会被保留并且赋予一个特定的值 NA。任何含有 NA 数据的运算结果都将是 NA。这样假设的道理很简单，如果一次操作的数据都是残缺的，那么结果也必然是不可预料的，因此也是不可得到的。

函数 `is.na(x)` 返回一个和 `x` 同样长度的向量。它的某个元素值为 TRUE 当且仅当 `x` 中对应元素是 NA。

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

特别要注意的是逻辑表达式 `x == NA` 和 `is.na(x)` 完全不同。因为 NA 不是一个真实的值而是一个符号以表示某个量是不可得到的。因此 `x == NA` 得到的是一个长度和 `x` 一致的向量。它的 *所有* 元素的值都是 NA 因为该逻辑表达式本身不完整，因此也是不可判断的。

还要注意数值计算会产生第二种“缺省”值，也称为 *非数值*（Not a Number），NaN，值。例如

```
> 0/0
```

或者

```
> Inf - Inf
```

二者都得到 NaN，这是因为它们的结果都不能显式的定义。

总之，对于 NA 和 NaN 用 `is.na(xx)` 检验都是 TRUE。为了区分它们，`is.nan(xx)` 就只对是 NaN 元素显示 TRUE。当字符向量以没有引号的形式打印时^[1]，缺省值可能会以 `<NA>` 形式显示。

Footnotes

[1] 译者注：我没遇到过这种情况，所以也不知有没有译错。原文为“when character vectors are printed without quotes”

Next: [Index vectors](#), Previous: [Missing values](#), Up: [Simple manipulations numbers and vectors](#)

2.6 字符向量

在 R 中，常常会用到字符量和字符向量，如图上的标注。在需要它们的时候，可以用双引号作分割符，如 "x-values", "New iteration results"。

字符串输入的时候既可以用双引号 (") 又可以用单引号 (')，但是打印的时候则采用双引号 (有时根本不用引号)。它们采用 C 语言形式的转义控制序列 (escape sequences)，用 \ 表示转义字符，所以输入 \\ 将会得到 \\ 的输出，而想插入 " 时则要输入 \。其他有用的转义字符如 \n (换行)，\t (制表符) 和 \b (退格键) 等等。

通过函数 c() 可以把几个字符向量连接成一个字符向量；这种用法的例子会常常出现的。函数 paste() 可以有任意多的参数，并且把它们一个接一个连成字符串。这些参数中的任何数字都将被显式地强制转换成字符串，而且以同样的方式在终端显示。默认的分隔符是单个的空格符，不过这可以被指定的参数修改。参数 sep=string 就是将分隔符换成 string，这个参数还有可能为空。

例如

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

使得 labs 变成一个字符向量。

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

特别要注意一下这里短列表发生了循环使用；因此 c("X", "Y") 重复了5次以吻合 1:10 [1](#)。

Footnotes

[\[1\]](#) paste(..., collapse=ss) 会在每个结果生成的字符串元素后面加上 ss。R 有更多的工具进行字符处理，参见 sub 和 substring 的帮助文档。

Next: [Other types of objects](#), Previous: [Character vectors](#), Up: [Simple manipulations numbers and vectors](#)

2.7 索引向量；选择和修改一个数据集的子集

一个向量的子集 (subset) 元素可以通过向量名后面的方括号中 加入 *索引/向量* 得到。如果一个表达式的结果是向量，则 我们可以直接在表达式的 末尾方括号中加入索引向量以 得到结果向量的子向量 (如果有的话)。

这种索引向量可以采用下面四种方式的任何一种。

1. 逻辑向量。这种情况下，索引向量必须 和被挑选元素的向量长度一致。向量中对应索引向量为 TRUE 的元素将会被选中，而那些对应 FALSE 的元素则被忽略。例如

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

这将创建(或重建)一个含 x 中非缺省且次序不变的元素的对象 y。注意，如果 x 含有缺省值，y 在长度上将会比 x 短。同样

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

将创建一个对象 z 并且把向量 x+1 的值赋给它，其中要求 x 中对应的元素 既非缺省又是正值。

2. 正整数向量。这种情况下，索引向量必须是 {1, 2, ..., length(x)} 的子向量。索引向量中索引对应的元素将会被选中，并且在结果向量中的次序和索引向量中的次序一致。这种索引向量可以是任意长度的，结果向量的长度 和索引向量完全一致。如 x[6] 表示 x 的第六个元素，此外

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

选择 x 的前10个元素 (我们假定 length(x) 长度 小于 10)。同样 (看上去好像不可能的事情)

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

会产生一个长度为 16，由 "x", "y", "y", "x" 重复4次而构成的向量。

3. 负整数向量。这种索引向量 指定被*排除*的元素而不是包括进来¹。因此

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

将 x 除开始五个元素外的其他元素都赋给 y。

4. 字符串向量。这可能仅仅用于一个对象可以用 names 属性来识别他的元素。这种情况下，名字向量的子向量可以像上面第二条提到的正整数标签一样使用。

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

名字索引 (name indices) 相比数值索引 (numeric indices) 的好处就是容易记。该用法在后面的数据框 (data frames) 操作中，优势最为明显。

索引表达式同样可以出现在赋值操作的接受端。在这种情况下，赋值操作仅仅发生在这些索引指定的向量元素中。表达式必须以 `vector[index_vector]` 的形式出现，其中向量名字可以用任何表达式代替。

被赋值的向量必须吻合索引向量的长度，特别在逻辑向量中它的长度必须和被建索引的向量长度一致。

例如

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

将会用0替换 x 中所有的缺省值，而

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

和下面式子等价。

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

Footnotes

[1] 译者注：索引向量中，不可以同时出现正整数和负整数

Previous: [Index vectors](#), Up: [Simple manipulations numbers and vectors](#)

2.8 其他类型的对象

向量是 R 里面最重要的对象，但还有其他几种类型的对象将在后面的内容中遇到。

矩阵（matrix）或者更为一般的**数组**（array）是多维的广义向量。实际上，它们就是向量，而且可以同时被两个或者更多个索引引用，并且以特有的方式显示出来。See [Arrays and matrices](#).

因子（factor）为处理分类数据提供一种有效的方法。See [Factors](#).

列表（list）是一种泛化（general form）的向量。它没有要求所有元素是同一类型，许多时候就是向量和列表类型。列表为统计计算的结果返回提供了一种便利的方法。See [Lists](#).

数据框（data frame）是和矩阵类似的一种结构。在数据框中，列可以是不同的对象。可以把数据框看作是一个行表示观测个体并且（可能）同时拥有数值变量和分类变量的`数据矩阵'。许多实验数据都可以很好的用数据框描述：处理方式看作分类变量而响应情况当作数值变量。See [Data frames](#).

函数（function）是可以保存在项目工作空间的 R 对象。该对象就提供了一个简单而又便利的 R 扩充方法。See [Writing your own functions](#).

Next: [Factors](#), Previous: [Simple manipulations numbers and vectors](#), Up: [Top](#)

3 对象及它们的模式和属性

[The intrinsic attributes mode and length](#): 内在属性：模式和长度

[Changing the length of an object](#): 改变对象的长度

[Getting and setting attributes](#): 访问和设置属性

[The class of an object](#): 对象的类

Next: [Changing the length of an object](#), Previous: [Objects](#), Up: [Objects](#)

3.1 Intrinsic attributes: 内在属性：模式和长度

R 操作的实体在技术上来说就是 *对象* (object)。如实数或复数向量，逻辑向量和字符串向量等对象属于“原子” (atomic) 结构的对象，因为它们的元素都是一样的类型或模式¹，R 的对象类型包括数值型² (numeric)，复数型 (complex)，逻辑型 (logical)，字符型 (character) 和原味型 (raw)。

向量必须保证它的所有元素是一样的模式。因此任何给定的向量必须明确属于逻辑性，数值型，复数型，字符型或者原味型。(这里有个特定的例外就是“值”为 NA 的元素，实际上有好几种形式的 NA)。注意空向量也有自己的模式。例如，空的字符串向量将会被显示 character(0) 和空的数值向量显示的 numeric(0)。

R 同样操作被称为列表的对象。这种对象在 R 里面是一种列表 (list) 模式。这些是可以为任何模式的对象的有序序列。列表被认为是一种“递归”结构而不是原子结构因为它们的元素可以用它们各自的方式单独列出。

另外两种递归结构是函数 (function) 和表达式 (expression)。构成 R 系统一部分的函数对象以及其他类似的用户定义的函数对象都将在后面的内容中深入讨论。表达式对象是 R 的高级部分，不是本文档的重点，我们只是在讨论 R 统计建模中的公式 (formulae) 时间间接的提一下。

一个对象的模式 (mode) 是该对象基本要素的类型。这是专门用来描述一个对象“特征” (property) 的术语。另外一个所有对象都有的特征是长度 (length)。函数 mode(object) 和 length(object) 可用于任何数据对象以得到其模式和长度³。

一个对象更进一步的特征常常通过 attributes(object) 得到，具体参见 [Getting and setting attributes](#)。正因为这样，模式和长度又叫做一个对象的“内在属性”。例如，如果 z 是一个长为 100 的复数向量，那么命令 mode(z) 就会得到字符串 "complex" 而 length(z) 对应的是 100。

R 可以在任何需要的时候对模式进行转换 (当然，有些时候没有必要)。例如

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

我们可以进行如下转换

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


这样，`digits` 就是一个字符向量 `c("0", "1", "2", ..., "9")`。我们可以再一次强制转换，或者说模式改变，以重建数值向量：

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

现在 `d` 和 `z` 就一样了^[4]。有一系列类似 `as.something()` 的函数，这些函数主要用于对象模式的强制转换，或者赋予某个对象一些先前没有的功能。读者可以参考不同的帮助文件以熟悉它们。

Footnotes

[1] 译者注：实际上 R 已经有自己独立的函数 `typeof()`，仍然保留模式的概念主要是为了和 S 兼容。

[2] 数值型模式实际上是两种独立模式的混合说法，即整数型（integer）和双精度型（double）。具体可以参加手册。

[3] 注意 `length(object)` 有时会返回一些没有意义的信息，如 `object` 是一个对象的时候。

[4] 一般情况下，从数值到字符的强制转换，然后再转回不总是可逆的。因为在数字的字符表示时有近似转换（roundoff）的问题。

Next: [Getting and setting attributes](#), Previous: [The intrinsic attributes mode and length](#), Up: [Objects](#)

3.2 改变对象长度

一个“空”的对象仍然有其模式的。例如

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

创建了一个数值模式的空向量结构 e。相似的是，character()是一个空的字符向量，等等。一旦一个任意长度的对象被创建，新元素可以通过给定一个在先前索引范围外的索引值^[1]而加入。因此

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

将创建一个长度为3的向量 e(此时，前两个元素都是 NA)。这可以用于任何数据结构，并且提供了一种添加和第一个位置的对象模式一致的对象的方式。

这种自动调整对象长度的方法是经常使用的，如等待输入的函数 scan()。(see [The scan\(\) function](#).)

相反，删减一个对象的大小只需要用赋值操作。因此，如果 alpha 是一个长度为10的对象，那么

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

创建一个由偶数索引位值上的元素构成的长度为5的对象(当然，老的索引将会被抛弃)。我们可以仅仅保留起始的三个值

```
> length(alpha) <- 3
```

并且一个向量可以用同样的办法扩充（扩充部分用缺省值）。

Footnotes

[1] 译者注：这里要小心一点。正整数，负整数，大于或小于原来的索引范围的值都有不同含义。

Next: [The class of an object](#), Previous: [Changing the length of an object](#), Up: [Objects](#)

3.3 读取和设置属性

函数 `attributes(object)` 给出对象当前定义的非内在属性（non-intrinsic attributes）的列表。函数 `attr(object, name)` 可以用来选择特定的属性。这些函数很少使用，只是在一些非常特殊的情况下，如为特定的目的而设计一些新的属性时才使用。但是，这个概念是最重要的。

对属性进行赋值和删除操作必须额外小心，因为它们是 R 对象系统的主要部分。

当它用于一个赋值操作的左边是，它既可用于关联 *object* 的新属性也可用于改变一个已经存在的属性。例如

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

允许 R 把 *z* 当作一个 10×10 的矩阵。

Previous: [Getting and setting attributes](#), Up: [Objects](#)

3.4 对象的类

R 里面的所有对象都属于一个类 (class) , 可以通过函数 `class` 查看。对于简单的向量, 就只是对应的模式 "numeric", "logical", "character" 或者 "list", 但是 "matrix", "array", "factor" 和 "data.frame" 就可能是其他值。

引入对象的类属性有利于 面向对象风格的¹R 编程。比如说, 如果一个对象有 "data.frame" 类, 那么它将会以一种特定的方式显示, 函数 `plot()` 也会以特定的方式显示它的图形并且 其他相关泛型函数² (generic function , 如`summary()`等) 就像对某个类一样对这个对象有所响应。

可以用函数`unclass()` 临时去掉一个对象的类作用。 举例说, 如果 `winter` 有一个 "data.frame" 的类, 那么

```
> winter
```

将会以和矩阵类似的数据框显示, 而

```
> unclass(winter)
```

就像一个普通的列表一样打印数据。仅仅在一些非常特殊的情况下, 你才需要使用这个函数。当然, 如果你想深入学习 类和泛型函数, 那就可能常常用到了。

泛型函数和类将会在 [Object orientation](#) 部分进一步讨论, 不过比较简略。

Footnotes

[1] 包 `methods` 提供了一种不同的方式 调用`正式' 或者`S4'中的类。

[2] 译者注: 如果学过 Java 或者 C++ 泛型, 理解会更深。

Next: [Arrays and matrices](#), Previous: [Objects](#), Up: [Top](#)

4 有序因子和无序因子

因子 (factor) 是一个对同长的其他向量元素 进行分类 (分组) 的向量对象。R 同时提供 有序 (ordered) 和 无序 (unordered) 因子。而 “真正” 使用因子是在设计模型公式的时候 (see [Contrasts](#))，这里我们只是看一些特别的例子。

4.1 一个特别的例子

假定我们有一份来自澳大利亚所有州和行政区30个税务会计师信息的 样本¹ 以及他们各自所在地的州名。州名以字符串向量的形式 保存在 state 中

```
> 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")
```

注意在字符向量中，“有序”意味着以字母排序的。

因子可以简单地用函数 factor() 创建：

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

函数 print() 处理因子和其他对象 有点不同：

```
> 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() 可以用来得到因子的水平 (levels)。

```
> levels(statef)
[1] "act" "nsw" "nt" "qld" "sa" "tas" "vic" "wa"
```

[The function tapply\(\) and ragged arrays](#): tapply()函数和不规则数组

[Ordered factors](#): 有序因子

Footnotes

[1] 读者注意一下，澳大利亚有8个州和行政区，分别是 Australian Capital Territory, New South Wales, the Northern Territory, Queensland, South Australia, Tasmania, Victoria 和 Western Australia。

Next: [Ordered factors](#), Previous: [Factors](#), Up: [Factors](#)

4.2 函数 `tapply()` 和不规则数组

沿用前面的例子，假定我们有这些税务会计师的收入信息并且保存在另外一个向量中(适当的货币单位)

```
> 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)
```

为计算样本中每个州的平均收入，我们可以用函数 `tapply()`：

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

这将给出一个均值向量。各个元素都用对应的水平名字标记了。

```
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()` 将一个函数（这里是 `mean()`）用于第二个参数 [1](#)（这里是 `statef`）定义于第一个参数（这里是 `incomes`）上得到的所有组。此时，各个组的数据好像是独立的向量。得到的结果向量长度和因子的水平数一致。读者可以通过帮助文档得到更多的信息。

假定我们进一步想计算各个州的标准误（standard error）[2](#)。我们写了一个 R 函数来计算任一给定向量的标准误。既然已经有内置函数 `var()` 计算样本方差，这个函数可以在一行写完，并且有一个参数等待赋值：

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

(编写函数可以参考后面的内容 [Writing your own functions](#)。这里的写的标准误演示函数其实是没有必要的，因为 R 有一个内置计算标准误的函数 `sd()`。) 赋值后，标准误被计算出来

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

值分别为

```
> 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，你可能想计算州平均收入的95%信度区间。于是，你可以再次使用 `tapply()` 和能得到样本量的函数 `length()`，以及能得到t-分布的分位数的函数 `qt()`。（你需要研究一下 R 为t-检验设计的工具。）

函数 `tapply()` 还可以用来处理一个由多个分类因子决定的向量下标组合。例如，我们可能期望通过州名和性别把这税务会计师分类。不过，就在上面最简单的情况中(仅仅一个变量)，我们也可以这样考虑这个问题（复杂因子组合时一样处理）。向量中的值可以根据因子中不同的水平分成许多组。函数就是独立的用于这些组。得到的值是这些函数结果的向量，并且以因子的水平属性标记。

因为子类的大小是不规则的，所以向量和作为标签的因子的组合就是我们偶尔会提及的 *不规则数组*（ragged array）一个特例罢了。当子类大小是一致的时候，索引最有效，正如我们在下一节中所看到的一样。

Footnotes

[1] 注意，当第二个参数不是因子时，函数 `tapply()` 同样有效，如 `tapply(incomes, state)`。这对一些其他函数也是有效，因为必要时 R 会用 `as.factor()` 把参数强制转换成因子。

[2] 译者注：注意和“标准差”的不同

Previous: [The function tapply\(\) and ragged arrays](#), Up: [Factors](#)

4.3 有序因子

因子的水平是以字母顺序排列的，或者显式地在 factor 中指定。

有时候因子的水平有自己的自然顺序的并且有意义的。我们记录下来可能用于进一步的统计。函数 `ordered()` 就是用来创建这样的有序因子。在其他方面，函数 `ordered()` 和 `factor` 基本完全一样。大多数情况下，有序和无序因子的唯一差别在于前者显示的时候反应了各水平的顺序。在线性模型拟合的时候，这种差异的意义比较明显。

Next: [Lists and data frames](#), Previous: [Factors](#), Up: [Top](#)

5 数组和矩阵

[Arrays](#): 数组

[Array indexing](#): 数组索引

[Index arrays](#): 索引数组

[The array\(\) function](#): array()函数

[The outer product of two arrays](#): 两个数组的外积

[Generalized transpose of an array](#): 数组广义转置

[Matrix facilities](#): 矩阵工具

[Forming partitioned matrices](#): 分块矩阵

[The concatenation function c\(\) with arrays](#): 数组串连函数c()

[Frequency tables from factors](#): 因子分布表

Previous: [The concatenation function c\(\) with arrays](#), Up: [Arrays and matrices](#)

5.10 因子的频率表

单个的因子会把各部分数据分成不同的组。相似的，一对因子可以实现交叉分组等。函数 `table()` 可以从等长的不同因子中计算出频率表。如果有 k 个因子参数，那么结果将是一个 k -维的频率分布数组。

假定 `statef` 是一个设定 数据向量元素个体的所在州的因子，那么下面的赋值

```
> statefr <- table(statef)
```

将会赋予 `statefr` 在一个样本中各种状态的频率分布表。这些频率会被排序且以 因子的水平特性标记。等价但 有点烦琐实现方式如下

```
> statefr <- tapply(statef, statef, length)
```

进一步假设 `incomef` 是一个 按适当定义的 “收入阶层” 对数据向量元素进行分类的因子。它可以用函数 `cut()` 定义：

```
> factor(cut(incomes, breaks = 35+10*(0:7))) -> incomef
```

然后计算频率的二维表：

```
> table(incomef, statef)
      statef
incomef act nsw nt qld sa tas vic wa
(35,45]  1  1  0  1  0  0  1  0
(45,55]  1  1  1  1  2  0  1  3
(55,65]  0  3  1  3  2  2  2  1
(65,75]  0  1  0  0  0  0  1  0
```

同样的方法也可以得到高维的频率分布表。

Next: [Array indexing](#), Previous: [Arrays and matrices](#), Up: [Arrays and matrices](#)

5.1 数组

数组可以看作是带有多个下标的类型相同的元素的集合，如数值型。R 有一些简单的工具创建和处理数组，特别是矩阵。

维度向量 (dimension vector) 是一个元素取正整数的向量。如果它的长度为 k ，那么该数组就是 k -维的，例如一个矩阵是 2-维数组。数组中元素的下标可以从 1 一直标到 维度向量中对应元素的值。

向量只有在定义了 *dim* 属性后才能作为数组在 R 中使用。假定， z 是一个含 1500 个元素的向量。那么

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

对 *dim* 属性的赋值使得该向量成一个 $3 \times 5 \times 100$ 的数组。

其他函数，如 `matrix()` 和 `array()` 可以更直观更容易地定义，具体参见 [The array\(\) function](#).

数据向量的值在数组中的排列顺序采用 FORTRAN 的数组元素次序，即“按列次序”，也就是说第一下标变化最快，最后下标变化最慢。

假定数组 a 的维数向量是 `c(3,4,2)`，则 a 中有 $3 * 4 * 2 = 24$ 元素，依次为 $a[1,1,1]$, $a[2,1,1]$, ..., $a[2,4,2]$, $a[3,4,2]$ 。

数组可以是一维的：这种数组的处理和向量完全一致(包括屏幕显示)，只是有时会导致一些混乱。

Next: [Index arrays](#), Previous: [Arrays](#), Up: [Arrays and matrices](#)

5.2 数组索引以及数组分割

数组元素可以通过给定 数组名及其后方括号中用逗号 隔开的下标访问。

更为一般的是，数组分割可以通过 在下标位置给定一系列索引/向量实现；不过，如果某个位置上给定的索引/向量为空，则该下标处所有可能值都会被取到。

延续前面的例子，`a[2,,]` 是一个 4×2 的数组。它的维度向量为 `c(4,2)`，数据向量依次包括下面的值

```
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])
```

`a[,,]`表示整个数组。这和忽略下标直接使用 `a` 效果是一样的。

对于数组 `Z`，用 `dim(Z)` 可以对 该数组维度向量进行显式的访问(可以放在在赋值的任何一边)。

还有，如果一个数组只给了一个下标或索引/向量，那么只有数据向量中对应的值才会被访问；在这种情况下，维度向量会被忽略的。但是，如果单个索引不是一个向量而是一个数组，可能就不是这样了，具体可以看下面的讨论。

[Index arrays](#): 索引数组

[The array\(\) function](#): `array()` 函数

Next: [The array\(\) function](#), Previous: [Array indexing](#), Up: [Arrays and matrices](#)

5.3 索引数组

和用于下标位置的索引向量一样，可以根据 *索引/数组* 去给数组中不规则的元素集合赋值 或者将数组中特定的元素返回到一个向量中。

用矩阵作为例子使这个过程变的更容易理解。在一个二维索引 数组中，索引矩阵 μ 可以假定含有两列及 任意多行。索引矩阵中的元素就是 数组的行列索引。假定 我们有一个 4×5 的数组 X ，我们可以 做如下的事情：

以向量的格式取出 $X[1,3]$, $X[2,2]$ 和 $X[3,1]$ ，
在数组 X 中用0替换这些元素。

在这个例子中，我们需要一个 3×2 下标数组，见 下面的例子。

```
> x <- array(1:20, dim=c(4,5)) # 产生一个  $4 \times 5$  的数组。
```

```
> 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          # i 是一个  $3 \times 2$  的索引矩阵。
```

```
  [,1] [,2]
[1,]  1  3
[2,]  2  2
[3,]  3  1
```

```
> x[i]          # 提取这些元素。
```

```
[1] 9 6 3
```

```
> x[i] <- 0      # 用0替换这些元素。
```

```
> 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
```

```
>
```

下面是一个不太明显的例子，假定我们要从一个双因子blocks(b水平)和varieties (v 水平) 区组化设计实验中得到一个设计矩阵。进一步假定 实验进行了n次。我们可以按 下面的步骤进行：

```
> 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)
```

我们可以用下面的办法创建关联矩阵 N。

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

但是最简单最直接的办法是使用函数 table()：

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

Next: [The outer product of two arrays](#), Previous: [Index arrays](#), Up: [Arrays and matrices](#)

5.4 The array() 函数

除了用设定一个向量的 dim 属性的方法来构建数组，它还可直接通过函数 array 将向量转换得到，具体格式为

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

假定向量 h 有24个或更少的数值，那么命令

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

就会利用 h 在 Z 中创建一个 $3 \times 4 \times 2$ 的数组。如果 h 的长度正好是24，那么和下面的命令等价

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

如果 h 的长度小于 24，它的元素将会被循环使用直到长度为 24 (see [The recycling rule](#))。一个极端但又普遍的例子是

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

这样就会使得 Z 是一个所有值都是0的数组。

此时，dim(Z) 表示维度向量 c(3,4,2)，Z[1:24] 表示数据向量（和 h 中一样）。空下标的Z[] 和没有下标的 Z 都表示整个数组。

数组可以用于算术表达式中，并且结果就是一个基于数据向量的对应元素而得到的数组。所有操作数的属性 dim 必须一致，而这个属性同样也是最终结果的维度向量。因此，如果 A, B 和 C 是相似矩阵，那么

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

D 同样是一个相似矩阵。它的值是给定的操作数对应元素计算所得。但是对于数组和向量混合运算还是要小心一点。

[The recycling rule](#): 循环使用原则

Previous: [The array\(\) function](#), Up: [The array\(\) function](#)

5.4.1 向量和数组混合运算以及循环使用原则

向量和数组间对应元素混合运算的确切规则有点 诡异，并且很难在参考文献中 找到权威的说法。根据自己的经验，我们列出下面的一些比较可靠的说明。

表达式运算是从左到右进行的。

短的向量操作数将会被循环使用 以达到其他操作数的长度。

只要短的向量和数组在一起^[1]，数组必须有一样的属性dim，否则就给出错误信息。

向量操作数比矩阵或者数组操作数长时会引起错误。

如果数组结构给定，同时也没有关于向量的错误信息和强制转换操作，结果将是一个和它的数组操作数属性 dim 一致的数组。

Footnotes

[1] 译者注：原文为 “ As long as short vectors and arrays only are encountered ”，我不明确其中为什么用 “ only ” 这个词

Next: [Generalized transpose of an array](#), Previous: [The array\(\) function](#), Up: [Arrays and matrices](#)

5.5 数组的外积

数组一个非常重要的运算就是外积运算 (outer product)。如果 a 和 b 是两个数值数组，它们的外积将是这样的一个数组：维度向量通过连接两个操作数的维度向量(顺序非常的重要)得到；数据向量则由 a 的数据向量元素和 b 的数据向量元素的所有可能乘积得到。外积是通过特别的操作符 `%o%` 实现：

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

一种备选的方案是，

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

命令中的乘法函数可以被这两个变量的任意其他函数代替。例如，我们想研究函数 $f(x; y) = \cos(y)/(1 + x^2)$ 在 R 定义的向量 x 和 y 形成的格子平面 (regular grid) 上的特征，可以按下面的步骤进行：

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

特别是，两个常规向量的外积是一个双下标的数组 (就是矩阵，最大秩为1)。注意，外积运算不符合交换律。定义你自己的 R 函数可以参考 [Writing your own functions](#).

一个例子：2 × 2 单数字矩阵的行列式

这是一个简单的例子，计算一个 2 × 2 的矩阵 $[a, b; c, d]$ 的行列式。该矩阵的所有元素是 0, 1, ..., 9 里面的一个数字。

问题是找出所有这种组合形式的矩阵行列式， $ad - bc$ ，同时以高密度 (high density) 图的形式显示这些行列式值的分布。如果所有数字的选取是独立随机的，这样做实际上会得到所有行列式的概率分布图。

解决这个问题一个巧妙的办法就是两次使用函数 `outer()`：

```
> d <- outer(0:9, 0:9)
> fr <- table(outer(d, d, "-"))
```

```
> plot(as.numeric(names(fr)), fr, type="h",  
      xlab="Determinant", ylab="Frequency")
```

注意，对频率表的属性 `names` 强制转换成 数值以表示行列式值的范围。“显式”解决这个问题是用 `for` 循环语句。这个将在 [Loops and conditional execution](#) 部分讨论，不过效率太低 以至实际中很少采用。

还有一点比较奇怪的是，每20个这样的矩阵就有 1个值是特异的。

Next: [Matrix facilities](#), Previous: [The outer product of two arrays](#), Up: [Arrays and matrices](#)

5.6 数组的广义转置

函数 `aperm(a, perm)` 可以用来重排一个数组 `a`。参数 `perm` 可以是 $\{1, \dots, k\}$ 的一个排列，其中 k 是 `a` 的下标数目。这个函数将产生一个和 `a` 大小一致的数组，不过旧的维度 `perm[j]` 将会变成第 `j` 个维度。这种操作实际上是对矩阵的一种广义转置。实际上，如果 `A` 是一个矩阵(双下标数组)，那么 `B`

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

仅仅是 `A` 的一个转置。这种情况下，有一个简单的函数 `t()` 可以使用。因此，我们可以用命令 `B <- t(A)` 代替上面的语句。

Next: [Forming partitioned matrices](#), Previous: [Generalized transpose of an array](#), Up: [Arrays and matrices](#)

5.7 矩阵工具

正如上面所说，矩阵仅仅是一个双下标的数组。但是，它非常的重要，以至于需要单独讨论。R 包括许多仅对矩阵操作的操作符和函数。例如上面提到的 `t(X)` 就是矩阵的转置函数。函数 `nrow(A)` 和 `ncol(A)` 将会分别返回 矩阵A 的行数和列数。

[Multiplication](#): 相乘

[Linear equations and inversion](#): 线性方程和求逆

[Eigenvalues and eigenvectors](#): 特征值和特征向量

[Singular value decomposition and determinants](#): 奇异值分解和行列式

[Least squares fitting and the QR decomposition](#): 最小二乘法拟合和 QR 分解

Next: [Linear equations and inversion](#), Previous: [Matrix facilities](#), Up: [Matrix facilities](#)

5.7.1 矩阵相乘

操作符 `%*` 用于矩阵相乘。 $n \times 1$ 或者 $1 \times n$ 矩阵在情况允许的时候可以作为一个长度为 n 的向量处理。反之，向量在矩阵相乘表达式中出现时可能会被自动转换成与矩阵对应的行或者列向量（有时候，会比较含糊，后面会有所论述）。

如果 A 和 B 大小一样的方阵，那么

```
> A * B
```

将是一个对应元素乘积的矩阵，并且

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

同样是一个矩阵积。如果 x 是一个向量，那么

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

是一个二次型（quadratic form）[1](#)。

函数 `crossprod()` 可以完成“矢积”（crossproduct）运算，也就是说 `crossprod(X, y)` 和 `t(X) %*% y` 等价，但是在运算上更为高效。如果 `crossprod()` 第二个参数忽略了，它将默认和第一个参数一样。

函数 `diag()` 的含义依赖于它的参数。当 v 是一个向量时，`diag(v)` 返回以该向量元素为对角元素的对角矩阵。当 M 是一个矩阵时，`diag(M)` 返回 M 的对角元素。这和 Matlab 中 `diag()` 的用法完全一致。不过有点混乱的是，如果 k 是单个值 [2](#)，那么 `diag(k)` 的结果就是 $k \times k$ 的方阵！

Footnotes

[\[1\]](#) 注意 `x %*% x` 就不明确，因为既可表示 $x'x$ 也可表示 xx' ，其中 x 是列形式。在这种情况下，小矩阵好像比较符合上面的解释，因此标量 $x'x$ 是结果（译者注：这里我也有点困惑，只是字面上翻译了）。矩阵 xx' 可以通过 `cbind(x) %*% x` 或者 `x %*% rbind(x)` 计算因为 `rbind()` 或者 `cbind()` 的结果都是矩阵。但是，计算 $x'x$ 和 xx' 最好的办法是 `crossprod(x)` 或者 `x %o% x`。

[2] 译者注：正整数和小数都可以，对于正实数会自动去掉小数部分

Next: [Eigenvalues and eigenvectors](#), Previous: [Multiplication](#), Up: [Matrix facilities](#)

5.7.2 线性方程和求逆

求解线性方程组是矩阵乘法的逆运算。当下面的命令运行后，

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

如果仅仅给出 A 和 b，那么 x 就是该线性方程组的根。在 R 里面，用命令

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

求解线性方程组，并且返回 x (有一些精度丢失)。注意在线性代数里面表示为 $x = A^{-1} \%*\% b$ ，其中 A^{-1} 表示 A 的逆 (inverse)。矩阵的逆可以用下面的命令计算，

```
solve(A)
```

不过这一般很少用到。在数学上，用直接求逆的办法解 $x <- \text{solve}(A) \%*\% b$ 相比 $\text{solve}(A,b)$ 不仅低效而且有一种潜在的不稳定性。

用于多变量计算的二次型 $x \%*\% A^{-1} \%*\% x$ 可以通过 [1](#) 像 $x \%*\% \text{solve}(A,x)$ 的方式计算得到，而不是直接计算 A 的逆。

Footnotes

[1] 最好的方式当然是用 $A = BB'$ 求解矩阵平方根 B 和利用 A 的 Cholesky 或特征值分解的办法得到 $By = x$ 的解的自乘长度 (squared length)。译者注：这里我还没有完全弄明白，可以参见原文。

Next: [Singular value decomposition and determinants](#), Previous: [Linear equations and inversion](#), Up: [Matrix facilities](#)

5.7.3 特征值和特征向量

函数 `eigen(Sm)` 用来计算矩阵 `Sm` 的特征值 和特征向量。这个函数的返回值 是一个含有 `values` 和 `vectors` 两个分量的列表。命令

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

将把这个列表赋给 `ev`。`ev$val` 表示 `Sm` 的特征值向量，`ev$vec` 则是 相应特征向量构成的一个矩阵。假定我们仅仅需要特征值，我们可以采用如下的命令：

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

`evals` 现在拥有特征向量，而第二个分量 则被抛弃了。如果以下面的表达式作为一个命令，

```
> eigen(Sm)
```

这两个成分连同它们的名字都会被显示。对于大的矩阵，若无必要，最好 不要用下面的表达式 计算特征向量

```
> evals <- eigen(Sm, only.values = TRUE)$values
```

Next: [Least squares fitting and the QR decomposition](#), Previous: [Eigenvalues and eigenvectors](#), Up: [Matrix facilities](#)

5.7.4 奇异值分解和行列式

函数 `svd(M)` 调用任意一个矩阵 `M` 且对 `M` 进行奇异值分解。这包括一个和 `M` 列空间一致的正交列 `U` 的矩阵，一个和 `M` 行空间一致的正交列 `V` 的矩阵，以及一个正元素 `D` 的对角矩阵，如 $M = U \%*\% D \%*\% t(V)$ 。`D` 实际上以对角元素向量的形式返回。`svd(M)` 的结果是由 `d`, `u` 和 `v` 构成的一个列表。

如果 `M` 是一个方阵，那么就不难看出

```
> absdetM <- prod(svd(M)$d)
```

这就计算 `M` 行列式的绝对值。如果在各种矩阵中都需要这种运算，我们可以把它定义为一种 R 函数

```
> absdet <- function(M) prod(svd(M)$d)
```

我们会发现 `absdet()` 是另外一个 R 函数。作为一个零碎但可能很有用的例子，你可能想想写一个用来计算方阵迹（trace）的函数 `tr()`。[提示: 你不需要外在的循环。仔细看一下函数 `diag()`。]

R 有一个计算行列式（包括符号）的内置函数 `det` 和另外一个给出符号和模（对数坐标可选）的函数。

Previous: [Singular value decomposition and determinants](#), Up: [Matrix facilities](#)

5.7.5 最小二乘法拟合和 QR 分解

函数 `lsfit()` 返回最小二乘法拟合 (Least squares fitting) 的结果列表。赋值可以采用如下命令

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

这样就得到最小二乘法拟合结果，其中 y 是观测向量， X 是设计矩阵。更细节的东西可以查看帮助文档，同时参考相关的回归诊断 (regression diagnostics) 函数 `ls.diag()`。注意总的平均值会被自动加入而没有必要显式的加入到 X 的列中。实际上，你在回归分析中可能已经习惯使用 `lm(.)` (see [Linear models](#)) 而不是 `lsfit()`。

另外一个密切相关的函数是 `qr()` 及其相关者。注意下面的赋值情况

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

这些将会计算 y 在 `fit` 的 X 上的正交投影，在 `res` 正交补空间上的投影以及在 `b` 投影上的系数向量。本质上而言，`b` 和 Matlab 反斜线操作符的结果是一致的。

这里没有要求 X 有完整的列秩，因此可能出现冗余。如果出现，就会被自动去掉。

这里提到的方法是原始的底层实现的最小二乘法计算。尽管在一些情况下仍然有效，但是它现在已经被统计模型特征分析的办法所替换，这些内容将会在 [Statistical models in R](#) 中讨论。

Next: [The concatenation function c\(\) with arrays](#), Previous: [Matrix facilities](#), Up: [Arrays and matrices](#)

5.8 用 cbind() 和 rbind() 构建分块矩阵

正如前面所暗示的，可以利用函数 cbind() 和 rbind() 把向量和矩阵拼成一个新的矩阵。概略地说，cbind() 把矩阵横行地合并成一个大矩阵（列方式），而 rbind() 是纵向合并（行方式）。

在命令中

```
> X <- cbind(arg_1, arg_2, arg_3, ...)
```

cbind() 的参数要么是任何长度的向量，要么是列大小一致的矩阵，也就是有一样的行数。结果将是一个合并 *arg_1, arg_2, ...* 的列形成的矩阵。

如果 cbind() 的参数中有一些比其他矩阵参数的列长度短的向量，它们将会被循环使用以吻合矩阵的列长度(在没有矩阵的情况下，吻合其中最长的向量的长度)。

函数 rbind() 对行进行类似的操作。其中任何向量参数都会被当作行向量且可能被循环使用。

假定 X1 和 X2 有一样的行数。下面的命令会把它们的列合并以得到矩阵 X，同时要求起始列都是 1

```
> X <- cbind(1, X1, X2)
```

rbind() 或者 cbind() 返回值常常是矩阵形式。因此，cbind(x) 和 rbind(x) 可能是把向量 x 分别转换成列或者行向量最简单的方法。

Next: [Frequency tables from factors](#), Previous: [Forming partitioned matrices](#), Up: [Arrays and matrices](#)

5.9 对数组实现连接操作的函数c()

需要注意的是，`cbind()` 和 `rbind()` 是考虑 `dim` 特性的连接函数，而函数 `c()` 则不考虑这些数值对象的 `dim` 和 `dimnames` 特性。这一点在有些时候 非常有用。

将一个数组强制转换成简单的向量的标准方式是用函数 `as.vector()`。

```
> vec <- as.vector(X)
```

一个相似效果的办法是采用单参数的 `c()`，就像下面：

```
> vec <- c(X)
```

这两种方法有少许不同，到底采用那种方式 关键是看对结果向量的格式要求(建议使用 前者)。

Next: [Reading data from files](#), Previous: [Arrays and matrices](#), Up: [Top](#)

6 列表和数据框

[Lists](#): 列表

[Constructing and modifying lists](#): 构建和修改列表

[Data frames](#): 数据框

Next: [Constructing and modifying lists](#), Previous: [Lists and data frames](#), Up: [Lists and data frames](#)

6.1 列表

R 的 *列表* (list) 是一个以对象的有序集合构成的对象。列表中包含的对象又称为它的分量 (components)。

分量可以是不同的模式或者类型，如一个列表可以包括数值向量，逻辑向量，矩阵，复向量，字符数组，函数等等。下面是一个例子演示怎么创建列表：

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

分量常常会被编号的 (numbered)，并且可以利用它来访问分量。如果向量 Lst 有四个分量，这些分量可以用 Lst[[1]], Lst[[2]], Lst[[3]] 和 Lst[[4]] 独立访问。如果 Lst[[4]] 是一个有下标的数组，那么 Lst[[4]][1] 就是该数组的第一个元素了。

因为 Lst 是一个列表，所以函数 length(Lst) 给出的仅仅是分量的数目(最高层次的)。

列表的分量可以被命名，这种情况下可以通过名字访问。这时，可以把字符串形式的分量名字放在列表名后面的双中括号中，或者干脆采用下面的形式

```
> name$component_name
```

完成一样的目的。

这种方法非常有用。使你在忘记分量编号的时候，可以很容易找到想要的东西。

因此在上面给定的例子中，

Lst\$name 和 Lst[[1]] 返回结果都是 "Fred"，

Lst\$wife 和 Lst[[2]] 返回的则是 "Mary"，

而 Lst\$child.ages[1] 和 Lst[[4]][1] 返回一样的数字 4。

另外你同样可以在双中括号中使用列表分量的名字，也就是和 Lst\$name 等价的 Lst[["name"]]。当分量的名字保存在另外一个变量中时，这种方法特别的有用。就如下面的例子所示

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

特别要注意一下 `Lst[[1]]` 和 `Lst[1]` 的差别。 `[[...]]` 是用来选择单个元素的操作符，而 `[...]` 是一个更为一般的下标操作符。因此前者得到的是列表 `Lst` 中的第一个对象，并且含有分量名字的命名列表（named list）中分量的名字会被排除在外的。后者得到的则是列表 `Lst` 中仅仅由第一个元素构成的子列表。如果是命名列表，分量名字会传给子列表的。

分量名字可以简写¹，原则是只要能很好的区分所有分量就行。因此 `Lst$coefficients` 可以简写为 `Lst$coe` 而 `Lst$covariance` 可以简写成 `Lst$cov`。

名字向量（记录分量名字的向量）实际上可以简单地看作是列表的一个属性，可以像其他属性一样操作。除了列表，其他的结构对象也可能会给出一个相似的名字属性。

Footnotes

[1] 译者注：当然这里的简写指的是在原来命名的基础上，采用开头连续的几个字母。

Next: [Data frames](#), Previous: [Lists](#), Up: [Lists and data frames](#)

6.2 构建和修改列表

可以通过函数 `list()` 将已有的对象构建成 列表。下面的命令

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

将创建一个含有 m 个分量的列表 `Lst`。它的分量分别是 *object_1*, ..., *object_m* , 分量名则是由 参数 得到 (命名没有特定要求)。如果这些名字 被忽略了, 那么分量只有被编号了。用于创建列表的分量只是原来对象的一份拷贝, 因此 原来的对象没有受到任何影响。

和其他所有被下标索引的对象一样, 列表是可以 扩充分量的。如

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

[Concatenating lists](#): 列表连接

Previous: [Constructing and modifying lists](#), Up: [Constructing and modifying lists](#)

6.2.1 列表连接

当连接函数 `c()` 的参数中有列表对象时，结果就是一个列表模式的对象。它的分量是这些作为参数的列表。

```
> list.ABC <- c(list.A, list.B, list.C)
```

前面提到过连接函数处理向量对象的情况。这种情况下，这些向量对象将会被合并成一个大的的向量结构。此时，所有的其他特性，如 `dim` 等，都会被忽略。

Previous: [Constructing and modifying lists](#), Up: [Lists and data frames](#)

6.3 数据框

数据框 (data frame) 是一个含有 "data.frame" 类的列表。不过，对于 可能列入数据框中的列表有下面一些限制，

分量必须是向量(数值, 字符, 逻辑)，因子，数值矩阵，列表或者其他数据框。

矩阵，列表和数据框为新的数据框提供了尽可能多的变量，因为它们各自拥有列，元素或者变量。

数值向量，逻辑值，因子保持原有格式，而字符 向量会被强制转换成因子并且它的水平就是 向量中出现的独立值。

在数据框中以变量形式出现的向量结构必须 长度一致，矩阵结构必须有一样的 行数。

数据框常常会被看作是一个由不同模式和属性的列构成的 矩阵。它能以 矩阵形式出现，行列可以通过 矩阵的索引习惯访问。

[Making data frames](#): 创建数据框

[attach\(\) and detach\(\)](#): attach() 和 detach()

[Working with data frames](#): 使用数据框

[Attaching arbitrary lists](#): 绑定任意列表

[Managing the search path](#): 管理搜索路径

Next: [attach\(\) and detach\(\)](#), Previous: [Data frames](#), Up: [Data frames](#)

6.3.1 创建数据框

可以通过函数 `data.frame` 创建符合 上面对列(分量)限制的数据框 对象：

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

符合数据框限制的列表可被函数 `as.data.frame()` 强制转换成数据框。从外部文件读取一个数据框最简单的方法是 使用函数 `read.table()`。这个将在 [Reading data from files](#) 部分详细讨论。

Next: [Working with data frames](#), Previous: [Making data frames](#), Up: [Data frames](#)

6.3.2 attach() 和 detach()

用 \$ 符号访问对象不是非常的方便，如 `accountants$statef`。一个非常有用的想法就是让列表或者数据框的分量可以通过它们的名字直接调用。而且这种调用是暂时性的，只是没有显式的引用列表名字。

函数 `attach()` 除了可以用目录路径作为参数，也可以使用数据框。假定数据框 `lentils` 有三个变量 `lentils$u`, `lentils$v`, `lentils$w`，那么

```
> attach(lentils)
```

将把数据框绑定在搜索路径的位置2（position 2）上¹。如果位置1没有变量 `u`, `v` 或 `w`，那么 `u`, `v` 和 `w` 可以直接在数据框中访问。此时，下面的命令

```
> u <- v+w
```

这个命令实际上没有替换数据框中的变量 `u`，而是被处于搜索路径位置1的工作空间中的变量 `u` 所屏蔽。为了真正改变数据框中的数据，最简单的办法还是使用 \$ 符号：

```
> lentils$u <- v+w
```

但是新的分量 `u` 是不可见的，直到数据框绑定去除或者重新绑定。

去除一个数据框的绑定，可以使用

```
> detach()
```

确切地说，该命令去掉了搜索路径中与位置2的绑定。此时，`u`, `v` 和 `w` 将不再可见，但可以用 `lentils$u` 类似的命令来查看这些变量。如果实体所处的环境位置值大于2，则可以通过把位置值直接传给 `detach` 的办法实现绑定去除，不过，最为安全的办法是直接使用名字，如 `detach(lentils)` 和 `detach("lentils")`

Note: 最新版本的 R 里面，列表和数据框只能在位置2或更高的层次绑定。这样就不可以直接对绑定的列表和数据框赋值(因此，在一定程度上它们是静态的)。

Footnotes

[1] 译者注：R 的搜索路径是一种层状结构，当前搜索位置是1，通过函数attach()可以进入搜索位置2。

Next: [Attaching arbitrary lists](#), Previous: [attach\(\) and detach\(\)](#), Up: [Data frames](#)

6.3.3 使用数据框

下面的转换可以使你在工作目录下同时处理不同的问题

将定义好的变量放在一起，不同问题的数据框采用意思比较明显的名字；
处理问题时，将适当的数据框绑定在位置 2，用位置 1 的工作目录操作定量的和临时的变量；
问题结束时，用 \$ 形式的赋值命令把任何你想保留的变量加入数据框中，然后利用函数 detach() 将绑定去除；
最后去掉工作目录中所有你不想要的变量，尽可能清空临时变量。

这样，就可以非常方便的在同一个目录下面处理多个含有类似名叫 x, y 和 z 变量的问题。

Next: [Managing the search path](#), Previous: [Working with data frames](#), Up: [Data frames](#)

6.3.4 绑定任意的列表

`attach()` 是一个泛型函数。它不仅允许搜索路径绑定 目录和数据框，而且还可以绑定其他对象。所以任何其他 "list" 模式的对象 都可以这样绑定：

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

任何被绑定的东西都可利用 `detach` 通过位置编号 或者名字（最好采用名字）去除绑定。

Previous: [Attaching arbitrary lists](#), Up: [Data frames](#)

6.3.5 管理搜索路径

函数 `search` 显示当前的搜索路径。它可以用来跟踪已经被绑定或者绑定去除的列表和数据框(以及包)。最开始, 它会给出

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

其中 `.GlobalEnv` 是工作空间^[1]。

`lentils` 被绑定后, 我们可以看到

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

`ls` (或者 `objects`) 可以用来查看搜索路径中任何位置的内容。

最后, 我们去除数据框的绑定, 并且确证 它是否已从搜索路径中去除。

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

Footnotes

[1] 第二个术语的意义 可以参考 `autoload` 的帮助文件。

Next: [Probability distributions](#), Previous: [Lists and data frames](#), Up: [Top](#)

7 从文件中读取数据

大的数据对象常常是从外部文件中读入，而不是在 R 对话时用键盘输入的。R 的导入工具非常简单但是对导入文件有一些比较严格的甚至顽固的限制。R 的设计者假定 你可以用其他工具（如文件编辑器或者 Perl¹）修改你的输入文件格式以使它们符合 R 的要求。一般来说，这还是比较简单的。

如果变量主要在数据框中操作，我们强烈建议 整个数据框用函数 `read.table()` 读入。还有其他更为古老的输入函数，如 `scan()`²。它可以直接接受键盘输入的。

关于 R 导入导出数据，可以参考另外一本手册 《R 数据的导入/导出》³。

[The read.table\(\) function](#): `read.table()` 函数

[The scan\(\) function](#): `scan()` 函数

[Accessing builtin datasets](#): 访问内置数据集

[Editing data](#): 编辑数据

Footnotes

[1] 在 UNIX 环境下，工具 Sed 或 Awk 也可以考虑。

[2] 译者注：虽然有点原始，但可能比 `read.table()` 用的还多。键入 `scan()` 后，可以直接把 Excel 或其他格式的数据，直接拷到控制台的

[3] 译者注：我正在翻译，现在可以看英文原版的，在 R 的官方网站上。

Next: [The scan\(\) function](#), Previous: [Reading data from files](#), Up: [Reading data from files](#)

7.1 read.table()函数

为了可以直接读取整个数据框，外部文件常常要求 有特定的格式。

第一行可以有该数据框各个变量的 名字。
随后的行中第一个条目是行标签，其他条目是各个变量的值。

在文件中，第一行比第二行少一个条目，这样做是 被强制要求的¹。因此一个被看作数据框读入的文件开头几行应该是下面的形式。

有列名字和行标签的输入文件格式：

	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(除了行标签)以数值变量的形式 读入，对应非数值变量则以因子的形式读入，如例子中的 Cent.heat 项。在必要时可以改变变量类型。

函数 read.table() 可以用来直接读入 数据框

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

你常常需要忽略掉行标签而直接使用默认的行标签。 这种情况下，输入文件就如下面所示的一样 省略行标签。

没有行标签的输入文件格式：

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
...					

这时，数据框可用如下命令读入

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

其中 header=TRUE 选项指定第一行是标题行，并且因此 省略文件中给定的行标签。

[The scan\(\) function](#): scan()函数

Footnotes

[1] 译者注：你可以让第一行和第二行条目数一致。但是 R 会把第一行当作数据数据处理而不是你所期望的变量名。这是可以设定 read.table() 的参数 header=TRUE 解决这个问题。

Next: [Accessing builtin datasets](#), Previous: [The read.table\(\) function](#), Up: [Reading data from files](#)

7.2 scan() 函数

假定有三个数据向量，长度一致并且要求并行读入。其中，第一个向量是字符模式，另外两个是数值模式，文件是 input.dat。第一步是用 scan() 以列表的形式读入这三个向量，如下所示

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

第二个参数是一个设定读入向量模式的虚拟列表结构（dummy list structure）。返回的结果 inp 是以三个读入向量作为分量的列表。为了把数据条目分成三个独立的向量，可以使用下面的赋值方式

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

更为便利的是，虚拟列表可以设定命名的变量。这种情况下，可以用名字直接访问向量。例如

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

如果你想分别访问这些变量，你要么可以提取这些数据框中的分量并且赋给新的变量：

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

要么把这个列表绑定在搜索路径的位置 2。（see [Attaching arbitrary lists](#)）

如果第二个参数是一个单一值（虚拟值）而不是列表，那么一个单一向量将会被读入。它的所有元素将会和这个虚拟值的模式一致。

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

R 里面有许多更为精细的输入工具，在其他一些手册中会有具体介绍。

Next: [Editing data](#), Previous: [The scan\(\) function](#), Up: [Reading data from files](#)

7.3 访问内置数据

R 提供了大约100个内置的数据集(在包 `datasets` 中)，其他的包(包括和R捆绑发布的推荐包) 也提供了一些作为例子的数据集。可以用下面的命令 查看当前能访问的数据集列表

```
data()
```

自 R 2.0.0版本开始，所有 R 提供的数据集可以通过名字 直接访问。但是，许多包 仍然沿用早期用 `data` 在 R 中导入数据的方式，例如

```
data(infert)
```

在标准包中，仍然可以这样使用(就像这个 例子一样)。许多情况下，这样会加载一个同名的 R 对象。不过，在某些情况下它会加载好几个包，你可以查看帮助文档弄清楚什么时候会出现意外。

7.3.1 从其他 R 包里面导入数据

为了访问某个特定包的数据，可以使用参数 `package`，例如

```
data(package="rpart")
data(Puromycin, package="datasets")
```

如果一个包已经被 `library` 绑定，那么它的数据集会被 自动地加入到它的搜索路径中。

用户捐献的包含有非常丰富的数据集。

Previous: [Accessing builtin datasets](#), Up: [Reading data from files](#)

7.4 编辑数据

用 `edit` 调用数据框和矩阵时，R 会产生一个电子表形式的编辑环境。这对在数据集上进行小的修改时非常有用的。它的命令是

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

这样做允许你编辑你的数据集 `xold`，将修改后的对象 保存到 `xnew` 里面。如果你想更改 原始的数据集 `xold`，最简单的办法是用 `fix(xold)`。它等价于 `xold <- edit(xold)`。

命令

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

允许你以电子表格的界面输入新数据。

Next: [Loops and conditional execution](#), Previous: [Reading data from files](#), Up: [Top](#)

8 概率分布

[R as a set of statistical tables](#): R 的统计表

[Examining the distribution of a set of data](#): 检验一个数据集的分布

[One- and two-sample tests](#): 单样本和双样本检验

Next: [Examining the distribution of a set of data](#), Previous: [Probability distributions](#), Up: [Probability distributions](#)

8.1 R 的统计表

R 给出了详尽的统计表。R 还提供了相关函数来 计算累计概率分布函数 $X \leq x$), 概率密度函数和分位数函数(给定 q , 符合 $P(X \leq x) > q$ 的最小 x 就是对应的分位数), 和基于概率分布的计算机模拟。

概率分布	R 对应的名字	附加参数
分布	beta	shape1, shape2, ncp
二项式分布	binom	size, prob
Cauchy 分布	cauchy	location, scale
卡方分布	chisq	df, ncp
指数分布	exp	rate
F分布	f	df1, df1, ncp
分布	gamma	shape, scale
几何分布	geom	prob
超几何分布	hyper	m, n, k
对数正态分布	lnorm	meanlog, sdlog
logistic 分布	logis	location, scale
负二项式分布	nbinom	size, prob
正态分布	norm	mean, sd
Poisson 分布	pois	lambda
t 分布	t	df, ncp
均匀分布	unif	min, max
Weibull 分布	weibull	shape, scale
Wilcoxon 分布	wilcox	m, n

不同的名字前缀表示不同的含义，d表示概率密度函数，p表示累积分布函数（cumulative distribution function，CDF），q表示分位函数以及r表示随机模拟(*random deviates*)或者随机数发生器。dxxx的第一个参数是x，pxxx是q，qxxx是p，和rxxx的是n (rhyper和rwilcox例外，二者的参数是nn)。偏态指数（non-centrality parameter）ncp现在仅用于累积分布函数，大多数概率密度函数和部分其他情况：更细节的内容可以参考帮助文档。

pxxx和qxxx函数都有逻辑参数lower.tail和log.p。dxxx也有一个逻辑函数log。它们可以用来计算所要的函数值。例如可以通过下式计算累计(“积分的”)风险(hazard)函数，

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

它们也可以直接用来计算更精确的对数似然值(dxxx(..., log = TRUE))。

此外还有函数ptukey和qtukey计算来自正态分布的样本的标准化全距(studentized range)的分布。

这里是一些例子

```
> ## t分布的双侧p值
> 2*pt(-2.43, df = 13)
> ## F(2, 7)分布的上1%分位数
> qf(0.99, 2, 7)
```

Next: [One- and two-sample tests](#), Previous: [R as a set of statistical tables](#), Up: [Probability distributions](#)

8.2 检验一个数据集的分布

我们可以用很多方法分析一个单变量的数据集。最简单的办法就是直接看数字。利用函数 `summary` 和 `fivenum` 会得到 两个稍稍有点差异的汇总信息。此外，`stem` (“ 茎叶 ” 图)也会反映整个数据集的数字信息的。

```
> 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 | 00002223333333577777777888822335777888
20 | 00002223378800035778
22 | 0002335578023578
24 | 00228
26 | 23
28 | 080
30 | 7
32 | 2337
34 | 250077
36 | 0000823577
38 | 2333335582225577
40 | 0000003357788888002233555577778
42 | 03335555778800233333555577778
44 | 0222233555778000000002333357778888
46 | 00002333577000000023578
48 | 000000222335800333
50 | 0370
```

茎叶图和柱状图相似，R 用函数 `hist` 绘制柱状图的。

```
> hist(eruptions)
## 让箱距缩小，绘制密度图
> hist(eruptions, seq(1.6, 5.2, 0.2), prob=TRUE)
> lines(density(eruptions, bw=0.1))
> rug(eruptions) # show the actual data points
```

许多不错的密度图都是用 density 绘制的。在这个例子中，我们加了一条由 density 产生的曲线。你可以用试错法（trial-and-error）选择带宽 bw（bandwidth）因为默认的带宽值让密度曲线过于平滑（这样做常常会让你得到非常有“意思”的密度分布）。（现在已经有自动的带宽挑选办法，在这个例子中 bw = "SJ" 给出了不错的结果。）

我们可以用函数 ecdf 得到一个数据集的经验累积分布（empirical cumulative distribution）

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

这个分布和其他标准分布差异很大。这种分布说明火山爆发都在3分钟以后发生¹。让我们拟合一个正态分布，并且重叠上面得到的经验累积密度分布。

```
> 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)
```

分位比较图（Quantile-quantile (Q-Q) plot）可以让我们更细致地研究二者的吻合程度。

```
par(pty="s") # 设置一个方形的图形区域
qqnorm(long); qqline(long)
```

上述命令得到的QQ图表明二者还是比较吻合的，但右侧尾部偏离期望的正态分布。我们可以用 t 分布得到一些模拟数据重复上面的过程

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

得到的QQ图常常会出现偏离正态期望的长尾部(如果是随机样本)。我们可以用下面的命令针对特定的分布绘制 Q-Q图

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

最后，我们需要更为正规的正态性检验。R提供了 Shapiro-Wilk 检验

```
> 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
```

(注意一般的统计分布理论 (distribution theory) 在这里可能无效 , 因为我们用同样的一个样本对正态分布的参数进行估计的。)

Footnotes

[1] 译者注 : 原文是 : How about the right-hand mode, say eruptions of longer than 3 minutes?. 有点不明白 “ right-hand mode ” 。

Previous: [Examining the distribution of a set of data](#), Up: [Probability distributions](#)

8.3 单样本和双样本检验

到现在为止，我们已经学会了单样本的正态性检验。而更常用的操作是比较两个样本的特征。在 R 里面，所有“传统”的检验都放在包 `stats` 里面。这个包常常会自动载入。

下面是冰融化过程的潜热（latent heat）（*cal/gm*）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
```

盒状图（boxplot）为这两组数据提供简单的图形比较。

```
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)
```

从图上可以直观的看出第一组数据 相比第二组数据倾向给出较大的值。

为了比较两个样本的均值是否相等，我们可以使用 *非配对* 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
```

上面的结果表明在正态前提下，二者有明显的统计差异。R 函数默认两个样本方差不齐，而 S-Plus 相似函数 `t.test` 则默认方差齐性。如果两个样本都是来自正态群体，我们可以用 F 检验来确定方差的齐性情况，

```
> 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
```

这表明二者方差在统计学上没有显著差异，我们可以采用传统的 假设方差齐性的 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
```

所有这些检验都假设了数据的正态性。双样本的 Wilcoxon (或者 Mann-Whitney) 检验没有正态性的前提，仅仅要求在无效假设情况下样本来自一个常规的连续分布。

```
> 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(A, B)
```

注意警告信息：在两个样本中都有同秩现象。这表明这些数据来自离散分布(可能会进行一定的近似处理)。

有好多种方法可以图形化的显示两个样本的差别。我们已经看过盒状图的比较。下面的命令

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

同时显示两个样本的经验累计概率分布，而 qqplot 得到的是两个样本的 Q-Q 图。Kolmogorov-Smirnov 检验是对两个经验累计概率分布间的最大垂直距离进行统计的。Kolmogorov-Smirnov 检验假定数据服从一个常规的连续分布：

```
> 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)
```


Next: [Writing your own functions](#), Previous: [Probability distributions](#), Up: [Top](#)

9 成组，循环和条件控制

[Grouped expressions](#): 成组表达式

[Control statements](#): 控制语句

Next: [Control statements](#), Previous: [Loops and conditional execution](#), Up: [Loops and conditional execution](#)

9.1 成组表达式

R 是一种表达式语言 (expression language) 因为它仅有的命令形式 是返回结果的函数和表达式。赋值操作实际上 也是一个表达式结果的再分配, 并且可以用在 任何表达式中, 甚至多重赋值 也是允许的。

命令可以用大括弧圈在一起 $\{expr_1; \dots; expr_m\}$ 。此时, 这一组命令的结果 是该组中最后一个命令的值。既然一个组依然是一个表达式, 它就可能放在括弧中, 放在一个更大的表达式中, 等等。

Next: [Repetitive execution](#), Previous: [Control statements](#), Up: [Control statements](#)

9.2.1 条件控制：if语句

R 语言的条件语句形式为

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

其中 *expr_1* 是控制条件，表达式的其他部分则是不言而喻的。

“短路”（short-circuit）操作符 `&&` 和 `||` 常常用于 if 语句的条件控制部分。这里要注意 `&` 和 `|` 将用于向量的所有元素，而 `&&` 和 `||` 仅用于长度为1的向量和必要时评价第二个参数^[1]

R 提供了 if/else 条件语句向量形式的函数 `ifelse`。它的使用方式是 `ifelse(condition, a, b)`，最终返回一个和最长的参数向量同长的向量。`condition[i]` 为真时，该向量对应的元素是 `a[i]`，否则为 `b[i]`。

Footnotes

[1] 译者注：原文为：evaluate their second argument if necessary。暂时还没有明白这句话。反正对于两个长度大于1的向量，短路操作符仅作用于第一个元素。

Previous: [Conditional execution](#), Up: [Control statements](#)

9.2.2 循环控制：for循环，repeat 和 while

R 语言有下面形式的 for 循环架构

```
> 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(常常以1:20这种形式出现)，而 *expr_2* 常常是根据虚拟变量 *name* 而设计的成组表达式。在 *name* 访问 *expr_1* 所有 可以取到的值时，*expr_2* 都会运行。

假定 ind 是分类指标向量（vector of class indicators），我们将依这种分类指标绘制 y 对 x 的散点图。一种方法是用函数 `coplot()`¹ 产生对应因子各个水平的散点图。另外一种方法就是把所有图在屏幕上同时 显示，就如下面命令所示：

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

(函数 `split()` 通过某个因子把一个 大的向量分成一系列小的向量。这是一个非常有用的函数，常常和 盒状图一起使用。具体细节可以用 `help` 命令得到。)

警告：相比其他程序语言，R 代码里面很少使用 `for()` 循环。在R里面用‘完整对象’（whole object）形式 可能既清晰又高效²。

另外一种循环语句包括

```
> repeat expr
```

和语句

```
> while (condition) expr
```

关键字 `break`可以用于结束任何循环，甚至是非常规的。它是结束 `repeat` 循环的唯一办法。关键

字 `next` 可以用来结束一次特定的循环，然后直接跳入 “ 下一次 ” 循环。控制语句常常用于 函数 中。这在下一章 [Writing your own functions](#) 会详细讨论，同时也会引用更多的例子。

Footnotes

[1] 将在后面的 的内容中讨论，或者用包 `lattice` 里面的函数 `xyplot`。

[2] 译者注：显式的循环常常是能避免的，可以利用函数对象对每个元素计算值。如用 `sum` 等统计函数及 `apply`、`lapply`、`sapply`、`tapply` 等函数都可以部分代替显式循环。

Previous: [Grouped expressions](#), Up: [Loops and conditional execution](#)

9.2 控制语句

[Conditional execution](#): 条件控制

[Repetitive execution](#): 循环控制

Next: [Statistical models in R](#), Previous: [Loops and conditional execution](#), Up: [Top](#)

10 编写函数

正如前面内容所暗示的一样，R 语言允许用户 创建自己的函数 (function) 对象。R 有一些 内部函数并且可以用在 其他的表达式中。通过这个过程，R 在程序的功能性，便利性和优美性上得到了扩展。学写这些有用的函数 是一个人轻松地创造性地使用 R 的 最主要的方式。

需要强调的是，大多是函数都作为 R 系统的一部分提供，如 `mean()`, `var()`, `postscript()` 等等。这些函数都是用 R 写的，因此在本质上和用户写的没有差别。

一个函数是通过下面的语句形式定义的。

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

其中 *expression* 是一个 R 表达式(常常是一个成组 表达式)，它利用参数 *arg_i* 计算最终的结果。该表达式的值就是返回给函数的最终值。

可以在任何地方以 *name(expr_1, expr_2, ...)* 的形式调用函数。

[Simple examples](#): 简单的例子

[Defining new binary operators](#): 定义新的二元操作符

[Named arguments and defaults](#): 参数命名和默认值

[The ellipsis argument \(...\)](#): 省略符号的参数(...)

[Assignment within functions](#): 函数内部的赋值

[More advanced examples](#): 高级的例子

[Scope](#): 作用域

[Customizing the environment](#): 定制环境

[Object orientation](#): 面向对象

Next: [Defining new binary operators](#), Previous: [Writing your own functions](#), Up: [Writing your own functions](#)

10.1 一个简单的例子

这是一个简单的例子，它用来计算双样本的 t-统计量，并且显示“所有步骤”。这是一个人为的例子，当然还有其他更简单的办法得到一样的结果。

函数定义如下：

```
> 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(s*(1/n1 + 1/n2))
  tst
}
```

通过这个函数，你可以下面的命令实现双样本 t-检验。

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

第二个例子是仿效 Matlab 里面的反斜杠命令。它是用来返回向量 y 正交投影到 X 列空间上面的系数。(这常常被称为回归系数的最小二乘法估计。)这可以用函数 `qr()` 来实现；但是直接使用这个函数有时有点难处理。下面提供了一个简单而又安全的函数。

给定 $n \times 1$ 的向量 y 和一个 $n \times p$ 的矩阵 X ，因此 $X \backslash y$ 可以定义如下 $(X'X)^{-1}X'y$ ，其中 $(X'X)^{-1}$ 这就是 $X'X$ 的广义逆矩阵 (generalized inverse)。

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

当这个对象创建后，它可能用于这样的命令中：

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


经典的 R 函数 `lsfit()` 可以很好的实现这个功能和其他一些相关的事情¹。它以一种有点违反直觉的方式依次用函数 `qr()` 和 `qr.coef()` 去完成这部分计算。如果一部分代码常常使用，我们可以把这部分代码单独列出来成为函数使用。如果是这样，我们可能期望矩阵的二元操作能以一种更为便利的方式进行。

Footnotes

[1] 参见 [Statistical models in R](#) 描述的的方法

Next: [Named arguments and defaults](#), Previous: [Simple examples](#), Up: [Writing your own functions](#)

10.2 定义新的元操作符

假定我们给予函数 `bslash()` 一个不同的名字，且以 下面的形式给出

%anything%

那么它将以二元操作符的形式在表达式中使用，而不是函数的形式。例如我们选择 `!` 作为中间的字符。函数可以如下定义

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

(注意要使用引号)。该函数然后就可以用于 `X %!% y`。(反斜杠符不是一个很好的选择，因为在某些情况下会引入一些特定的问题。)

矩阵的乘法操作符 `%*%` 和外积操作符 `%o%` 同样是这种方式定义的 二元操作符。

Next: [The ellipsis argument \(...\)](#), Previous: [Defining new binary operators](#), Up: [Writing your own functions](#)

10.3 参数命名和默认值

和 [Generating regular sequences](#) 提示的一样，如果调用的函数的参数以 “*name=object*” 的方式给出，它们可以用任何顺序。但是，参数赋值序列可能以未命名的，位置特异性的方式给出，同时也有可能在这些位置特异性的参数后加上命名参数赋值。

因此，如果有下面方式定义的函数 fun1

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

那么函数将会被好几种方式调用，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)
```

上面所有的方式是等价的。

许多时候，参数会被设定一些默认值。如果默认值适合你要做的事情，你可以省略这些参数。例如，函数 fun1 用下面的方式定义

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

它可以被如下命令调用

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

这和前面的三种情况等价。

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

这就改变了一个默认值。

特别说明一下，默认值可以是任何表达式，甚至是函数本身所带有的其他参数；它们没有要求

是常数。我们的例子采用常数只是使问题简单容易说明。

Next: [Assignment within functions](#), Previous: [Named arguments and defaults](#), Up: [Writing your own functions](#)

10.4 ... 参数

还有一种常常出现的情况就是要求一个函数的参数设置可以传递给另外一个函数。例如图形函数如果调用了函数 `par()` 和其他如 `plot()` 类的函数，`par()` 函数的图形设置将会传递给图形输出的设备控制。(See [The par\(\) function](#), 后面的章节会给出函数 `par()` 更为详细的内容。)这个可以通过给函数增加一个额外的参数来实现。这个参数字面上就是 `...`，它可以被传递。一个概述性的例子可以如下所示。

```
fun1 <- function(data, data.frame, graph=TRUE, limit=20, ...) {  
  [省略一些语句]  
  if (graph)  
    par(pch="*", ...)  
  [省略其他语句]  
}
```

Next: [More advanced examples](#), Previous: [The ellipsis argument \(...\)](#), Up: [Writing your own functions](#)

10.5 在函数中赋值

注意任何在函数内部的普通赋值都是局部的 暂时的，当退出函数时都会丢失。因此 函数中的赋值语句 `X <- qr(X)` 不会影响 调用该函数的程序赋值情况。

如果要彻底理解 R 赋值管理的原则，读者需要熟悉解析 框架（Evaluation frame）的概念。这属于高级内容，不在这里深入讨论。

如果想在函数里面全局赋值或者永久赋值，那么可以采用“强赋值”（superassignment）操作符 `<<-` 或者采用函数 `assign()`。用 `help` 命令可以得到更细节的内容。S-Plus 用户需要注意 `<<-` 在 R 里面有着不同的语义（semantics）。这些会在 [Scope](#) 部分会详细讨论。

Next: [Scope](#), Previous: [Assignment within functions](#), Up: [Writing your own functions](#)

10.6 更多高级的例子

[Efficiency factors in block designs](#): 区组设计中的效率因子

[Dropping all names in a printed array](#): 去除打印数组中的名字

[Recursive numerical integration](#): 递归式的数值积分

Next: [Dropping all names in a printed array](#), Previous: [More advanced examples](#), Up: [More advanced examples](#)

10.6.1 区组设计中的效率因子

下面是一个有点枯燥但较为完整的例子。它是用来计算一个区组设计中的效率因子(这个问题的方面在 [Index arrays](#) 已经讨论过了)。

区组设计 (block design) 需要考虑两个因子 blocks (b 个水平) 和 varieties (v 个水平)。如果 R 和 K 分别是 $v \times v$ 和 $b \times b$ 重复 (replications) 及 区组大小 (block size) 矩阵而 N 则是 $b \times v$ 的关联矩阵, 那么效率因子就是这个矩阵的特征值。 $E = I_v - R^{-1/2} N' K^{-1} N R^{-1/2} = I_v - A' A$, where $A = K^{-1/2} N R^{-1/2}$. 写这个函数的一种方式就是

```
> 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))    # 去掉 dim 属性
  R <- as.vector(table(varieties)) # 去掉 dim 属性
  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)
}
```

这种情况下, 奇异值分解 比求解特征值效率高。

函数的结果是一个列表。它不仅以第一个分量的形式给出了效率因子, 还给出了区组和规范对照信息, 因为有些时候这些会给出额外有用的定量信息。

Next: [Recursive numerical integration](#), Previous: [Efficiency factors in block designs](#), Up: [More advanced examples](#)

10.6.2 去除打印数组中的名字

为了显示一个大的数组或者矩阵，常常需要 需要以一个完整的块的形式显示，同时去掉数组名和编号。去掉 `dimnames` 属性是不能达到这个要求的，因为 R 环境会把空的字符串赋给 `dimnames` 属性。为了打印一个矩阵 X

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

这个可以非常便利地通过下面的函数 `no.dimnames()` 实现。它是利用一种“卷绕”（wrap around）的方式实现的。这个例子还说明一些非常高效有用的用户函数 也可以是非常简洁的。

```
no.dimnames <- function(a) {
  ## 为了更紧凑的打印，可以去除数组中的维度名字
  d <- list()
  l <- 0
  for(i in dim(a)) {
    d[[l <- l + 1]] <- rep("", i)
  }
  dimnames(a) <- d
  a
}
```

通过这个函数，数组可以用一种紧凑的方式 显示

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

这对大的整数数非常的有用，因为这些数组 表现出来的式样（pattern）可能比它们的值更为重要。

Previous: [Dropping all names in a printed array](#), Up: [More advanced examples](#)

10.6.3 递归式的数值积分

函数可以是递归的，可以在函数内部调用自己。但是需要注意的是，这些函数或者变量，不会被更高层次的解析框架（evaluation frame）中的被调用函数所继承。如果它们在搜索路径中，这种情况就会出现。

下面的例子显示了一个最简单的一维数值积分方法。被积函数在所积范围的两端和中点的值会被计算。如果单面板梯形法则（one-panel trapezium rule）的结果和双面板的非常相似，那么就以后者作为返回值。否则同样的过程会递归用于各个面板。这是一个自适应的积分过程。它会集中各个被积函数接近线性的区域函数计算值。但是这种方法开销有点过大¹。相对其他积分算法，它的优势体现在被积函数既平滑又很难求值时。

这个例子同样可以部分的作为一个 R 编程的难题给出。

```
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)
}
```

[Scope](#): 作用域

[Object orientation](#): 面向对象

Footnotes

[1] 译者注：原句为：“There is, however, a heavy overhead”

Next: [Customizing the environment](#), Previous: [More advanced examples](#), Up: [Writing your own functions](#)

10.7 作用域

这一部分的内容相对本文档其他部分的内容更偏向一些技术性的问题。但是它会澄清 S-Plus 和 R 一些重要的差异。

在函数内部的变量可以分为三类：形式参数，局部变量和自由变量。形式参数是出现在函数的参数列表中的变量。它们的值由实际的函数参数 *绑定* 形式参数的过程决定的。局部变量由函数内部的表达式的值决定的。既不是形式参数又不是局部变量的变量是自由变量。自由变量如果被赋值将会变成局部变量。考虑下面的函数定义过程。

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

在这个函数中，*x* 是形式参数，*y* 是局部变量，*z* 是自由变量。

在 R 里面，可以利用函数被创建的环境中某个变量的第一次出现解析一个自由变量的绑定。这称为 *词法作用域* (lexical scope)。我们可以定义一个函数 *cube*。

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

函数 *sq* 中的变量 *n* 不是函数的参数。因此它是自由变量。一些作用域的原则可以用来确定和它相关的值。在静态作用域 (S-Plus)，这个值指的是一个和全局变量 *n* 相关的值。在词法作用域 (R)，它指的是函数 *cube* 的参数。因为当 *sq* 定义的时候，它会动态绑定参数 *n*。在 R 里面解析和在 S-Plus 里面解析不同点在于 S-Plus 搜索全局变量 *n* 而 R 在 *cube* 调用时首先寻找环境创建的变量 *n*。

```
## 首先用 S 解析  
S> cube(2)  
Error in sq(): Object "n" not found
```

```
Dumped
```

```
S> n <- 3
```

```
S> cube(2)
```

```
[1] 18
```

```
## 同样的函数在 R 中解析
```

```
R> cube(2)
```

```
[1] 8
```

词汇作用域会给予函数可变状态（mutable state）。下面的例子演示 R 如何模仿一个银行的帐户。真正的银行帐户必须同时有跟踪收支平衡或者总额的变量，提供提款业务，取款业务和显示当前余额的函数。我们可以在 `account` 里面创建三个函数，然后返回一个包含它们的列表。当调用 `account` 时，它读入一个数值参数 `total`，并且返回一个包含三个函数的列表。因为这些函数时定义在一个有变量 `total` 的环境中，它们可以访问它的值。

`<<-` 是一个特别的赋值操作符，它用来更改和 `total` 相关的值。这个操作符会回溯到一个含有标识符 `total` 的密闭环境中，当它找到这个环境，它会用操作符右边的值替换环境中这个变量的值。如果在全局变量或者最高层次的环境中仍然没有找到标识符 `total`，那么该变量就会被创建并且在那里被赋值。大多数用户用 `<<-` 创建全局变量，并且把操作符右边的值赋给它¹。仅仅当 `<<-` 用于一个函数的输出是另外一个函数的输入时，这里描述的独特行为才会出现。

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

Footnotes

[1] 从某种意义上来说，这样做有点模仿 S-Plus 的用法，因为在 S-Plus 中，这种操作符就是 创建全局变量并且进行赋值操作。

Next: [Object orientation](#), Previous: [Scope](#), Up: [Writing your own functions](#)

10.8 定制环境

用户可以有几种办法定制环境。可以修改 位置初始化文件，并且每个目录都有它特有的一个初始化文件。还有就是利用函数 `.First` 和 `.Last`。

位置初始化文件的路径可以通过 环境变量 `R_PROFILE` 设置。如果该变量没有设置，默认是R安装目录下面的子目录 `etc` 中的 `Rprofile.site`。这个文件包括你每次执行 R 时一些自动运行的命令。第二个定制文件是 `.Rprofile`¹，它可以放在任何目录下面。如果 R 在该目录下面 被调用，这个文件就会被载入。这个文件允许用户 定制它们的工作空间，允许在不同的工作目录下 设置不同的起始命令。如果在起始目录中没有 `.Rprofile`，R 会在用户主目录下面搜索 `.Rprofile` 文件并且调用它（如果它 存在的话）。

在这两个文件或者 `.RData` 中任何叫 `.First()` 的函数 都有特定的状态的。它会在 R 对话的开始时自动执行并且初始化环境。下面例子中的定义允许 将提示符改为 `$`，以及设置其他有用的东西。这些设置同样会在其他会话中起作用。

因此，这些文件的执行顺序是 `Rprofile.site`，`.Rprofile`，`.RData` 然后是 `.First()`。后面文件中 定义会屏蔽掉前面文件中的定义。

```
> .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"))
                                     # my personal functions
  library(MASS)                      # attach a package
}
```

相似的是，如果定义了函数 `.Last()`，它(常常)会在对话 结束时执行。一个例子就是

```
> .Last <- function() {
  graphics.off()                    # 一个小的安全措施。
  cat(paste(date(),"\nAdios\n"))    # 该吃午饭了？
}
```

Footnotes

[1] 在 UNIX 系统中，这种文件不可见。

Previous: [Customizing the environment](#), Up: [Writing your own functions](#)

10.9 类，泛型函数和面向对象

一个对象的类决定了它会如何被一个泛型函数处理。相反，一个泛型函数通过参数类的特异参数来完成特定工作或者事务的。如果参数缺乏任何类属性，或者在该问题中有一个不能被任何泛型函数处理的类，泛型函数会有一种默认的处理方式。

下面的一个例子使这个问题清晰。类机制为用户提供了为特定问题设计和编写泛型函数的便利。在其他泛型函数中，`plot()` 用于图形化显示对象，`summary()` 用于各种类型的概述分析，以及 `anova()` 用于比较统计模型。

能以特定方式处理类的泛型函数的数目非常的庞大。例如，可以在非常时髦的类对象 "data.frame" 中使用的函数有

```
[ []<- any as.matrix  
[<- mean plot summary
```

可以用函数 `methods()` 得到当前对某个类对象 可用的泛型函数列表：

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

相反，一个泛型函数可以处理的类同样很多。例如，`plot()` 有默认的方法和变量处理对象类 "data.frame"，"density"，"factor"，等等。一个完整的列表同样可以通过函数 `methods()` 得到：

```
> methods(plot)
```

读者可以参考完整描述这一机制的正式文档。

Next: [Graphics](#), Previous: [Writing your own functions](#), Up: [Top](#)

11 R中的统计模型

这一部分假定读者已经对统计方法，特别是回归分析和方差分析有一定的了解。后面我们将会假定读者对广义线性模型和非线性模型也比较了解。

R 已经很好地定义了统计模型拟合中的一些前提条件，因此我们能构建出一些通用的方法以用于各种问题。

R 提供了一系列紧密联系的统计模型拟合的工具，使得拟合工作变得简单。正如我们在绪论中提到的一样，基本的屏幕输出是最为简洁的，因此用户需要调用一些辅助函数来提取细节的结果信息。

[Formulae for statistical models](#): 统计模型中的公式

[Linear models](#): 线性模型

[Generic functions for extracting model information](#): 提取模型信息的泛型函数

[Analysis of variance and model comparison](#): 方差分析和模型比较

[Updating fitted models](#): 拟合模型的更新

[Generalized linear models](#): 广义线性模型

[Nonlinear least squares and maximum likelihood models](#): 非线性最小二乘法和最大似然模型

[Some non-standard models](#): 一些非标准模型

Next: [Linear models](#), Previous: [Statistical models in R](#), Up: [Statistical models in R](#)

11.1 定义统计模型的公式

下面的统计模型的模板是一个基于 独立的方差齐性数据的线性模型

$$y_i = \sum_{j=0}^p \beta_j x_{ij} + e_i, \quad i = 1, \dots, n,$$

其中 e_i 属于 $NID(0, \sigma^2)$ 。用矩阵格式表示，它可以写为

$$y = X \beta + e$$

其中 y 是响应向量， X 是 **模型矩阵** (model matrix) 或者 **设计矩阵** (design matrix)。 X 的列 x_0, x_1, \dots, x_p 是 **决定变量** (determining variable)。通常， x_0 列就是 **截距** (intercept) 项。

例子

在给予正式的定义前，下面的例子可能会给出 一些不错的描述。

假定 $y, x, x_0, x_1, x_2, \dots$ 是数值 变量， X 是一个矩阵，而 A, B, C, \dots 是因子。下面的例子中，左边给出公式，右边给出该公式的统计模型的描述。

$$y \sim x$$

$$y \sim 1 + x$$

二者都反映了 y 对 x 的简单线性模型。第一个包含了一个隐式的截距项，而第二个则是一个显式的截距项。

$$y \sim 0 + x$$

$$y \sim -1 + x$$

$$y \sim x - 1$$

y 对 x 的通过原点的简单线性模型 (也就是没有截距项)。

$$\log(y) \sim x_1 + x_2$$

y 的变换形式 $\log(y)$ 对 x_1 和 x_2 进行的多重回归 (有一个隐式的截距项)。

$$y \sim \text{poly}(x, 2)$$

$$y \sim 1 + x + I(x^2)$$

y 对 x 的二阶多项式回归。第一种是 正交多项式 (orthogonal polynomial)，第二种是显式地注明各项的幂次。

$$y \sim X + \text{poly}(x, 2)$$

y 利用模型矩阵 X 和二阶多项式项 x 进行多重回归。

$y \sim A$

y 的单一分类因子的方差分析模型。分类因子 由 A 决定。

$y \sim A + x$

y 的单一分类因子的方差分析模型。因子项为 A ，共变项为 x 。

$y \sim A*B$

$y \sim A + B + A:B$

$y \sim B \%in\% A$

$y \sim A/B$

基于 A 和 B 的非可加两因子方差分析模型 y 。前两个公式表示相同的交叉分类设计，后两个公式表示相同的嵌套分类设计。概略地说，四个公式指明同一个模型子空间。

$y \sim (A + B + C)^2$

$y \sim A*B*C - A:B:C$

三因子实验。该模型包括一个主要影响因子，另外两个因子 有相互作用。这两个公式等价。

$y \sim A * x$

$y \sim A/x$

$y \sim A/(1 + x) - 1$

A 的每一水平拟合 y 对 x 的简单线性回归。三个公式的编码不一样。最后一个公式会对 A 每一水平分布估计截距项 和斜率项的。

$y \sim A*B + \text{Error}(C)$

一个实验设计有两个处理因素 A 和 B 以及 因子 C 决定的误差分层 (error strata)。例如在一个 区组由因子 C 决定的裂区实验设计 (split plot experiment)。

操作符 \sim 用于定义 R 的模型公式。一个普通线性模型的公式可以表示为

$response \sim op_1 term_1 op_2 term_2 op_3 term_3 \dots$

其中

$response$

是一个作为响应变量的向量或者矩阵， 或者是一个值为向量/矩阵的表达式。

op_i

是一个操作符。它要么是 $+$ 要么是 $-$ ，分别表示在一个模型中加入 或者去除某一项(默认值为前者)。

$term_i$

可以

- 是一个向量，矩阵表达式或者 1，
- 因子
- 是一个由因子，向量或矩阵通过公式操作符 (formula operators) 连接产生的公式表达式 (formula expression)。

基本上，公式中的项决定了模型矩阵中列要么被加入要么被去除。1 表示截距项，并且 默认加入模型矩阵除非 显式地去除这一选项。

公式操作符在效果上和用于程序 Glim 和 Genstat 中的 Wilkinson & Rogers 标记符相似。一个不可避免的改变是 操作符 `.` 变成了 `:` 因为点号在 R 里面是合法的名字字符。

这些符号总结如下(参考 Chambers & Hastie, 1992, p.29):

$Y \sim M$

Y 由模型 M 解释。

$M_1 + M_2$

同时包括 M_1 和 M_2 项。

$M_1 - M_2$

包括 M_1 但排除 M_2 项。

$M_1 : M_2$

M_1 和 M_2 的张量积 (tensor product)。如果两项都是因子，那么将产生 “子类” 因子^{[1](#)}。

$M_1 \%in\% M_2$

和 $M_1:M_2$ 类似，但编码方式不一样。

$M_1 * M_2$

$M_1 + M_2 + M_1:M_2$.

M_1 / M_2

$M_1 + M_2 \%in\% M_1$.

M^n

M 的所有各项以及所有到 n 阶为止的 “交互作用” 项

$I(M)$

隔离 M 。 M 内所有操作符当一般的运算符处理。该项出现在模型矩阵中。

注意，在常常用来封装函数参数的括弧中的操作符按普通的四则运算法则解释。 $I()$ 是一个恒等函数 (identity function)，它使得常规的算术运算符可以用在模型公式中。

还要注意模型公式仅仅指定了模型矩阵的列项，参数则没有显式地指定。在某些情况下可能不是这样，如非线性模型。

[Contrasts](#): 对照

Footnotes

[1] 译者注：因子交互作用项

Previous: [Formulae for statistical models](#), Up: [Formulae for statistical models](#)

11.1.1 对照

我们至少要知道模型公式是如何指定模型矩阵的列项的。对于连续变量这是比较简单的，因为每一个变量对应于模型矩阵的一个列(如果模型中包含截距，会在矩阵中列出这一列的)。

对于一个 k -水平的因子 A 该如何处理呢？无序和有序因子的结论是不一样的。对于无序因子 $k - 1$ 列用于作为因子第二，...，第 k 水平的指标(因此隐含的参数就是其他水平和第一个水平的响应程度的比较)。对于有序因子， $k - 1$ 列是在 $1, \dots, k$ 上的正交项，并且忽略常数项。

尽管这里的回答有点复杂，但这不是事情的全部。首先在含有一个因子项的模型中忽略截距项，这一项将会被编入指示所有因子水平的 k 列中。其次整个行为可以通过 options 设置参数 contrasts 而改变。R 的默认设置为

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

提这些内容的主要原因是 R 和 S 对无序因子采用不同的默认值。S 采用 Helmert 对照。因此，当你需要比较某本书上或者论文上用 S-Plus 写的代码，你必须设置

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

这是一个经过认真考虑的改变。因为处理对照 (treatment contrast) (R 默认) 对于新手是比较容易理解的。

这还没有结束，因为在各个模型的各个项中对照方式可以用函数 contrasts 和 C 重新设置。我们还没有考虑交互作用项：这将会产生各分量项的复合项。

尽管细节是复杂的，R 里面的模型公式可以产生统计专家所期望的各种模型^[1]。例如，利用关联项而非主要效应的模型拟合常常会产生令人惊讶的结果，不过这些仅仅为统计专家们设计的。

Footnotes

[1] 译者注：这里还有一个条件：provided that marginality is preserved。不知有何关联？

Next: [Generic functions for extracting model information](#), Previous: [Formulae for statistical models](#),
Up: [Statistical models in R](#)

11.2 线性模型

对于常规的多重模型 (multiple model) 拟合 , 最基本的函数是 `lm()`。 下面是调用它的一种方式的一种改进版 :

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

例如

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

将会拟合 y 对 x_1 和 x_2 的多重回归模型(和一个隐式的截距项)。

一个重要的(技术上可选)参数是 `data = production`。 它指定任何构建这个模型的参数首先必须来自数据框 `production`。 这里不需要考虑数据框 `production` 是否被绑定在搜索路径中。

Next: [Analysis of variance and model comparison](#), Previous: [Linear models](#), Up: [Statistical models in R](#)

11.3 提取模型信息的泛型函数

`lm()` 的返回值是一个模型拟合结果对象；技术上就是 "lm" 的一个结果列表类。关于拟合模型的信息可以用能调用对象类 "lm" 的泛型函数显示，提取，图示等等。这包括

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

其中一些常用的泛型函数可以简洁描述如下。

`anova(object_1, object_2)`
比较一个子模型和原模型，并且产生方差分析表。

`coef(object)`
提取回归系数 (矩阵)。

全称：`coefficients(object)`。

`deviance(object)`
残差平方和，如有权重可加权。

`formula(object)`
提取模型信息。

`plot(object)`
产生4个图，显式残差，拟合值和一些诊断图¹。

`predict(object, newdata=data.frame)`
提供的数据框必须有同原始变量一样标签的变量。结果是对应于 `data.frame` 中决定变量的预测值的向量或矩阵。

`print(object)`

简要打印一个对象的内容。常常隐式使用^[2]。

`residuals(object)`

提取残差(矩阵)，有权重时可加权，

省略方式：`resid(object)`。

`step(object)`

通过增加或者减少模型中的项并且保留层次来选择合适的模型。在逐步搜索过程中，AIC (Akaike 信息规范)值 最大的模型将会被返回。

`summary(object)`

显示较详细的模型拟合结果。

Footnotes

[1] 译者注：主要用于评估模型拟合情况的残差图。

[2] 译者注：一般不用`print()`，直接键入对象名也可以用来显示。

Previous: [Analysis of variance and model comparison](#), Up: [Analysis of variance and model comparison](#)

11.4.1 方差分析表

方差表的分析实际上是对拟合模型序列而进行的。在模型序列的特定地方增加特定的项会使残差平方和降低。因此仅仅在正交实验中，模型中增加项的顺序是没有影响的。

在多层实验设计中，程序首先把响应值依次投射到各个误差层次上，并且用均值模型去拟合各个投射。细节内容可以参考 Chambers & Hastie (1992)。

除了常规的方差分析表（ANOVA table）使用外，你还可以直接用函数 `anova()` 来比较两个模型。

```
> anova(fitted.model.1, fitted.model.2, ...)
```

结果将是一个方差分析表以显示依次加入的拟合模型的差异。需要比较的拟合模型常常是等级序列（hierarchical sequence）。这个和默认的实际上没有差别，只是使它更容易理解和控制。

Next: [Updating fitted models](#), Previous: [Generic functions for extracting model information](#), Up: [Statistical models in R](#)

11.4 方差分析和模型比较

模型拟合函数 `aov(formula, data=data.frame)` 和函数 `lm()` 非常的相似，在[Generic functions for extracting model information](#) 部分列出的泛型函数同样适用。

需要注意的是 `aov()` 还允许分析 多方差层次的模型，如裂区实验设计，利用区组内信息进行的平衡不完全区组设计（balanced incomplete block design）等。模型公式

$$response \sim mean.formula + Error(strata.formula)$$

strata.formula 定义了一个多层次实验设计中 多个方差因素。最简单的情况是，*strata.formula* 是单因素的。它定义了一个双层次的实验，也就是研究在这些因子的水平内 或者水平间的实验响应。

例如，考虑到所有的决定变量因子，模型公式可以 设计如下：

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

这常常用来描述一个同时含有均值模型 $v + n*p*k$ 和三个误差层次（“农田之间”，“农田内但区组之间”和“区组内”）的实验。

[ANOVA tables](#): 方差分析表

Next: [Generalized linear models](#), Previous: [Analysis of variance and model comparison](#), Up: [Statistical models in R](#)

11.5 更新拟合模型

函数 `update()` 是一个非常便利的函数。它允许 拟合一个比原先模型增加或减少一个项的模型。它的形式是

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

在 *new.formula* 中，公式中包含的句点，`.`，仅仅表示“旧的公式模型中的对应部分”。例如

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

这将拟合从数据框 `production` 中得到的五个变量的 多重回归模型，拟合额外增加一个变量的 六个回归量的模型，和进一步对响应值进行 平方根转换后的模型拟合。

注意参数 `data=` 在最开始调用模型拟合函数的时候指定。这个信息将会通过拟合模型对象传递给函数 `update()` 及其相关者。

符号 `.` 同样可以用在其他情况下，不过含义 有点不同。如

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

它将会拟合响应量 `y` 对 数据框 `production` 中所有变量回归的模型。

其他研究逐步回归的函数是 `add1()`, `drop1()` 和 `step()`。从字面上就可以看出这些函数的意义，更细节的内容可以参考 帮助文档。

Next: [Nonlinear least squares and maximum likelihood models](#), Previous: [Updating fitted models](#),
Up: [Statistical models in R](#)

11.6 广义线性模型

广义线性建模是线性模型在研究响应值的非正态分布以及非线性模型的简洁直接的线性转化时的一种发展。广义线性模型是基于下面一系列假设前提的：

有一个响应变量 y 和一系列有趣的刺激变量 (stimulus variable) x_1, x_2, \dots 。这些刺激变量决定响应变量的最终分布。

刺激变量仅仅通过一个线性函数影响响应值 y 的分布。该线性函数称为 **线性预测器** (linear predictor)，常常写成

$$\eta = \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p,$$

因此 x_i 当且仅当 β_i 等于0时对 y 的分布没有影响。
 y 分布的形式为

$$f_Y(y; \mu, \phi) = \exp\left(\frac{A(\phi)}{\phi} \left(y \lambda(\mu) - g(\lambda(\mu)) \right) + \tau(y, \phi)\right)$$

其中 ϕ 是度量参数 (scale parameter) (可能已知)，对所有观测恒定； A 是一个先验的权重，假定知道但是可能随观测不同有所不同； μ 是 y 的均值。也就是说假定 y 的分布是由均值和一个可能的度量参数决定的。

均值 μ 是线性预测器的平滑可逆函数 (smooth invertible function)：

$$\mu = m(\eta), \quad \eta = m^{-1}(\mu) = \text{ell}(\mu)$$

该可逆函数 $\text{ell}()$ 称为 **关联函数** (link function)。

这些假定比较宽松，足以包括统计实践中大多数有用的统计模型，但也足够严谨，使得可以发展计算和推论中一致的方法 (至少可以近似一致)。读者如果想了解这方面最新的进展，可以参考 McCullagh & Nelder (1989) 或者 Dobson (1990)。

[Families](#): 族

[The glm\(\) function:](#) glm()函数

Next: [The glm\(\) function](#), Previous: [Generalized linear models](#), Up: [Generalized linear models](#)

11.6.1 族

R 提供了一系列广义线性建模工具，从类型上来说包括 *gaussian*, 二项式, *poisson*, 反 *gaussian* 和 *gamma* 模型的响应变量分布以及 在响应变量分布没有明确给定时的逆似然 (quasi-likelihood) 模型。在后者，方差函数 (variance function) 可以认为是均值的函数，但是在另外一些情况下，该函数可以由响应变量的分布得到。

每一种响应分布允许各种关联函数将均值和线性预测器关联起来。这些自动关联函数如下表所示：

Family name	Link functions
binomial	logit, probit, log, cloglog
gaussian	identity, log, inverse
Gamma	identity, inverse, log
inverse. gaussian	1/ μ^2 , identity, inverse, log
poisson	identity, log, sqrt
quasi	logit, probit, cloglog, identity, inverse, log, 1/ μ^2 , sqrt

这些用于模型构建过程中的响应分布，关联函数和各种其他必要的信息统称为广义线性模型的族 (family)。

Previous: [Families](#), Up: [Generalized linear models](#)

11.6.2 glm()函数

既然响应的分布仅仅通过单一的一个线性函数依赖于刺激变量，那么用于线性模型的机制同样可以用于指定一个广义模型的线性部分。但是族必须以一种不同的方式指定。

R 用于广义线性回归的函数是 `glm()`，它的使用形式为

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

和 `lm()` 相比，唯一的一个新特性就是描述族的参数 *family.generator*。它是产生函数和表达式列表的函数名字。这些函数用于定义和控制模型的构建与计算过程。尽管开始看起来有点复杂，但它们非常容易使用。

这些名字是标准的。程序给定的族生成器可以参见 [Families](#) 列表中的“族名”。当选择一个关联函数时，该关联函数名和族名可以同时也在括弧里面作为参数设定。在拟（quasi）家族里面，方差函数也是以这种方式设定。

一些例子可能会使这个过程更清楚。

gaussian族

命令

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

和下面的命令结果一致。

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

但是效率上，前者差一点。注意，gaussian族没有相关参数，因此它不提供关联函数的。如一个问题需要用非标准关联函数的 gaussian 族，那么只能采用我们后面讨论的拟族。

二项式族

考虑 Silvey (1970) 提供的一个小的例子。

在 Kalythos 的 Aegean 岛上，男性居民常常患有一种先天的眼科疾病，并且随着年龄的增长而变的愈显著。现在搜集了各种年龄段岛上男性居民的样本，同时记录了盲眼的数目。数据显示如下：

```
年龄:      20 35 45 55 70
No. 检      50 50 50 50 50
测:
No. 盲      6 17 26 37 44
眼:
```

我们想知道的是这些数据是否吻合 logistic 和 probit 模型，并且分别估计各个模型的 LD50，也就是一个男性居民盲眼的概率为50%时候的年龄。

如果 y 和 n 是年龄为 x 时的盲眼数目和检测样本数目，两种模型的形式都为 $y \sim B(n, F(\beta_0 + \beta_1 x))$ ，其中在 probit 模型中， $F(z) = \Phi(z)$ 是标准的正态分布函数，而在 logit 模型 (默认) 中， $F(z) = e^z / (1 + e^z)$ 。这两种模型中， $LD50 = -\beta_0 / \beta_1$ ，即分布函数的参数为0时所在的点。

第一步是把数据转换成数据框。

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

在 glm() 拟合二项式模型时，响应变量有三种可能性：

如果响应变量是向量，则假定操作二元 (binary) 数据，因此要求是0/1向量。

如果响应变量是双列矩阵，则假定第一列为试验成功的次数第二列为试验失败的次数。

如果响应变量是因子，则第一水平作为失败 (0) 考虑而其他的作为`成功'(1) 考虑。

我们采用的是第二种惯例。我们在数据框中增加了一个矩阵：

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

为了拟合这些模型，我们采用

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

既然 logit 的关联函数是默认的，因此我们可以在第二条命令中省略该参数。为了查看拟合结果，我们使用

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

两种模型都拟合的很好。为了计算 LD50，我们可以利用一个简单的函数：

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

从这些数据中得到的年龄分别是43.663年和 43.601年。

Poisson 模型

在 Poisson 族中，默认的关联函数是 log。在实际操作中，这一族常常用于拟合计数资料的 Poisson 对数线性模型。这些计数资料的实际分布往往符合二项式分布。这是一个非常重要而又庞大的话题，我们不想在这里深入展开。它构成了非-gaussian 广义模型内容的很大一部分。

有时候，实践中产生的 Poisson 数据在对数或者平方根转化后可当作正态数据处理。作为后者的另一种选择是，一个 Poisson 广义线性模型可以通过下面的例子拟合：

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

拟似然模型

对于所有的族，响应变量的方差依赖于均值并且拥有作为系数（multiplier）的尺度参数。方差对均值的依赖方式是响应分布的一个特性；例如对于poisson分布 $\text{Var}(y) = \mu$ 。

对于拟似然估计和推断，我们不是设定精确的响应分布而是设定关联函数和方差函数的形式。因为关联函数和方差函数都依赖于均值。既然拟似然估计和 gaussian 分布使用的技术非常相似，因此这一族顺带提供了一种用非标准关联函数或者方差函数拟合 gaussian 模型的方法。

例如，考虑非线性回归的拟合 $y = \theta_1 z_1 / (z_2 - \theta_2) + e$ 同样还可以写成 $y = 1 / (\beta_1 x_1 + \beta_2 x_2) + e$ 其中 $x_1 = z_2/z_1$, $x_2 = -1/x_1$, $\beta_1 = 1/\theta_1$, and $\beta_2 = \theta_2/\theta_1$ 。假如有适合的数据框，我们可以如下进行非线性拟合

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

如果需要的话，读者可以从其他手册或者帮助文档中得到更多的信息。

Next: [Some non-standard models](#), Previous: [Generalized linear models](#), Up: [Statistical models in R](#)

11.7 非线性最小二乘法和最大似然法模型

特定形式的非线性模型可以通过广义线性模型 (`glm()`) 拟合。但是许多时候，我们必须把非线性拟合的问题 作为一个非线性优化的问题解决。R的非线性优化程序是 `optim()` 和 `nlm()`。二者分别替换 S-Plus 的 `ms()` 和 `nlminb()`。我们通过搜寻 参数值使得缺乏度 (lack-of-fit) 最低，如 `nlm()` 就是通过循环调试各种参数值得到最优值。和线性回归不同，程序不一定会收敛到一个稳定值。`nlm()` 需要设定参数搜索的起始值，而参数估计是否收敛在很大程度上依赖于 起始值设置的质量。

[Least squares](#): 最小二乘法

[Maximum likelihood](#): 最大似然法

Next: [Maximum likelihood](#), Previous: [Nonlinear least squares and maximum likelihood models](#),
Up: [Nonlinear least squares and maximum likelihood models](#)

11.7.1 最小二乘法

拟合非线性模型的一种办法就是使误差平方和 (SSE) 或残差平方和最小。如果观测到的误差极似正态分布，这种方法是非常有效的。

下面是例子来自 Bates & Watts (1988)，51页。具体数据是：

```
> 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)
```

被拟合的模型是：

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

为了进行拟合，我们需要估计参数拟合起始值。一种判断合理起始值的办法把数据图形化，然后估计一些参数值，并且利用这些值初步绘制模型曲线。

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

当然，我们可以做的更好，但是初始值 200 和 0.1 已经足够了。现在做拟合：

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

拟合后，out\$minimum 是误差的平方和 (SSE)，out\$estimate 是参数的最小二乘估计值。为了得到参数估计过程中近似的标准误(SE)，我们可以：

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

上述命令中的2表示参数的个数。一个95%的信度区间可以通过 ± 1.96 SE 计算得到。我们可以把这个最小二乘拟合曲线画在一个图上：

```

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

```

标准包 `stats` 提供了许多用最小二乘法拟合非线性模型的扩充工具。我们刚刚拟合过的模型是 Michaelis-Menten 模型，因此可以利用下面的命令得到类似的结论。

```

> 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
212.68370711 0.06412123
residual sum-of-squares: 1195.449
> summary(fit)

```

Formula: $y \sim \text{SSmicmen}(x, Vm, K)$

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

Residual standard error: 10.93 on 10 degrees of freedom

Correlation of Parameter Estimates:

	Vm
K	0.7651

Previous: [Least squares](#), Up: [Nonlinear least squares and maximum likelihood models](#)

11.7.2 最大似然法

最大似然法 (Maximum likelihood) 也是一种非线性拟合办法。它甚至可以用于 误差非正态的数据。这种方法估计的参数 将会使得对数似然值最大或者负的对数似然值 最小。下面的例子来自 Dobson (1990), pp. 108 – 111。这个例子对剂量 - 响应数据拟合 logistic 模型 (当然也可以用 glm() 拟合)。数据是：

```
> 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)
```

要使负对数似然值最小，则：

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

我们选择一个适当的起始值，开始拟合：

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

拟合后，out\$minimum 就是负对数似然值，out\$estimate 就是最大似然拟合的参数值。为了得到拟合过程近似的标准误，我们可以：

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

一个 95% 信度期间可根据公式 $\pm 1.96 \text{ SE}$ 计算得到。

Previous: [Nonlinear least squares and maximum likelihood models](#), Up: [Statistical models in R](#)

11.8 一些非标准模型

我们简单提一下 R 里面某些用于某些特殊回归和 数据分析问题的 工具。

混合模型。用户捐献包 nlme 里面提供了 函数 lme() 和 nlme()。这些函数可以用于混合效应模型 (mixed-effects models) 的线性和非线性回归。也就是说 在线性和非线性回归中，一些系数受随机因素的影响。这些函数需要充分利用公式来描述 模型。

局部近似回归。函数 loess() 利用局部加权回归进行一个非参回归。这种回归对显示一组凌乱数据的趋势和 描述大数据集的整体情况非常 有用。

函数 loess 和投影跟踪回归 (projection pursuit regression) 的代码 一起放在标准包 stats 中。稳健回归。有多个函数可以用于 拟合回归模型，同时尽量不受数据中 极端值的影响。在推荐包 MASS 中的函数 lqs 为高稳健性的拟合提供了 最新的算法。另外，稳健性较低但统计学上高效的方法 同样可以在包 MASS 中得到，如函数 rlm

累加模型。这种技术期望可以通过决定变量的 平滑累加函数 (smooth additive function) 构建回归函数。一般来说，每个决定变量 都有一个平滑累加函数。用户捐献的包 acepack 里面的函数 avas 和 ace 以及包 mda 里面的函数 bruto 和 mars 为这种技术提供了一些例子。这种技术的一个扩充是用户捐献包 gam 和 mgcv 里面实现的 广义累加模型。

树型模型。除了利用外在的全局线性模型 预测和解释数据，还可以利用树型模型在决定变量的判断点上循环探索 数据的分叉现象。这样做会 把数据最终分成几个不同组，使得 组内尽可能相似而组间尽可能差异。这样常常会得到一些其他数据分析方法不能 产生的信息。

模型可以用一般的线性模型形式指定。该模型拟合函数 是 tree()，而且许多泛型函数，如 plot() 和 text() 都可以很好的用于树型模型拟合结果的 图形显示。

R 里面的树型模型函数是 通过用户捐献的包 rpart 和 tree 得到。

Next: [Packages](#), Previous: [Statistical models in R](#), Up: [Top](#)

12 图形工具

图形工具是 R 环境里面一个非常重要和多用途的组成部分。我们可以用这些图形工具显示 各种各样的统计图并且创建一些 全新的图。

图形工具既可交互式使用，也可以批处理使用。在许多情况下，交互式使用是最有效的。打开 R 时，它会启动一个图形设备驱动（device driver）。该驱动会打开特定的图形窗口（graphics window）以显示交互式的图片。尽管这些都是自动实现的，了解用于 UNIX 系统的 X11() 命令和 Windows 系统的 windows() 命令是非常有用的。

一旦设备驱动启动，R 绘图命令可以用来产生统计图或者设计全新的图形显示。

绘图命令可以分成了三个基本的类：

高级绘图命令在图形设备上产生一个新的图区，它可能包括坐标轴，标签，标题等等。
低级画图命令会在一个已经存在的图上加上更多的图形元素，如额外的点，线和标签。
交互式图形命令允许你交互式地用定点设备（如鼠标）在一个已经存在的图上添加图形信息或者提取图形信息。

此外，R 有一系列图形参数。这些图形参数可以修改从而定制你的图形环境。

这些信息仅仅描述了‘基本’图形绘制命令。在包 grid 里面有一个独立的图形子系统与基本图形包同时存在。它的功能非常的强大当然也非常地难用。有一个基于 grid 的推荐包 lattice 提供了产生多重面板图（multi-panel plot）的工具。这有点类似 S 里面 Trellis 系统。

[High-level plotting commands](#): 高级绘图命令

[Low-level plotting commands](#): 低级绘图命令

[Interacting with graphics](#): 交互式使用图形工具

[Using graphics parameters](#): 使用图形参数

[Graphics parameters](#): 图形参数

[Device drivers](#): 设备驱动

[Dynamic graphics](#): 动态图形

Next: [Low-level plotting commands](#), Previous: [Graphics](#), Up: [Graphics](#)

12.1 高级绘图命令

高级图形显示函数是用来产生输入数据的完整图片。其中适当的坐标轴，标签和标题会自动产生(除非你有其他要求)。高级图形显示命令会开始一个新的图区，必要时会清空当前的图区。

[The plot\(\) function](#): plot() 函数

[Displaying multivariate data](#): 显示多元数据

[Display graphics](#): 显示图形

[Arguments to high-level plotting functions](#): 高级图形函数的参数

Next: [Displaying multivariate data](#), Previous: [High-level plotting commands](#), Up: [High-level plotting commands](#)

12.1.1 plot() 函数

在 R 里面最常用的一个图形函数是 `plot()`。这是一个泛型函数：产生的图形依赖于第一个参数的类型或者类。

`plot(x, y)`

`plot(xy)`

如果 x 和 y 是向量，`plot(x, y)` 将产生 y 对 x 的散点图。单参数(第二种形式)可以实现同样的结果。对于单参数要么是一个包含两个分量 x 和 y 的列表要么是一个双列的矩阵。

`plot(x)`

如果 x 是一个时间序列，这将产生一个时间序列图。如果 x 是一个数值向量，它将产生一个向量值对该向量索引的图。如果 x 是一个复向量，它将会产生一个向量元素的虚部对实部的图。

`plot(f)`

`plot(f, y)`

f 实一个因子对象， y 是一个数值向量。第一种形式产生 f 的直方图；第二种形式产生 y 在 f 的各种水平下的盒状图。

`plot(df)`

`plot(~ expr)`

`plot(y ~ expr)`

df 是一个数据框， y 是任何对象， $expr$ 是一个对象名被 '+' 分割的表列(如， $a + b + c$)。前两种将产生数据框中变量的分布图（第一种形式）和一系列对象的分布图(第二种形式)。第三种形式将得到 y 相对 $expr$ 中各个对象画的图。

Next: [Display graphics](#), Previous: [The plot\(\) function](#), Up: [High-level plotting commands](#)

12.1.2 显示多元数据

R 为描绘多元数据提供了两个非常有用的函数。如果 X 是一个数值矩阵或者数据框，命令

```
> pairs(X)
```

将产生 X 的列之间两两相对的成对散点图阵列 (pairwise scatterplot matrix)。也就是说， X 的每一列相对 X 的所有其他列而产生 $n(n-1)$ 个图，并且把这些图以阵列个形式显示在图区。这个图形阵列的行列图形尺度一致。

用 *coplot* 处理三个或者四个变量的产生的图可能会更清晰。如果 a 和 b 是数值向量， c 是数值向量或者因子对象(长度都一致)，那么 下面的命令

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

将产生一系列在给定的 c 值下 a 对 b 的散点图。 c 是因子对象，这就简单的表示 a 在 c 各个水平下对 b 画的散点图。当 c 是数值向量，它将会被分割成一系列条件区间 (conditioning intervals)，对于任一区间，区间内 c 对应的 a ， b 值将绘制 a 对 b 的散点图。区间的数值和位置由 *coplot()* 的参数 *given.values=* 控制——函数 *co.intervals()* 用于选择区间。你还可以使用两个给定的变量，如下面的命令

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

它将会产生在任何在 c 和 d 联合区间内的 a 对 b 的散点图。

函数 *coplot()* 和 *pairs()* 都有一个参数 *panel=*。这个参数可以用来设置各个面板中的图形样式。默认值 *points()* 用来产生散点图，但是通过将低级命令用于两个向量 x 和 y 并赋值给参数 *panel=*，你可以产生任何你所期望的图。一个用于 *coplot* 的 *panel* 函数的例子是命令 *panel.smooth()*。

Next: [Arguments to high-level plotting functions](#), Previous: [Displaying multivariate data](#), Up: [High-level plotting commands](#)

12.1.3 图形显示

其他高级绘图函数可以产生多种类型的图。 一些例子：

`qqnorm(x)`

`qqline(x)`

`qqplot(x, y)`

分位比较图。第一种形式显示数值向量 x 相对期望的正态有序分位 (Normal order scores) 的比较图(正态分位图)；第二种形式是在这个图上加一条理论上的分位对分位的直线。第三种形式产生 x 的分位对 y 的分位的图以比较二者的分布是否一致。

`hist(x)`

`hist(x, nclass= n)`

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

产生数值向量 x 的柱状分布图。程序会自动选择适合的分类数目，但是可以通过设定参数 `nclass=` 来改变分类数。还有一种方法是，通过参数 `breaks=` 精确设置断点 (breakpoint)。如果给定参数 `probability=TRUE`，柱高度表示频率而不是频数。

`dotchart(x, ...)`

产生数据 x 的点图。在一个点图里面， y -轴给定 x 里面数据的标签， x -轴给出它们的值。这种图非常容易从视觉上看出在某个特定范围内的数据元素。

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

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

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

画三变量图。`image` 产生一个长方形的网格，用不同的颜色表示 z 的值，`contour` 以等高线 (contour line) 来表示 z 的值，`persp` 产生 3D 表面。

Previous: [Display graphics](#), Up: [High-level plotting commands](#)

12.1.4 高级图形命令的参数

传递给高级绘图函数的命令有许多，如：

`add=TRUE`

强制函数以低级绘图函数的形式运行，在当前的图上加载新的图形元素(仅适合于部分函数)。

`axes=FALSE`

禁止产生坐标轴—对你要想用函数 `axis()` 绘制个性化坐标轴非常有用。默认值是 `axes=TRUE`，表示 允许坐标轴。

`log="x"`

`log="y"`

`log="xy"`

让 x 轴，y 轴或者两者都成为对数坐标轴。这对 很多图都有效，但不是全部。

`type=`

参数 `type=` 控制输出图形（特别是线条）的类型：

`type="p"`

只显示点（默认）

`type="l"`

Plot lines

`type="b"`

(同时)显示点和线

`type="o"`

将点覆盖在线上

`type="h"`

绘制从点到零轴（x 轴）的垂直线(高密度)

`type="s"`

`type="S"`

步阶图。第一种形式，顶部垂直线匹配数据点；第二种形式，底部匹配。

`type="n"`

图形不显示。但是坐标轴仍然显示(默认)，并且坐标依然以数据设定。这个非常适合 随后用低级绘图函数画图。

`xlab=string`

`ylab=string`

设定 x 和 y 轴的标签。可以用这些参数修改 默认标签。默认标签常常是用于高级绘图函数中的 对象的名字。

`main=string`

图形标题，以大字体置于图形的顶部。

`sub=string`

子标题，以小字体放在 x-轴底部。

Next: [Interacting with graphics](#), Previous: [High-level plotting commands](#), Up: [Graphics](#)

12.2 低级图形函数

有些时候，高级图形函数不能精确产生你想要的图。此时，低级图形命令可以在当前图上精确增加一些额外信息 (如点，线或者文字)。

一些非常有用的低级图形命令是：

`points(x, y)`

`lines(x, y)`

在当前图上增加点或者连接线。`plot()` 的参数 `type=` 可用于这些函数 (`points()` 的默认值是 "p", `lines()` 是 "l")。

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

在图上给定的 `x, y` 位置添加文字。`labels` 经常是整数或者字符向量，此时，`labels[i]` 放在 `(x[i], y[i])` 处。默认值是 `1:length(x)`。

注意：这个命名常常用于下面的命令

```
> plot(x, y, type="n"); text(x, y, names)
```

图形参数 `type="n"` 不让点显示，但设置坐标轴。函数 `text()` 提供了一个特别的字符向量，因为相应点的位置上的符标由字符向量 `names` 设定。

`abline(a, b)`

`abline(h=y)`

`abline(v=x)`

`abline(lm.obj)`

在当前图上增加一个斜率为 `b` 截距为 `a` 的直线。`h=y` 可用于指定贯穿整个图的水平线高度的 `y`-坐标。`v=x` 相似地用于指定垂直线的 `x`-坐标。同样，`lm.obj` 可能是一个有长度为2的 `coefficients` 分量 (如模型拟合的结果) 的列表。该分量中依次含有截距和斜率。

`polygon(x, y, ...)`

绘制由 `(x, y)` 作为顶点定义的多边形。并且可以用剖面线处理 (可选)，或者在图形设备允许的情况下填充其他东西。

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

在当前图的特定位置增加图例 (`legend`)。标识字符，线条格式，颜色等都是被字符向量 `legend` 中的标签所注释。另外一个含有画图单位对应值的参数 `v` (一个和 `legend` 长度一致的向量) 是必须给定的：

```
legend(, fill=v)
```

填充盒子的颜色

`legend(, col=v)`

点或者线条的颜色

`legend(, lty=v)`

线条样式

`legend(, lwd=v)`

线条宽度

`legend(, pch=v)`

标识字符 (字符向量)

`title(main, sub)`

将 main 定义的标题以大字体的形式放在当前图的顶部，同时可以将 sub 定义的小标题以小字体的形式放在下部（可选）。

`axis(side, ...)`

在第一个参数(1 到 4，从底部顺时针方式数)定义的某一侧增加一个坐标轴。另一个参数控制坐标轴相对图区的位置，刻度位置 和 标签。这对调用参数设置为 `axes=FALSE` 的 `plot()` 函数后增加定制的坐标轴非常有用。

低级图形函数常常需要一些位置信息 (如，x 和 y 坐标) 来决定新的图形的放置。坐标是由用户坐标设置。而用户坐标根据先前高级图形命令定义以及由用户提供的数据决定。

其中 x 和 y 参数是必须的。如果提供一个同时含有参数 x 和 y 的列表对象作为参数也是允许的充分的。相似的，一个双列的矩阵同样是合法的输入。在这种情况下，函数 `locator()` (见后面的内容) 可以交互式地在一个图上设定位置。

[Mathematical annotation](#): 数学标注

[Hershey vector fonts](#): Hershey 矢量字体

Next: [Hershey vector fonts](#), Previous: [Low-level plotting commands](#), Up: [Low-level plotting commands](#)

12.2.1 数学标注

在某些情况下，在一个图上加上数学符号和公式是非常有用的。在 R 里面，这可以通过函数 *expression* 实现，而不是直接把数学符号和公式加在 *text*，*mtext*，*axis*，或者 *title* 的字符串里面。例如，下面的代码将写出二项式概率分布的公式：

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

更多的信息，包括 R 这些特性的列表，可以利用下面的命令得到：

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

Previous: [Mathematical annotation](#), Up: [Low-level plotting commands](#)

12.2.2 Hershey 矢量字体

函数 `text` 和 `contour` 可以使用 Hershey 矢量字体。使用 Hershey 矢量字体 有三个理由：

Hershey 字体 会产生更好的输出，特别在计算机屏幕上，或者用于旋转以及小字体时。
Hershey 字体 提供一些标准字体库没有的字体。如提供星座记号，地图符号 和天文学符号。

Hershey 字体提供西里尔字符（cyrillic）和日语字符（假名和日本汉字）。

更多的信息（包括 R 里面的 Hershey 字符列表）可以通过如下命令得到：

```
> help(Hershey)
> demo(Hershey)
> help(Japanese)
> demo(Japanese)
```

Next: [Using graphics parameters](#), Previous: [Low-level plotting commands](#), Up: [Graphics](#)

12.3 交互使用图形环境

R 同时提供了允许用户直接用鼠标在一个图上提取和提交信息的函数。其中最为简单的是函数 `locator()`：

`locator(n, type)`

等待用户用鼠标左键点击当前图上的特定位置。这个过程直到由 `n` (默认 512) 个点 被选择，或者另外一个鼠标键被点击了。参数 `type` 允许在被选择的点上画图并且有高级画图命令一样的效果；默认情况下不能画图。`locator()` 以双分量 `x` 和 `y` 的列表形式返回所选中点的位置信息。

`locator()` 常常没有参数。当我们很难设定一些图形元素（如图例和标签）在图上的放置位置时，交互式选定位置信息可能是一种非常好的办法。例如，在特异点（outlying point）的旁边标注一些提示信息，我们可以用下面的命令

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

(如果当前设备（如 postscript）不支持交互式使用，则 `locator()` 会被自动忽略。)

`identify(x, y, labels)`

允许用户将 `labels` 定义的标签（在 `labels` 为空时，默认为点的索引值）放置在 `x` 和 `y` (利用鼠标左键)决定的点旁边。当鼠标右键被点击时，返回选择的点的索引。

有时候我们想标定一个图上的一些特定点，而不是它们的位置。例如，我们可能期望用户能在图形显示上选择一些有意思的点，然后以某种方式处理。假定有两个数值向量 `x` 和 `y` 构成的一系列坐标 `(x, y)`，我们可以如下使用函数 `identify()`：

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

函数 `identify()` 自己不会标识，但允许用户简单的移动鼠标指针和在一个点附近点击鼠标左键。如果有一个点在鼠标指针附近，那么它将会把自己的索引值（也就是在 `x/y` 向量中的位置）标记在点的旁边。还有一种方案是，你可以通过 `identify()` 的参数 `labels` 设置其他的文字信息(如样本名字等)，并且可以通过参数 `plot = FALSE` 禁止标记重叠在一起。在这个过程结束时（见上面），`identify()` 返回所选点的索引值；你可以利用这些索引值提取原始向量 `x` 和 `y` 中的点信息。

Next: [Graphics parameters](#), Previous: [Interacting with graphics](#), Up: [Graphics](#)

12.4 使用图形参数

当创建图形时，特别是用于做陈述或者出版，R 的默认设置往往不能符合要求。但是你可以利用图形参数几乎可以定制任何你想显示的方式。R 拥有一个数目很大的图形参数列表。该列表包括控制线条样式，颜色，图形排列和文字对齐等方面的参数。每一个图形参数都有名字(如`col`，控制颜色)和值(如颜色值)。

每一个活动的设备都有一套独立的图形参数列表，和启动时各参数的默认值。图形参数可以用两种方式设定：要么是永久性的，影响所有访问当前设备的图形函数；要么是临时性的，仅仅影响特定的图形函数。

[The par\(\) function](#): par() 函数

[Arguments to graphics functions](#): 图形函数的参数

Next: [Arguments to graphics functions](#), Previous: [Using graphics parameters](#), Up: [Using graphics parameters](#)

12.4.1 Permanent changes: par() 函数

函数 par() 用于访问和修改当前图形设备的 参数列表。

par()

没有参数，将返回所有图形参数的列表和当前 设备的设定值。

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

设定一个字符串向量的参数，仅仅返回指定的 参数(同样是一个列表)。

par(col=4, lty=2)

给定参数(或者是一个列表参数)，设置指定图形参数的值，以列表的形式返回 参数的初试值。

通过函数 par() 设定图形参数会永久地改变 参数值，因为以后所有在当前设备中 调用的图形函数都会受这些设置值影响。 你可以用这种办法设定图形参数的“默认”值。这些默认值将会被 所有图形函数调用，除非你设定了其他值。

注意，调用 par() 常常会影响图形参数的全局值，这种情况甚至出现在函数内部调用 par()。 这种行为不是我们想要的 - 我们通常只是想 设定一些图形参数，绘制一些图片，然后恢复原始值以免影响到 其他用户的 R 会话。你可以通过保存 par() 的值来恢复初试值。当图形绘制结束后，你可以重新 载入这些保存的初试值¹。

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

```
... 图形命令 ...
```

```
> par(oldpar)
```

为保存和恢复 所有设定的²图形参数，可以使用

```
> oldpar <- par(no.readonly=TRUE)
```

```
... plotting commands ...
```

```
> par(oldpar)
```

Footnotes

[1] 译者注：有些版本会自动在绘图结束后恢复的原始设定。

[2] 一些图形参数，如当前设备的大小，仅仅是信息提示作用。

Previous: [The par\(\) function](#), Up: [Using graphics parameters](#)

12.4.2 临时性地改变：图形函数的参数

图形参数可以以命名参数的形式传递到(几乎)所有图形函数。这和函数 `par()` 设定的参数有同样的影响，除非参数的改变仅仅影响当前函数调用这个过程而不影响其他函数调用。例如：

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

这将产生一个以加号为标符的散点图。这个命令不会影响后面的图形命令的默认图标。

不过，这不是总是有效的，有些时候还是需要通过 `par()` 设定和重新设定图形参数。

Next: [Device drivers](#), Previous: [Using graphics parameters](#), Up: [Graphics](#)

12.5 图形参数列表

下面的内容将会具体描述一些常用的图形参数。R 帮助文档中关于 `par()` 函数的内容 会给出更简练的概述；这里会给出一个有点 具体的描述。

图形参数将会以下面的形式给出：

name=value

对参数影响的描述。*name* 是参数名，它可以用于 `par()` 或一个图形函数中。*value* 是你想给这个参数 设定的值。

注意 `axes` 不是图形参数而是 一些 `plot` 方法的参数：参见 `xaxt` 和 `yaxt`。

[Graphical elements](#): 图形元素

[Axes and tick marks](#): 坐标轴和刻度

[Figure margins](#): 图片边缘

[Multiple figure environment](#): 多重图形环境

Next: [Axes and tick marks](#), Previous: [Graphics parameters](#), Up: [Graphics parameters](#)

12.5.1 图形元素

R 图形由点，线，文本和多边形（闭合区域）构成。图形参数就是用来控制这些 *图形元素*（graphical element）如何绘制：

`pch="+"`
用于显示点的符标。默认值会随图形驱动不同而有些差异，不过常常是一个小圈。被显示的点有点高于或者低于适当的位置，除非你用位于中心的 "." 作为显示符标。

`pch=4`
当 `pch` 设定一个0到25之间的整数，一个特定的符标将会产生。为了查看各个符标的样子，可以采用下面的命令

```
> legend(locator(1), as.character(0:25), pch = 0:25)
```

这些位于21到25间的符标看上去是前面符标的重复，不过它们可以用不同的颜色显示：查看 `points` 的帮助文件和例子。

此外，`pch` 可以是在 32:255 范围内的字符和数字。它们会以字符的当前字体形式显示。

`lty=2`
线条类型。尽管不是所有的图形设备都支持图形类型（就是在支持的设备中，也有点差异），但是类型1常常是实线，类型0是不可见的，类型2和其他常常是点线和虚线，或者是点划线。

`lwd=2`
线条宽度。以“标准”线条宽度的倍数设定线条宽度。坐标轴线条和利用函数 `lines()` 等产生的线条都会受影响。不是所有的设备支持这种特性，并且一些设备会对线条宽度有所限制的。

`col=2`
点，线，文本，填充区域和图像的颜色设置。值可以是来自当前调色板的数字（见 `?palette`）或者一个命名的颜色。

`col.axis`

`col.lab`

`col.main`

`col.sub`

分别用于轴标注，x 和 y 轴的标签，主标题和次标题等的颜色设置。

`font=2`
整数是用来指定用于文中的字体类型。一般情况下，设备驱动设定的 1 对应于纯文本，2

对应粗体，3 对应斜体，4 对应粗斜体，5 对应符号体(包括希腊字母)。

font.axis

font.lab

font.main

font.sub

分别用于轴标注，x 和 y 轴标签，主标题和次标题的字体设置。

adj=-0.1

调整文本对于图形的相对位置。0 表示左对其，1 表示右对其，0.5 表示图形位置的水平居中。实际值表示出现在图形位置左侧的文本宽度的比率，因此 -0.1 表示将会在文本和图形位置间留下文本宽度的 10%。

cex=1.5

字符扩张率。这个值表示期望字符(包括绘图字符)大小相对默认大小的比率。

Next: [Figure margins](#), Previous: [Graphical elements](#), Up: [Graphics parameters](#)

12.5.2 轴和刻度

许多 R 的高级图形含有坐标轴，此外你可以用低级图形函数 `axis()` 设置你自己的坐标轴。坐标轴包括三个主要部分：**轴线** (axis line) (线条格式由图形参数 `lty` 控制)，**刻度** (tick mark) (划分轴线上的刻度) 和 **刻度标记** (tick label) (标记刻度上的单位)。这些部分可以通过下面的图形参数设置。

`lab=c(5, 7, 12)`

前两个参数分别是 x 和 y 轴期望的刻度间隔数目。第三个参数 刻度标记的字符长度 (包括小数点)。这个参数设的太小会导致所有的标记变成一样的数字。

`las=1`

刻度标记的方向。0 表示总是平行于坐标轴，1 表示总是水平，以及 2 表示总是垂直于坐标轴。

`mgp=c(3, 1, 0)`

三个坐标成分的位置。第一个参数是轴标签相对轴位置的距离，以文本行作为参照单位的。第二个参数表示刻度标记的距离，最后一个参数是轴位置到轴线的距离(常常是0)。正值表示在图形外，负值表示在图形内。

`tck=0.01`

刻度的长度，以画图区域大小的比率作为度量。当 `tck` 比较小(小于 0.5)，x 和 y 轴上的刻度强制大小一致。值为1时，给出网格线。负值时刻度在图形外。`tck=0.01` 和 `mgp=c(1,-1.5,0)` 表示内部刻度。

`xaxs="r"`

`yaxs="i"`

分别设定 x 和 y 轴的形式。"i" (内在的) 和 "r" (默认) 形式的刻度都适合数据的范围，但是 "r" 形式的刻度会在刻度范围两边留一些空隙(S 还有一些在 R 里面没有实现的刻度形式)。

Next: [Multiple figure environment](#), Previous: [Axes and tick marks](#), Up: [Graphics parameters](#)

12.5.3 图片边缘

R 里面一个单独的图被称为 figure，并且还包括一个被边缘（可能包括轴标签，标题等）和坐标轴包围的绘图区域（plot region）。

控制图版面的图形参数有：

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

分别表示底部，左边，上部和右边的空间，单位是英寸。

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

和 mai 相似，只是度量单位采用文本行。

mar 和 mai 是等价的因为二者可以相互转换。这些参数的默认值常常有点大；右侧边缘常常是不需要的，没有标题时，顶部边缘也不是需要的。底部和左侧足以用来绘制坐标轴和刻度标记。需要说明的时，默认值没有考虑到设备层面；例如，用设定参数height=4 的 postscript() 驱动会导致一个图有 50% 的边缘。这时可以显式地设置 mar 或者 mai。当使用多重图形是（下面会讨论），边缘会减少，不过在很多图共享一页时这还是不够的。

Previous: [Figure margins](#), Up: [Graphics parameters](#)

12.5.4 多重图形环境

R 允许你在一个单页上创建一个 $n \times m$ 图形阵列。每一个图有它自己的边缘，图形阵列是被可选的 **外边缘** (outer margin) 包围着。具体可见下面的图。

和多重图形相关的图形参数如下：

```
mfc col=c(3, 2)
```

```
mfc row=c(2, 4)
```

设置多重图形阵列的大小。第一个值是行数；第二个值是列数。这两种形式的差异在于 `mfc col` 使得图片以列的方式放置；`mfc row` 则以行的方式放置。

例图中版面设计可以通过设置 `mfc row=c(3,2)` 实现；图中显示了画了四个图后该页的情况。

设置任一可以减小符号或者文本基本大小参数 (`par("cex")`和设备的点大小)的递减因子。在一个双行列的版面中，基本大小是通过因子 0.83 递减的；如果有三个或者三个以上的行列时，递减因子是 0.66。

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

当前图片在一个多重图形环境中的位置。前两个数字是当前图的行列编号；后面两个数字是多重图形阵列的行列编号。设置这个参数将会调整图形的位置。你甚至可以用和本页上不均等大小图 (unequally-sized figures) 的实际值不等的值设定后面两个参数。

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

当前图片在页上的位置。值分别表示左侧，右侧，下侧和上侧的边缘宽度，并且以左侧底部作为参照点得到的页面百分比。演示值表示一个在页面右下侧的图形。可以通过设定这些参数而将图片放置在一个页面的任何地方。如果你想在当前页上增加一个图，可以使用参数 `new=TRUE` (和 `S` 不同)。

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

```
omi=c(0, 0, 0.8, 0)
```

外边缘的大小。同 `mar` 和 `mai` 相似，第一个以文本行度量，第二个以英寸度量。从底部边缘算起，以顺时针方向设置值。

外边缘对页面形式的标题特别有用。文本可以通过函数 `mtext()` 和参数 `outer=TRUE` 加在外边缘。默认没有外边缘，但是你可以用 `oma` 或 `omi` 显式地定义它们。

更为复杂的多重图形排列可以采用函数 `split.screen()` 和 `layout()`，以及包 `grid` 和 `lattice`。

Next: [Dynamic graphics](#), Previous: [Graphics parameters](#), Up: [Graphics](#)

12.6 设备驱动

R 几乎可以在任何形式的显示和打印设备上产生图片(包括各种显示质量的)。但是在显示图片前，R 必须指明采用什么设备处理图片。这可以通过通过启动设备驱动 (device driver) 来实现。设备驱动的目的 在于将 R 的画图指令(如 “draw a line, ”) 转化成特定设备可以识别的指令。

我们可以调用设备驱动函数来启动设备驱动。每一种设备驱动都有对应的函数：输入 help (Devices) 可以得到它们的列表。例如，发出命令

```
> postscript()
```

会使得所有图形以 PostScript 的形式输出。一些常用的设备驱动是：

X11()

用 UNIX 类型的系统的 X11 桌面系统

windows()

用于 Windows 系统

quartz()

用于 MacOS X 系统

postscript()

用于 PostScript 打印机，或者创建 PostScript 文件。

pdf()

创建可以插入 PDF 文件中 PDF 文件

png()

创建 PNG 位图文件。(不总是有效的：参考它的帮助文件)

jpeg()

创建 JPEG 位图文件，非常适用于影像 (image) 输出。(不总是有效的：参考它的帮助文件)

当你结束使用一个设备时，用下面的命令 终止设备驱动

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

这将确信设备完全结束了；例如在硬拷贝设备中，这 将保证每一页都已完成并且已经发送给打印机。(在常规会话的结束，这些过程会自动完成。)

[PostScript diagrams for typeset documents](#): 排版文档用的 PostScript 图表

[Multiple graphics devices](#): 多重图形设备

Next: [Multiple graphics devices](#), Previous: [Device drivers](#), Up: [Device drivers](#)

12.6.1 排版文档用的 PostScript 图表

通过将参数 `file` 传递给 `postscript()` 设备驱动函数，你可以在你选择的文件中以 PostScript 形式保存图片。图片会被横行放置。你可以设定参数 `horizontal=FALSE` 使得图片纵向放置。图形大小将通过参数 `width` 和 `height` 设置 (图形会被适当调整以适合参数要求)。例如，命令

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

将产生一个高为5英寸以 PostScript 格式编码的图片文件。特别要提醒的是，如果命令中的文件名已经存在，原来的文件将会被覆盖。在同一目录下的 R 会话中，一定要小心先前创建的文件名字。

PostScript 输出的图片常常用于插入其他文档中。此时最好采用*封装的* (encapsulated) PostScript 图片：R 常常产生构造性的输出 (conformant output)，但是只在 `onefile=FALSE` 参数提供时才以标记输出。这种奇怪的用法和 S 不兼容：它实际上表示输出将会是单页的 (这是 EPSF 规范的一部分)。因此对于内插图片的生成可以采用下面的命令^[1]：

```
> postscript("plot1.eps", horizontal=FALSE, onefile=FALSE,  
  height=8, width=6, pointsize=10)
```

Footnotes

[1] 译者注：我是用 GSview 查看 PostScript 文件的，所以对以 PostScript 格式的图片我都用这个软件直接转换了。效果还是不错的。

Previous: [PostScript diagrams for typeset documents](#), Up: [Device drivers](#)

12.6.2 多重图形设备

在 R 的高级使用里面，几个图形设备的同时使用是非常有用的。不过，在一个时间点，只有一个图形设备可以接受图形命令，这个设备就是所谓的 *当前设备*。当多个设备同时启动时，它们将形成一个以名字作为标识符的有限任务序列。

操作多重图形设备的主要命令和意义 列举如下：

```
X11()
  [UNIX]
windows()
win.printer()
win.metafile()
  [Windows]
quartz()
  [MacOS X]
postscript()
pdf()
...
```

任何一次调用设备驱动函数都会打开一个图形设备，该设备自动加到一个设备列表的后面。这个设备现在就是当前设备，所有的图形都会输出到这个设备上。（一些平台可能有更多的设备可用。）

```
dev.list()
  返回所有处于活动状态的设备的编号和名字。位于列表位置 1 的设备常常是零设备（null device），它不接受任何图形命令。
```

```
dev.next()
dev.prev()
  分别返回当前设备的后一个或者前一个设备的名字和编号。
```

```
dev.set(which=k)
  将设备列表中位置  $k$  的设备作为当前设备。返回该设备的标签和编号。
```

```
dev.off(k)
  终止设备列表中位置  $k$  的设备。对于一些设备，如 postscript 设备，该命令要么立即打印该文件，要么正确地结束文件。这依赖于设备时如何初试设置。
```

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

```
dev.print(device, ..., which=k)
  为设备  $k$  创建一份拷贝。这里，device 是一个设备函数，如 postscript，必要时可以加入由 ... 指定的额外参数。dev.print 类似，但是被拷贝的设备会迅速关闭，因此收尾活动，如
```

打印硬拷贝等，会立刻执行。

`graphics.off()`

关闭设备列表中除零设备外的所有设备。

Previous: [Device drivers](#), Up: [Graphics](#)

12.7 动态图形

R 目前还没有提供内置的函数处理动态图形，如旋转散点云，“粉刷”（brush）（交互式强调）图形等。但是，强大的动态图形函数可以从 Swayne, Cook 和 Buja 设计的系统 XGobi 中得到

<http://www.research.att.com/areas/stat/xgobi/>

并且可以通过 R 的包 xgobi 直接访问。XGobi 可以在 UNIX 或者 Windows 的 X 桌面系统中运行，R 都有相关的接口。

XGobi 的一个衍生版本 GGobi 现在正在开发，可以参见：<http://www.ggobi.org>.

Next: [A sample session](#), Previous: [Graphics](#), Up: [Top](#)

13 包

所有的 R 函数和数据集是保存在 (packages) 里面的。只有当一个包被载入时，它的内容才可以被访问。这样做一是为了高效 (完整的列表会耗去大量的内存并且增加搜索的时间)，一是为了帮助包的开发者防止命名和其他代码的中的名字冲突。开发包的过程在 [Creating R packages \(Writing R Extensions\)](#) 里面有详细的介绍。这里，我们仅仅从用户的角度来描述这个问题。

可以使用下面的命令查看你当前环境中安装的包

```
> library()
```

命令中没有参数。为了载入某个特别的包(如包 boot ，其中包含的函数来自 Davison & Hinkley (1997))，使用如下命令

```
> library(boot)
```

用户可以使用函数 CRAN.packages() 连接因特网 (也可以通过 Windows 和 RAqua 的图形界面上的 Packages 菜单访问)，并且可以自动更新和安装包。

为了查看当前有那些包载入了，可以用

```
> search()
```

产生搜索列表。有一些列表虽然被载入但不会出现在搜索列表中(see [Namespaces](#))。

为了查看已经安装的包的所有可以访问的帮助主题列表，可以使用

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

这将启动一个 HTML 形式的帮助系统，然后通过 Reference 部分链接到所有包的列表。

[Standard packages](#): 标准包

[Contributed packages and CRAN](#): 捐献包和 CRAN

[Namespaces](#): 命名空间

Next: [Contributed packages and CRAN](#), Previous: [Packages](#), Up: [Packages](#)

13.1 标准包

标准(基本)包构成 R 原代码的一个重要部分。它们包括允许 R 工作的的基本函数，和本文档中描述的数据集，标准统计和图形工具。在任何 R 的安装版本中，它们都会被自动获得。See [R packages \(R FAQ\)](#), 可以得到一个完整的列表。

Next: [Namespaces](#), Previous: [Standard packages](#), Up: [Packages](#)

13.2 捐献包和 CRAN

许多不同作者为 R 捐献了好几百个包。其中一些包实现了特定的统计方法，另外一些 给予数据和硬件的访问接口，其他则 作为教科书的补充材料。一些包(*推荐包*)和二进制形式的 R 捆绑发布。更多的可以从 CRAN (<http://CRAN.R-project.org/> 和它的镜像)和其他一些 资源，如 Bioconductor (<http://www.bioconductor.org/>) 下载得到。 *R FAQ* 含有一个已经发布包的列表，不过可以得到的包的清单变更是非常迅速的。

Previous: [Contributed packages and CRAN](#), Up: [Packages](#)

13.3 命名空间

包有自己的命名空间（namespaces），并且现在所有基本的和推荐的包都依赖于包 datasets。命名空间主要三个作用：它们允许包的作者隐藏函数和数据，即只允许内部用户使用，它们防止函数在一个用户（或其他包的作者）使用相同名字时被破坏，它们提供了一种访问特定包的某个对象的方法。

例如，`t()` 是 R 里面的转置函数，但是用户可能会定义一个函数叫 `t`。命名空间防止用户定义的函数居先和破坏矩阵转置的函数。

有两个操作符和命名空间相关。双冒号操作符 `::` 选择一个特定命名空间得到的函数定义。在上面的例子中，转置函数总是可以通过 `base::t` 使用，因为它是在包 `base` 中定义的。一个包中的函数可以通过这种方式访问。

三冒号操作符 `:::` 可能会出现在一些 R 代码中：它有点像双冒号操作符，但可以访问隐藏对象。用户还可能使用函数 `getAnywhere()`，它会搜索多重包。

包常常是包之间依赖的（inter-dependent），载入其中一个可能会引起其他包的自动载入。上面提到的冒号操作符同样会引起相关包的自动载入。当有命名空间的包自动载入时，它们不会被自动加入搜索列表。

Next: [Invoking R](#), Previous: [Packages](#), Up: [Top](#)

Appendix A 一个演示会话

下面的会话让你在操作中对 R 环境的一些特性有个简单的了解。你对系统的许多特性开始时可能有点不熟悉和困惑，但这些迷惑会很快消失的。

登录，启动你的桌面系统。

\$ R

以适当的方式启动 R¹。

R 程序开始，并且有一段引导语。

(在 R 里面，左边的提示符将不会被显示防止混淆。)

help.start()

启动 HTML 形式的在线帮助(使用你的计算机里面可用的浏览器)。你可以用鼠标 点击上面的链接。

最小化帮助窗口，进入下一部分。

x <- rnorm(50)

y <- rnorm(x)

产生两个伪正态随机数向量 x 和 y。

plot(x, y)

画二维散点图。一个图形窗口会自动出现。

ls()

查看当前工作空间里面的 R 对象。

rm(x, y)

去掉不再需要的对象。(清空)。

x <- 1:20

等价于 $x = (1, 2, \dots, 20)$ 。

w <- 1 + sqrt(x)/2

标准差的'权重'向量。

dummy <- data.frame(x=x, y= x + rnorm(x)*w)

dummy

创建一个由 x 和 y 构成的双列数据框，查看它们。

fm <- lm(y ~ x, data=dummy)

summary(fm)

拟合 y 对 x 的简单线性回归，查看 分析结果。

fm1 <- lm(y ~ x, data=dummy, weight=1/w^2)


```
summary(fm1)
```

现在我们已经知道标准差，做一个加权回归。

```
attach(dummy)
```

让数据框中的列项可以像一般的变量那样使用。

```
lrf <- lowess(x, y)
```

做一个非参局部回归。

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

标准散点图。

```
lines(x, lrf$y)
```

增加局部回归曲线。

```
abline(0, 1, lty=3)
```

真正的回归曲线：(截距 0，斜率 1)。

```
abline(coef(fm))
```

无权重回归曲线。

```
abline(coef(fm1), col = "red")
```

加权回归曲线。

```
detach()
```

将数据框从搜索路径中去除。

```
plot(fitted(fm), resid(fm),
```

```
  xlab="Fitted values",
```

```
  ylab="Residuals",
```

```
  main="Residuals vs Fitted")
```

一个检验异方差性 (heteroscedasticity) 的标准回归诊断图。你可以看见吗？

```
qqnorm(resid(fm), main="Residuals Rankit Plot")
```

用正态分位数图检验数据的偏度 (skewness)，峰度 (kurtosis) 和异常值 (outlier)。(这里没有多大的用途，只是演示一下而已。)

```
rm(fm, fm1, lrf, x, dummy)
```

再次清空。

第二部分将研究 Michaelson 和 Morley 测量光速的经典实验。这个数据集可以从对象 morley 中得到，但是我们从中读出数据以演示函数 read.table 的作用。

```
filepath <- system.file("data", "morley.tab", package="datasets")
```

```
filepath
```

得到文件路径。

```
file.show(filepath)
```

可选。查看文件内容。

```
mm <- read.table(filepath)
```

```
mm
```

以数据框的形式读取 Michaelson 和 Morley 的数据，并且查看。数据由五次实验(Expt 列)，每次运行 20 次 (Run 列)的观测得到。数据框中的 sl 是光速的记录。这些数据以适当形式编码。

```
mm$Expt <- factor(mm$Expt)
```

```
mm$Run <- factor(mm$Run)
  将 Expt 和 Run 改为因子。
attach(mm)
  让数据在位置 3 (默认) 可见 (即可以直接访问)。
plot(Expt, Speed, main="Speed of Light Data", xlab="Experiment No.")
  用简单的盒状图比较五次实验。
fm <- aov(Speed ~ Run + Expt, data=mm)
summary(fm)
  分析随机区组, `runs' 和 `experiments' 作为因子。
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
  拟合忽略 `runs' 的子模型, 并且对模型更改前后 进行方差分析。
detach()
rm(fm, fm0)
  在进行下面工作前, 清空数据。
```

我们现在查看更有趣的图形显示特性：等高线和影像显示。

```
x <- seq(-pi, pi, len=50)
y <- x
  x 是一个在 区间  $[-\pi, \pi]$  内等间距的50个元素的向量, y 类似。
f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))
  f 是一个方阵, 行列分别被 x 和 y 索引, 对应的值是函数  $\cos(y)/(1 + x^2)$  的结果。
oldpar <- par(no.readonly = TRUE)
par(pty="s")
  保存图形参数, 设定图形区域为 “ 正方形 ”。
contour(x, y, f)
contour(x, y, f, nlevels=15, add=TRUE)
  绘制 f 的等高线; 增加一些曲线显示细节。
fa <- (f-t(f))/2
  fa 是 f 的 “ 非对称部分 ” (t() 是转置 函数)。
contour(x, y, fa, nlevels=15)
  画等高线, ...
par(oldpar)
  ... 恢复原始的图形参数。
image(x, y, f)
image(x, y, fa)
  绘制一些高密度的影像显示, (如果你想要, 你可以保存 它的硬拷贝), ...
objects(); rm(x, y, f, fa)
  ... 在继续下一步前, 清空数据。
```

R 可以做复数运算。

```
th <- seq(-pi, pi, len=100)
```

```
z <- exp(1i*th)
```

1i 表示复数 i。

```
par(pty="s")
```

```
plot(z, type="l")
```

图形参数是复数时，表示虚部对实部画图。这可能是一个圆。

```
w <- rnorm(100) + rnorm(100)*1i
```

假定我们想在这个圆里面随机抽样。一种方法 将让复数的虚部和实部值是标准正态随机数 ...

```
w <- ifelse(Mod(w) > 1, 1/w, w)
```

... 将圆外的点映射成它们的倒数。

```
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
```

```
lines(z)
```

所有的点都在圆中，但分布不是均匀的。

```
w <- sqrt(runif(100))*exp(2*pi*runif(100)*1i)
```

```
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
```

```
lines(z)
```

第二种方法采用均匀分布。现在圆盘中的点 看上去均匀多了。

```
rm(th, w, z)
```

再次清空。

```
q()
```

离开 R 程序。你可能被提示是否保存 R 工作空间，不过对于一个调试性的会话，你可能不想保存它。

Footnotes

[1] 译者注：Windows 用户直接点击 R 的快捷图标进入。和其他 Windows 程序一样操作。

Next: [The command-line editor](#), Previous: [A sample session](#), Up: [Top](#)

Appendix B 调用 R

[Invoking R from the command line](#): 以命令行调用 R

[Invoking R under Windows](#): 在 Windows 下调用 R

[Invoking R under Mac OS X](#): 在 Mac OS X 下调用 R

Next: [Invoking R under Windows](#), Previous: [Invoking R](#), Up: [Invoking R](#)

B.1 以命令行调用 R

在 UNIX 或者 Windows 的命令模式下操作时，命令 R 可以用下面的形式来启动 R 的主程序

```
R [options] [<infile] [>outfile],
```

我们还可以通过 R CMD 接口来启动。该接口是各种不想被“直接”访问的(如，处理 R 文档格式的文件或者操作另外加上去的包) R 工具的一种包装。

在 UNIX 下面，你要确信环境变量 TMPDIR 还没有设置或者它已指向了一个合理的路径来创建临时文件和目录。

许多可选项是用来控制一个 R 会话的起始和终止时的事务。启动机制将会在下面描述(查看帮助文档中关于 Startup 的主题，下面的内容有一些 Windows-特异的细节)。

除非给定 `--no-environ`，R 将会自动搜索用户和位置文件来设置环境变量。位置文件名字是由环境变量 `R_ENVIRON` 指定的；如果这没有设定，默认是 `$R_HOME/etc/Renviron.site` (如果它存在)。用户文件是在当前或者用户根目录下面的 `.Renviron`。这些文件包含以 `name=value` 格式赋值的行 (`help(Startup)` 可以得到更准确的描述)。你想设定的变量可能还包括 `R_PAPERSIZE` (默认的页面类型/大小)，`R_PRINTCMD` (默认的打印命令) 和 `R_LIBS` (指定搜索添加包 R 库的树型列表)。

R 搜索位置广泛的启动初试设置文件，除非你已经设置了命令可选项 `--no-site-file`。配置文件的名字可以通过环境变量 `R_PROFILE` 访问。如果该变量没有设置，将会使用默认的文件 `$R_HOME/etc/Rprofile.site` (如果它存在)。

然后，除非 `--no-init-file` 被设定，R 会依次搜索当前目录或者用户根目录一个叫 `.Rprofile` 的文件然后载入它。

如果有 `.RData` 文件，它将会被载入(除非指定 `--no-restore` 或者 `--no-restore-data`)。

最后如果 `.First` 存在，它也会被执行。这个函数(在 R 对话结束时运行的 `.Last` 也一样)可以在适当的启动配置文件中定义，或者放在文件 `.RData` 中。

此外，有可选项控制 R 进程的内存使用情况(参考在线帮助关于 Memory 的主题)。用户一般很少用到这些设置，除非你想用尽 R 的所有内存。

R 接受如下的命令可选项。

`--help`

-h

向标准输出打印简短的帮助信息并且顺利退出。

--version

向标准输出打印版本信息并且顺利退出。

--encoding=*enc*

对控制台或者 stdin 的输入指定编码方式。这和 iconv 类似：查看它的帮助文件。

RHOME

向标准输出打印 R 的“根目录”并且顺利退出。独立于 shell 脚本和 man 页，R 安装过程中会把所有东西(执行文件，包，等等)都放在这个目录下。

--save

--no-save

设置 R 会话结束时是否保存数据集。如果在一次交互式会话中，二者都没有给定，那么当用户键入 q() 退出时，会要求决定是否保存数据；在非交互式运行时，其中一种方式要求设定。

--no-environ

不读取任何用户文件去设定环境变量。

--no-site-file

启动时不读取任何位置文件。

--no-init-file

启动时不读取任何用户设置文件。

--restore

--no-restore

--no-restore-data

控制是否在启动时恢复镜像文件(指的是 R 启动时目录下的文件 .RData)。默认是恢复。
(--no-restore 暗示所有指定的可选项 --no-restore-*。)

--no-restore-history

控制是否在启动时恢复历史文件(常常是 R 启动目录下的文件 .Rhistory，可以通过环境变量 R_HISTFILE 重新设定)。默认是恢复。

--vanilla

同时设定了 --no-save, --no-environ --no-site-file, --no-init-file 和 --no-restore。

--no-readline

(仅用于 UNIX) 禁止通过 readline 进行命令行编辑。这对在 Emacs 中用 ESS (“ Emacs Speaks Statistics ”) 包运行 R 非常有用。See [The command-line editor](#), 可以得到更多的信息。

--ess

(仅用于 Windows)通过 R-inferior-mode 模式在 ESS 中启动 Rterm。

--min-vsize=*N*

--max-vsize=*N*

通过设定“向量堆栈”(vector heap)大小为 *N* 字节，指定可变大小对象的最大和最小内存占有量。*N* 要么是一个整数，要么是以分别表示 'Giga' (2^{30})，'Mega' (2^{20})，'Kilo' (2^{10} ，计算机)，和 'kilo' (1000，标准)的 G，M，K 和 k 结尾的整数。

--min-nspace=*N*

--max-nspace=*N*

通过设定“cons 单元”(cons cells)的数目为 *N*，指定不变大小对象的最大和最小内存占有

有量。参看前面关于 N 的细节内容。在32位机里面，一个 cons 单元等于28个字节，而在64位机里面，常常是 56 个字节。

--max-ppsize= N

设定指针保护栈的最大大小为 N 个位点。默认值为 10000，有时需要增加该值以适应大的复杂计算。现在允许的最大设定值为 100000。

--max-mem-size= N

(仅适用于 Windows) 设定用于 R 对象和工作空间内存总量的限度。默认设置为 1024Mb 和机器上的物理内存，但最小要求是 16Mb。

--quiet

--silent

-q

不显示起始的版权和欢迎信息。

--slave

让 R 尽可能平静地运行。该选项用于支持利用 R 结果作为输入的程序。

--verbose

在 R 选项 verbose 设定为 TRUE 时，显示运行过程中更多的信息。R 使用这个选项控制诊断信息的输出。

--debugger=*name*

-d *name*

(仅用于 UNIX) 通过调试器 *name* 运行 R。注意在这种情况下，进一步的命令行可选项会被忽略，不过可以在 R 从内部调试器开始执行时给定。

--gui=*type*

-g *type*

(仅用于 UNIX) 使用 *type* 作为用户图形接口(注意这也包括交互式图形接口)。现在，可能用于 *type* 的值是 X11 (默认，假定已经安装 Tcl/Tk)，Tk，gnome (假定 GNOME 已经安装) 和 none。

--args

这个标记会让剩余的命令行忽略：这对想用 commandArgs() 得到返回值非常的有用。

注意，可以用常规的方法(用 < 和 >) 重定向输入输出。除了 Windows 9X/ME 系统，警告和错误信息会被传给标准错误信息流 (stderr)。

命令 R CMD 允许调用各种可以连接 R 的不能“直接”访问的工具。一般的形式为

R CMD *command args*

其中 *command* 是工具名，*args* 是传递给它的参数。

现在，下面的工具可以调用

BATCH

批处理模式运行 R。

COMPILE

(仅用于 UNIX)用 R 编译文件。

SHLIB

为动态载入创建共享库 (shared library) 。

INSTALL

安装额外添加的包。

REMOVE

去除额外添加的包。

build

创建(也就是打包)额外添加的包。

check

选择额外添加的包。

LINK

(仅用于 UNIX)创建可执行程序的前端。

Rprof

R 的后处理 (Post-process) 设置文件。

Rdconv

将 Rd 格式转换成各种其他格式，包括 HTML，Nroff，LaTeX，纯文本和 S 文档格式。

Rd2dvi

将 Rd 格式转换成 DVI/PDF 格式。

Rd2txt

将 Rd 格式转换成文本格式。

Sd2Rd

将 S 文件转换成 Rd 格式。

config

(仅用于 UNIX) 获得配置信息。

使用

R CMD *command* --help

获得可以用 R CMD 接口访问的工具 的使用信息。

Next: [Invoking R under Mac OS X](#), Previous: [Invoking R from the command line](#), Up: [Invoking R](#)

B.2 在 Windows 下调用 R

有两种方法可以在 Windows 下运行 R。在终端窗口 (如cmd.exe, command.com, 或者能力更强的脚本) 里面 运行 R.exe 或者更直接的 Rterm.exe时, 前面提到的方法可能都可以使用它们主要用于批处理)。对于交互式用户, 可以使用基于控制台的图形界面 (Rgui.exe)。

在 Windows 下面的启动过程和 UNIX 中的过程非常相似, 但是必须指明'根目录' (home directory), 因为 Windows 系统不会定义这个目录。如果定义了环境变量 R_USER, 则根目录由这个变量指定。接下来, 如果环境变量 HOME 定义了, 它也会指定根目录。通过这两个用户可控的设置, R 期望可以找到用户定义的根目录。它首先采用 Windows 系统的"私人"目录 (典型的私人目录如 Windows XP 系统中的 C:\Documents and Settings\username\My Documents)。如果这样不成功并且环境变量 HOMEDRIVE 和 HOMEPATH 都设定了 (它们常常会在 Windows NT/2000/XP 系统下定义), 这些将会决定根目录。如果这些都没有设定, 那么就把 起始目录当成根目录。

环境变量可以以 *name=value* 形式 放在命令行的尾部。

下面的命令行可选项在 调用 RGui.exe 时可用。

--mdi

--sdi

--no-mdi

决定以 MDI 程序(默认, 在一个主窗口中可以运行多个子窗口)还是SDI 软件 (图形, 页面和控制台的多重高层次窗口)的形式运行 Rgui。

--debug

允许 Rgui 的菜单项 “ Break to debugger ” 可用, 并且可以在命令行处理中设定断点。

在 Windows 系统中, 你可以用 R CMD 运行你指定的 *.bat 和 *.exe 而不需要内部命令。下面的环境变量可用于这个命令中: R_HOME, R_VERSION, R_CMD, R_OSTYPE, PATH, PERL5LIB, 和 TEXINPUTS。例如, 如果你设置的路径中有 latex.exe, 那么

```
R CMD latex.exe mydoc
```

将对 mydoc.tex 运行 LaTeX 并且把 R 的 share/texmf 宏包路径加到环境变量 TEXINPUTS 中。

Previous: [Invoking R under Windows](#), Up: [Invoking R](#)

B.3 在 Mac OS X 下调用 R

在 Mac OS X 下面运行 R 有两种方式。在 Terminal.app 窗口调用 R 时，前面提到的方法都可用。同样还有基于控制台的图形界面 (R.app)。它默认安装在系统的 Applications 文件夹中。它是一个标准的双击启动的 Mac OS X 程序。

Mac OS X 的启动流程和 UNIX 下的流程非常相似。`根目录'在 R.framework 中设置。启动目录和当前目录 设置为用户的根目录，当然，你可以在图形界面中的参数选择窗口中重新设定一个不同的启动目录。

Next: [Function and variable index](#), Previous: [Invoking R](#), Up: [Top](#)

Appendix C 命令行编辑器

C.1 预备工作

如果你的 UNIX 系统已经安装了 GNU readline 库，那么 R 配置中允许在 UNIX 下编译 R 代码，调用内置的 命令行编辑器，编辑和重新调用以前用过的命令。注意：该附录提到的接口不是用于 UNIX 系统的 GNOME 接口，而仅仅用于标准的命令行 接口。

如果启动时设置了参数 `--no-readline` (使用 ESS 时非常有用¹)，则该命令不可用。

Windows 版本的 R 有简单的命令行编辑功能；见 GUI 界面的 Help 菜单下的 Console，已经描述 Rterm.exe 的命令行操作的文件 README.Rterm。

当使用 readline 写 R 命令时，下面描述的函数可用。

这些函数常常是控制字符或者是元字符（Meta character）。控制字符，如 Control-m 表示同时按住 <CTRL> 和 <m> 键，并且以 C-m 形式表示。元字符，如 Meta-b 表示同时按住 <META>² 和 键，下面以 M-b 形式记录。如果你的终端没有 <META> 键，你可以用 ESC 开始的两个字符序列 键入元字符。因此对于 M-b，你可以键入 <ESC>。ESC 字符序列在 有真正元键的终端也是允许的。注意这种情况 对元字符有特殊意义的。

C.2 编辑

R 保存你键入的命令行的历史，包括错误的命令。历史文件中的命令可以被重新调用，修改 以新的命令的形式重新提交。在 Emacs-形式的命令行编辑中，任何直接的输入 都会将字符直接插入到你所编辑的命令中，并且取代光标右侧的字符。vi 输入模式是通过 M-i 或 M-a 启动，字符可以被键入并且通过键入 <ESC> 结束输入模式。

任何时候键入 <RET> 都会使得命令 重新被提交。

其他的编辑命令在下面的表中有所总结。

C.3 命令行编辑总结

重新调用命令和垂直移动

C-p

跳到前一个命令(回溯历史文件)。

C-n

跳到下一个命令(前溯历史文件)。

C-r *text*

搜索含有字符串 *text* 的最后一个命令。

在大多数终端，你可以使用上下键分别代替 C-p 和 C-n。

水平移动指针

C-a

回到命令行开头。

C-e

跳到命令行结束。

M-b

回溯一个单词。

M-f

前溯一个单词。

C-b

回溯一个字符。

C-f

前溯一个字符。

在大多数终端，你可以使用左右键分别代替 C-b 和 C-f。

编辑和再提交

text

在光标处插入文本 *text*。

C-f *text*

在光标后插入 *text*。

删除前面的字符(光标左侧)。

C-d

删除光标处的字符。

M-d

删除光标处单词以外的部分，并且“保存”它们。

C-k

删除光标到命令结束的部分，并且“保存”它们。

C-y

插入最后“保存”的文本。

C-t

转置光标处的文本。

M-l

将字符转换成小写字符。

M-c

将单词转换成大写。

<RET>

重新向 R 提交命令。

最后的 <RET> 命令将会终止命令行编辑。

Footnotes

[1] 'Emacs Speaks Statistics' 包；参见 URL <http://ESS.R-project.org>

[2] 在 PC 键盘上，它常常是 Alt 键，偶尔是 'Windows' 键。

Next: [Concept index](#), Previous: [The command-line editor](#), Up: [Top](#)

Appendix D 函数和变量索引

!: [Logical vectors](#)

!=: [Logical vectors](#)

%*%: [Multiplication](#)

%o%: [The outer product of two arrays](#)

&: [Logical vectors](#)

&&: [Conditional execution](#)

*: [Vector arithmetic](#)

+: [Vector arithmetic](#)

-: [Vector arithmetic](#)

.: [Updating fitted models](#)

.First: [Customizing the environment](#)

.Last: [Customizing the environment](#)

/: [Vector arithmetic](#)

:: [Generating regular sequences](#)

::: [Namespaces](#)

:::: [Namespaces](#)

<: [Logical vectors](#)

<<-: [Scope](#)

<=: [Logical vectors](#)

==: [Logical vectors](#)

>: [Logical vectors](#)

>=: [Logical vectors](#)

?: [Getting help](#)

^: [Vector arithmetic](#)

abline: [Low-level plotting commands](#)

ace: [Some non-standard models](#)

add1: [Updating fitted models](#)

anova: [Generic functions for extracting model information](#)

anova: [ANOVA tables](#)

aov: [Analysis of variance and model comparison](#)

aperm: [Generalized transpose of an array](#)

[array](#): [The array\(\) function](#)
[as.data.frame](#): [Making data frames](#)
[as.vector](#): [The concatenation function c\(\) with arrays](#)
[attach](#): [attach\(\) and detach\(\)](#)
[attr](#): [Getting and setting attributes](#)
[attributes](#): [Getting and setting attributes](#)
[avas](#): [Some non-standard models](#)
[axis](#): [Low-level plotting commands](#)
[boxplot](#): [One- and two-sample tests](#)
[break](#): [Repetitive execution](#)
[bruto](#): [Some non-standard models](#)
[c](#): [Character vectors](#)
[C](#): [Contrasts](#)
[c](#): [Concatenating lists](#)
[c](#): [Vectors and assignment](#)
[c](#): [The concatenation function c\(\) with arrays](#)
[cbind](#): [Forming partitioned matrices](#)
[coef](#): [Generic functions for extracting model information](#)
[coefficients](#): [Generic functions for extracting model information](#)
[contour](#): [Display graphics](#)
[contrasts](#): [Contrasts](#)
[coplot](#): [Displaying multivariate data](#)
[cos](#): [Vector arithmetic](#)
[crossprod](#): [Multiplication](#)
[crossprod](#): [Index arrays](#)
[cut](#): [Frequency tables from factors](#)
[data](#): [Accessing builtin datasets](#)
[data.frame](#): [Making data frames](#)
[density](#): [Examining the distribution of a set of data](#)
[det](#): [Singular value decomposition and determinants](#)
[detach](#): [attach\(\) and detach\(\)](#)
[determinant](#): [Singular value decomposition and determinants](#)
[dev.list](#): [Multiple graphics devices](#)
[dev.next](#): [Multiple graphics devices](#)
[dev.off](#): [Multiple graphics devices](#)
[dev.prev](#): [Multiple graphics devices](#)
[dev.set](#): [Multiple graphics devices](#)
[deviance](#): [Generic functions for extracting model information](#)

[diag: Multiplication](#)

[dim: Arrays](#)

[dotchart: Display graphics](#)

[drop1: Updating fitted models](#)

[ecdf: Examining the distribution of a set of data](#)

[edit: Editing data](#)

[eigen: Eigenvalues and eigenvectors](#)

[else: Conditional execution](#)

[Error: Analysis of variance and model comparison](#)

[example: Getting help](#)

[exp: Vector arithmetic](#)

[F: Logical vectors](#)

[factor: Factors](#)

[FALSE: Logical vectors](#)

[fivenum: Examining the distribution of a set of data](#)

[for: Repetitive execution](#)

[formula: Generic functions for extracting model information](#)

[function: Writing your own functions](#)

[glm: The glm\(\) function](#)

[help: Getting help](#)

[help.search: Getting help](#)

[help.start: Getting help](#)

[hist: Examining the distribution of a set of data](#)

[hist: Display graphics](#)

[identify: Interacting with graphics](#)

[if: Conditional execution](#)

[ifelse: Conditional execution](#)

[image: Display graphics](#)

[is.na: Missing values](#)

[is.nan: Missing values](#)

[jpeg: Device drivers](#)

[ks.test: Examining the distribution of a set of data](#)

[legend: Low-level plotting commands](#)

[length: The intrinsic attributes mode and length](#)

[length: Vector arithmetic](#)

[levels: Factors](#)

[lines: Low-level plotting commands](#)

[list: Lists](#)

[lm: Linear models](#)

[lme: Some non-standard models](#)

[locator: Interacting with graphics](#)

[loess: Some non-standard models](#)

[log: Vector arithmetic](#)

[lqs: Some non-standard models](#)

[lsfit: Least squares fitting and the QR decomposition](#)

[mars: Some non-standard models](#)

[max: Vector arithmetic](#)

[mean: Vector arithmetic](#)

[min: Vector arithmetic](#)

[mode: The intrinsic attributes mode and length](#)

[NA: Missing values](#)

[NaN: Missing values](#)

[ncol: Matrix facilities](#)

[next: Repetitive execution](#)

[nlm: Maximum likelihood](#)

[nlm: Nonlinear least squares and maximum likelihood models](#)

[nlm: Least squares](#)

[nlme: Some non-standard models](#)

[nrow: Matrix facilities](#)

[optim: Nonlinear least squares and maximum likelihood models](#)

[order: Vector arithmetic](#)

[ordered: Ordered factors](#)

[outer: The outer product of two arrays](#)

[pairs: Displaying multivariate data](#)

[par: The par\(\) function](#)

[paste: Character vectors](#)

[pdf: Device drivers](#)

[persp: Display graphics](#)

[plot: Generic functions for extracting model information](#)

[plot: The plot\(\) function](#)

[pmax: Vector arithmetic](#)

[pmin: Vector arithmetic](#)

[png: Device drivers](#)

[points: Low-level plotting commands](#)

[polygon: Low-level plotting commands](#)

[postscript: Device drivers](#)

[predict](#): [Generic functions for extracting model information](#)
[print](#): [Generic functions for extracting model information](#)
[prod](#): [Vector arithmetic](#)
[qqline](#): [Examining the distribution of a set of data](#)
[qqline](#): [Display graphics](#)
[qqnorm](#): [Display graphics](#)
[qqnorm](#): [Examining the distribution of a set of data](#)
[qqplot](#): [Display graphics](#)
[qr](#): [Least squares fitting and the QR decomposition](#)
[quartz](#): [Device drivers](#)
[range](#): [Vector arithmetic](#)
[rbind](#): [Forming partitioned matrices](#)
[read.table](#): [The read.table\(\) function](#)
[rep](#): [Generating regular sequences](#)
[repeat](#): [Repetitive execution](#)
[resid](#): [Generic functions for extracting model information](#)
[residuals](#): [Generic functions for extracting model information](#)
[rlm](#): [Some non-standard models](#)
[rm](#): [Data permanency and removing objects](#)
[scan](#): [The scan\(\) function](#)
[sd](#): [The function tapply\(\) and ragged arrays](#)
[search](#): [Managing the search path](#)
[seq](#): [Generating regular sequences](#)
[shapiro.test](#): [Examining the distribution of a set of data](#)
[sin](#): [Vector arithmetic](#)
[sink](#): [Executing commands from or diverting output to a file](#)
[solve](#): [Linear equations and inversion](#)
[sort](#): [Vector arithmetic](#)
[source](#): [Executing commands from or diverting output to a file](#)
[split](#): [Repetitive execution](#)
[sqrt](#): [Vector arithmetic](#)
[stem](#): [Examining the distribution of a set of data](#)
[step](#): [Updating fitted models](#)
[step](#): [Generic functions for extracting model information](#)
[sum](#): [Vector arithmetic](#)
[summary](#): [Generic functions for extracting model information](#)
[summary](#): [Examining the distribution of a set of data](#)
[svd](#): [Singular value decomposition and determinants](#)

[T: Logical vectors](#)

[t: Generalized transpose of an array](#)

[t.test: One- and two-sample tests](#)

[table: Frequency tables from factors](#)

[table: Index arrays](#)

[tan: Vector arithmetic](#)

[tapply: The function tapply\(\) and ragged arrays](#)

[text: Low-level plotting commands](#)

[title: Low-level plotting commands](#)

[tree: Some non-standard models](#)

[TRUE: Logical vectors](#)

[unclass: The class of an object](#)

[update: Updating fitted models](#)

[var: The function tapply\(\) and ragged arrays](#)

[var: Vector arithmetic](#)

[var.test: One- and two-sample tests](#)

[vector: Vectors and assignment](#)

[while: Repetitive execution](#)

[wilcox.test: One- and two-sample tests](#)

[windows: Device drivers](#)

[X11: Device drivers](#)

[|: Logical vectors](#)

[||: Conditional execution](#)

[~: Formulae for statistical models](#)

Next: [References](#), Previous: [Function and variable index](#), Up: [Top](#)

Appendix E 概念索引

[CRAN: Contributed packages and CRAN](#)

[Kolmogorov-Smirnov 检验。: Examining the distribution of a set of data](#)

[QR 分解: Least squares fitting and the QR decomposition](#)

[Shapiro-Wilk 检验: Examining the distribution of a set of data](#)

[t 检验: One- and two-sample tests](#)

[Wilcoxon 检验: One- and two-sample tests](#)

[二元操作符: Defining new binary operators](#)

[从文件中读取数据: Reading data from files](#)

[作用域: Scope](#)

[公式: Formulae for statistical models](#)

[分位比较图: Examining the distribution of a set of data](#)

[列表: Lists](#)

[列表连接: Concatenating lists](#)

[删除对象: Data permanency and removing objects](#)

[制表: Frequency tables from factors](#)

[动态图形: Dynamic graphics](#)

[包: Packages](#)

[包: R and statistics](#)

[单样本和双样本检验: One- and two-sample tests](#)

[参数命名: Named arguments and defaults](#)

[向量: Simple manipulations numbers and vectors](#)

[命名空间: Namespaces](#)

[因子: Contrasts](#)

[因子: Factors](#)

[图形参数: The par\(\) function](#)

[图形设备驱动: Device drivers](#)

[奇异值分解: Singular value decomposition and determinants](#)

[字符向量: Character vectors](#)

[定制环境: Customizing the environment](#)

[定向输入和输出: Executing commands from or diverting output to a file](#)

[密度估计: Examining the distribution of a set of data](#)

[对照: Contrasts](#)

[对象: Objects](#)

[局部近似回归: Some non-standard models](#)

[属性: Objects](#)

[工作空间: Data permanency and removing objects](#)

[广义线性模型: Generalized linear models](#)

[循环使用原则: Vector arithmetic](#)

[循环使用原则: The recycling rule](#)

[循环和条件控制: Loops and conditional execution](#)

[成组表达式: Grouped expressions](#)

[控制语句: Control statements](#)

[搜索路径: Managing the search path](#)

[数据框: Data frames](#)

[数组: Arrays](#)

[数组的外积: The outer product of two arrays](#)

[数组的广义转置: Generalized transpose of an array](#)

[数组索引: Array indexing](#)

[方差分析: Analysis of variance and model comparison](#)

[族: Families](#)

[更新拟合模型: Updating fitted models](#)

[最大似然法: Maximum likelihood](#)

[最小二乘法拟合: Least squares fitting and the QR decomposition](#)

[有序因子: Factors](#)

[有序因子: Contrasts](#)

[树型模型: Some non-standard models](#)

[概率分布: Probability distributions](#)

[正则序列: Generating regular sequences](#)

[泛型函数: Object orientation](#)

[混合模型: Some non-standard models](#)

[特征值和特征向量: Eigenvalues and eigenvectors](#)

[盒状图: One- and two-sample tests](#)

[矩阵: Arrays](#)

[矩阵相乘: Multiplication](#)

[稳健回归: Some non-standard models](#)

[算术函数和运算符: Vector arithmetic](#)

[类: The class of an object](#)

[类: Object orientation](#)

[索引向量: Index vectors](#)

累加模型: [Some non-standard models](#)

线性方程: [Linear equations and inversion](#)

线性模型: [Linear models](#)

经验累积分布函数: [Examining the distribution of a set of data](#)

统计模型: [Statistical models in R](#)

编写函数: [Writing your own functions](#)

缺省值: [Missing values](#)

行列式: [Singular value decomposition and determinants](#)

访问内置数据: [Accessing builtin datasets](#)

赋值: [Vectors and assignment](#)

非线性最小二乘法: [Nonlinear least squares and maximum likelihood models](#)

面向对象: [Object orientation](#)

默认值: [Named arguments and defaults](#)

Previous: [Concept index](#), Up: [Top](#)

Appendix F 参考文献

D. M. Bates and D. G. Watts (1988), *Nonlinear Regression Analysis and Its Applications*. John Wiley & Sons, New York.

Richard A. Becker, John M. Chambers and Allan R. Wilks (1988), *The New S Language*. Chapman & Hall, New York. This book is often called the “ *Blue Book* ” .

John M. Chambers and Trevor J. Hastie eds. (1992), *Statistical Models in S*. Chapman & Hall, New York. This is also called the “ *White Book* ” .

John M. Chambers (1998) *Programming with Data*. Springer, New York. This is also called the “ *Green Book* ” .

A. C. Davison and D. V. Hinkley (1997), *Bootstrap Methods and Their Applications*, Cambridge University Press.

Annette J. Dobson (1990), *An Introduction to Generalized Linear Models*, Chapman and Hall, London.

Peter McCullagh and John A. Nelder (1989), *Generalized Linear Models*. Second edition, Chapman and Hall, London.

John A. Rice (1995), *Mathematical Statistics and Data Analysis*. Second edition. Duxbury Press, Belmont, CA.

S. D. Silvey (1970), *Statistical Inference*. Penguin, London.

Next: [Word from the Translator](#), Previous: [Preface](#), Up: [Preface](#)

版权声明

这一部分是我单独列出来的，下面的内容是原文档的版权说明。为了保持原义，我就原文拷贝。

Copyright © 1990 W. N. Venables

Copyright © 1992 W. N. Venables & D. M. Smith

Copyright © 1997 R. Gentleman & R. Ihaka

Copyright © 1997, 1998 M. Maechler

Copyright © 1999 – 2005 R Development Core Team

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the R Development Core Team.

参考译文如下（具体以英文原文为准）：

版权 © 1990 W. N. Venables

版权 © 1992 W. N. Venables & D. M. Smith

版权 © 1997 R. Gentleman & R. Ihaka

版权 © 1997, 1998 M. Maechler

版权 © 1999 – 2005 R Development Core Team

在遵守并包含本文档版权声明的前提下，制作和发布本文档的完整拷贝是允许的。并且，所有这些拷贝均受到本许可声明的保护。

在遵守上述完整拷贝版本有关版权声明的前提下，拷贝和发布基于本文档完整拷贝的修改版本是允许的，并且，发布所有通过修改本文档而得到的工作成果，须使用与本文档的许可声明一致的许可声明。

在遵守上述修改版本版权声明的前提下，拷贝和发布本文档其它语言的翻译版本是允许的，如果本许可声明有经 R 核心开发小组（R Development Core Team）核准的当地化译本，则遵循当地化译本。

关于本中文翻译文档的版权声明：

本文档为自由文档（GNU FDL），在GNU自由文档许可证（<http://www.gnu.org/copyleft/fdl.html>）下发布，不明示或者暗示有任何保证。本文档可以自由复制，修改，散布，但请保留使用许可声明。