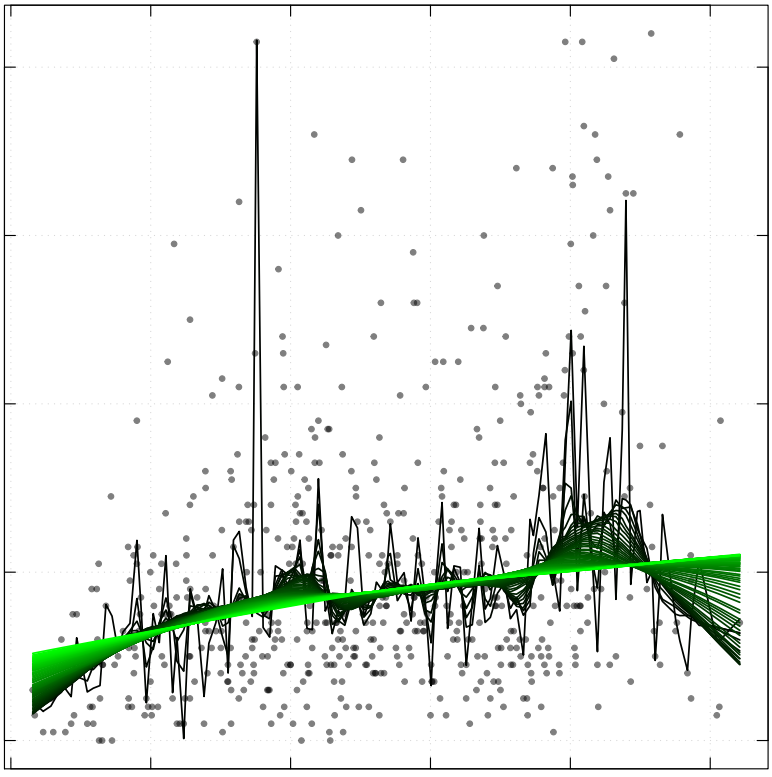
现代统计图形

谢益辉

2010 年 2 月 22 日



版权声明

本书电子版采用Creative Commons（简称CC）许可证“署名—非商业性使用—相同方式共享2.5 中国大陆”，该许可证的全文可以从http://creativecommons.org/licenses/by-nc-sa/2.5/cn/获得；一份普通人可以理解的法律文本概要可以从http://creativecommons.org/licenses/by-nc-sa/2.5/cn/legalcode获得。

责任权利

本CC许可证赋予读者复制、发行、展览、表演、放映、广播或通过信息网络传播本作品以及创作演绎作品的自由，而无需向原作者征求许可或支付任何费用；本许可证与出版社版权独立，因此复制、传播或演绎本作品也无须征求出版社许可。您需要遵循的条件是：

- 声明原作者的署名（Attribution）：不得将本作品归为自己的劳动
- 不得将本作品用于商业目的（Noncommercial）
- 基于本作品的演绎作品须遵守同样许可证发布（Share Alike）

作者采用CC许可证的考虑主要有三点：

- 让读者能免费、自由获得本书，节省经济支出；在有网络和电子文档的时代，我们应该充分利用这些工具的优势，如传播快捷、读者交流反馈方便（以便提高书籍质量）等
- 版权的本来意义不在于控制所有权，它只不过是为了对原创者的一种署名激励；如果版权的存在妨碍了知识的传播，那么本人认为版权就没有太大的意义；CC许可证中的“非商业”和“同样许可证”限制条款在书籍出版14年后会自动取消，即读者可以用于商业目的或更改至其它许可证；CC许可证规定的14年似乎是很长的时间，但读者须知：通常的版权只有在原作者去世后50年才会被取消！换句话说，版权告诉我们一个很深刻的哲理：长寿是很重要的

- 自由软件用户往往有某种痴狂的特征，而这种痴狂往往来源于自由软件的分享精神；R语言让本人受益颇多，这本书可视作是对它的一种回馈；既然R语言是自由的，那么本书也将尽量“自由”

特别声明

尽管CC许可证没有限制作品的传播方式，但本作者不愿看到本书被任何人以论坛附件的方式发布在任何论坛，原因是本书稿尚未成熟，或许有诸多不完善之处甚至严重错误，作者在不断更新中，若要传播本书稿给他人，请仅仅给出本书的原始链接`http://yihui.name/cn/publication/`，否则作者对传播过程中的错误概不负责。

捐赠说明

如果本书对您有任何帮助，您不妨考虑为“统计之都”网站（自愿）捐赠：`http://cos.name/donate/`；捐赠对象非作者本人，但本作者将从一定程度上根据捐赠情况判断本书工作的价值。捐赠所得将用于推广统计学和自由统计软件。捐赠之后请及时告知网站管理人员：admin@cos.name。

致谢

本书写作过程中收到了不少读者反馈，在此一并感谢。感谢魏太云对本书文字的校对和建议；感谢赵彦云老师对本书书名和写作风格的建议；感谢李皞对写lattice系统和rgl包的提议；

目录

插图

表格

序言

代序一

代序二

作者导读

我们常说“一图胜千言”，然而现实情况是我们了解的图形种类太少、使用的作图工具缺乏灵活性，这在很大程度上制约了统计图形的发展，使得统计图形在数据分析中应有的潜力没有被充分挖掘出来，正是这样的背景催生了本书的写作。

本书根据统计图形制作的需要，将所有内容分为七章：第一章先选择性回顾历史上的四幅著名统计图形，在欣赏前人智慧的基础上说明统计图形在社会生活的各个方面所能体现的价值；第二章介绍图形工具，本书主要以R软件为制图工具，因此本章也会介绍关于R语言的一些基础知识；第三章详细介绍图形参数，用以对图形进行细节调整，若读者对图形细节要求不高则可直接跳过这一章；第四章讲解基础图形元素的使用，包括点、线、多边形、颜色和文本等，本章会给那些期望能自定义统计图形的读者提供方便的解决方案；第五章是本书的一大核心，集中介绍讲解现有的统计图形种类如直方图、条形图、茎叶图、饼图、箱线图等，此外还会引入若干较特殊和不太常见的图形种类和数据图示方法，并且配以相应的统计数据分析实例深入说明统计图形的用法和含义；第六章介绍基础图形系统（base graphics）之外的其它图形系统如grid、lattice和ggplot2，第七章和第八章对各种统计图形分别从模型方法和数据类型的角度给出一些实例并作出归纳总结，以便让读者清楚区分统计图形运用的条件和场合；附录中

给出了一些作图方面的技巧。

本书的所有图形文件和程序代码可以从作者的个人主页下载（`http://yihui.name/cn/publication/`），另外本书也配有相应的R包**MSG**(Xie 2010)，使用说明参见附录D。阅读本书过程中若有任何疑问请Email联系作者：xie@yihui.name。

关于本书中R程序代码，记号说明如下：

`>` 表示一段R程序的开始；在R中可以用`options(prompt = '> ')`设置程序行的起始符号，默认为“`> `”；后文中只要遇到这个标记，则说明该标记*之后*的程序语句可以直接在R中运行（读者在运行书中的示例时不要把这个符号也敲进命令行中）

`+` 续行符；当一段程序在某一行中没有完整显示出来时，就会折到下一行，此时R会以“`+`”表示程序语句上不完整，正在继续

`[.]` 其中“`.`”表示一个整数，中括号括上一个整数表示R程序输出的行号，比如`[1]`表示这是第1行输出

`#` 表示R程序注释，即不会被执行的语句（只是为了增强程序的可读性而做的“标记”）

另外，本书R代码以等宽斜体排版，代码中带有程序起始符以及续行符，左侧以数字标出行号，例：

```
1 > plot(x, y)
2 > plot(cumsum(rnorm(80)), type = "l", col = "blue",
3 +     xlab = "Step", ylab = "Brownian Motion")
```

代码的输出以等宽正体排版，左侧不带行号提示，例：

```
> library(MSG)
> data(PlantCounts)
> summary(lm(counts ~ altitude, PlantCounts))

Call:
lm(formula = counts ~ altitude, data = PlantCounts)

Residuals:
    Min      1Q  Median      3Q     Max
-27.631 -10.272  -3.777   6.491  65.788
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -26.67224    6.45287  -4.133 4.09e-05 ***
altitude      0.08159    0.01107   7.372 5.61e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.3 on 598 degrees of freedom
Multiple R-squared: 0.08331,        Adjusted R-squared: 0.08178
F-statistic: 54.35 on 1 and 598 DF,  p-value: 5.615e-13
```

正文中的代码以等宽正体表示，如“`inline R code`”，函数名称以斜体表示，如“*function()*”，对象类名称和参数名称用无衬线字体表示，如“class using sans serif”，R程序包用粗体表示，如“**package**”。

顾炎武在《日知录》中曾有一句话：“形而上者谓之道，形与下者谓之器。”对本书来讲，统计作图的（计算机）技术本身即为“器”，而数据处理以及统计图形的灵活应用则为“道”。本书的写作目的正是希望能够基于“器”的练习和启发，让读者在统计数据处理和分析中真正得“道”，使统计图形在数据的探索分析中发挥福尔摩斯探案般的功效。

谢益辉
于爱荷华州立大学
2010年2月

第一章　历史

> “这易如反掌，”他说，“我看到你左脚穿的那只鞋的内侧，也就是炉火刚好照到的地方，皮面上有六道几乎平行的划痕。显然，这些划痕是有人为了去掉沾在鞋跟上的泥疙瘩，极其粗心大意地顺着鞋跟刮泥而造成的。因此，现在你就明白了我得出的这两个推断：其一，你曾经在恶劣的天气外出过；其二，你穿的皮靴上面的特别难看的划痕是伦敦的女佣所为。至于你开业行医，这么说吧，如果一位先生走进我的房间，身上带有碘的气味，右手食指上有硝酸银腐蚀的黑斑，高顶黑色大礼帽的右侧鼓起一块，那里面藏着听诊器，而我不断言他是医务界的一位活跃分子，那我不是太迟钝了吗？”
>
> — 柯南·道尔《波希米亚丑闻》

统计图形的意义在于引导我们观察到统计数据中的信息。用著名统计学家John Tukey的话来讲，就是“图形的最大价值就是使我们注意到我们从来没有料到过的信息”（The greatest value of a picture is when it forces us to notice what we never expected to see）。从这个意义上讲，统计图形的重要性自然不言而喻。

在统计图形历史上，能够达到“揭示人们不曾料到的信息”这种高度的图形并不多，那么这里我们首先欣赏四幅前人创造出的名垂青史的统计图形。

1.1　饼图和线图的起源

饼图和线图都是当今社会中常用的统计图形，它们是由有着“统计图形奠基人”之称的苏格兰工程师兼政治经济学家William Playfair发明的。

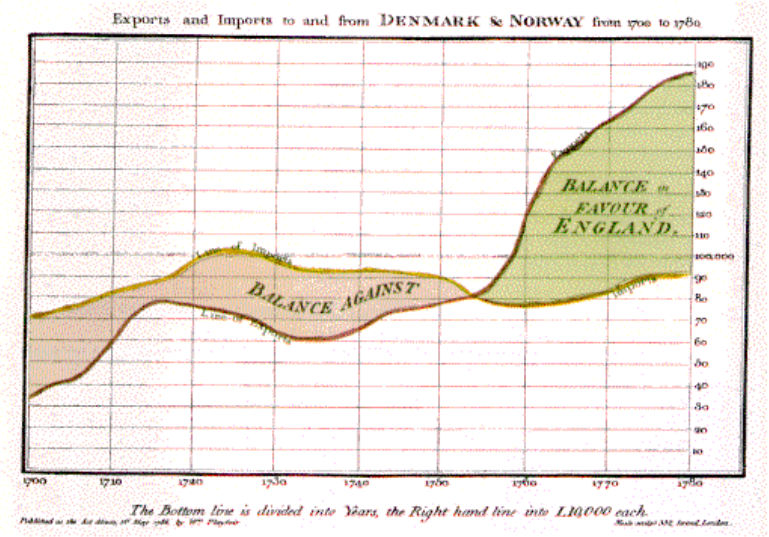


图 *1.1: Playfair (1786)*绘制的线图。这幅图主要展示了*1700*年至*1780*年间英格兰的进出口时序数据，左边表明了对外贸易对英格兰不利，而随着时间发展，大约*1752*年后，对外贸易逐渐变得有利。图片来源：*http://en.wikipedia.org/wiki/William_Playfair*

在“The Commercial and Political Atlas”(Playfair 1786)一书中，他用线图展示了英格兰自1700年至1780年间的进出口数据（如图1.1），从图中可以很清楚看出对英格兰有利和不利（即顺差、逆差）的年份；而在“The Statistical Breviary”(Playfair 1801)一书中，他第一次使用了饼图来展示一些欧洲国家的领土比例，图1.2即为史上第一例饼图。从左下方的饼图中我们可以清楚看出当时的土耳其帝国分别在亚洲、欧洲和非洲的领土面积比例。这两幅图在今天看来似乎没有什么惊世骇俗之处，但在当时统计图形种类极为稀少的年代，能以这种方式清晰展示数据结构，也实属难能可贵。事实上，除了这两种图形之外，他还发明了条形图和圆环图。

1.2 霍乱传染之谜

袭击欧洲大城市最严重的天灾要数19世纪的霍乱。由于垃圾没有得到及时清理，清洁水源的缺少，以及下水管道系统的不足，伦敦成为无药可医的流行病滋生的最佳地点。公众一致认为霍乱是由空气传播的，如果呼

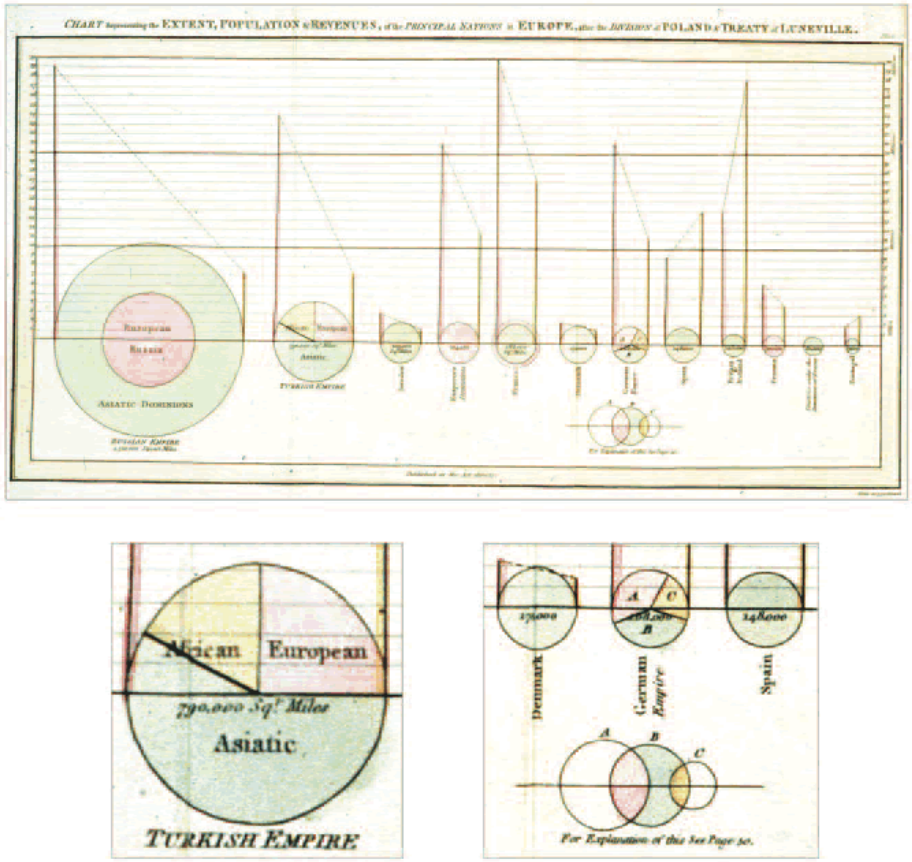


图 1.2: *Playfair (1801)*绘制的饼图。这是历史上第一幅饼图，描述了法国大革命前后一些欧洲国家的统计数据。上方的大图展示了各个国家的领土面积（和圆圈成比例）以及人口（左垂线）、税收（右垂线）、国土在各大洲分布比例等数据，两条垂线连线的斜率可表示税负的轻重（这一点颇有争议，因为斜率与圆的半径有关）。左下方的饼图展示了土耳其帝国在三大洲的国土面积分布。图片来源：*http://www.psych.utoronto.ca/users/spence/Spence%202005.pdf*

吸到了“瘴气”或者接触到霍乱患者，就会染上这种病。医生兼自学成才的科学家John Snow对这个观点颇为怀疑，他决心通过彻底调查这种致命疾病的根源来证实他的怀疑。

通过和当地居民交谈，他确定了霍乱爆发的源头是位于Broad大街的公共水泵。他对这种疾病类型的研究看起来很可信，因此他成功说服了当地政府废弃那个水泵。他所利用的主要证据就是图1.3：死亡发生的地点有明显的地理规律，在这种规律的指引和相关调查证据的支持下，他最终确定了霍乱的源头。后来证实离这口井仅三英尺远的地方有一处污水坑，坑内释放出来的细菌正是霍乱发生的罪魁祸首。

1.3 提灯女士的玫瑰图

南丁格尔（Florence Nightingale）是我们耳熟能详的“提灯女士”，她不仅是现代护理的鼻祖及现代护理专业的创始人，而且是历史上使用极坐标面积图的先驱。这种图形外形如玫瑰，因此后来也称之为玫瑰图，其主要构思是用“花瓣”的面积表示统计数值的大小。图1.4反映了克里米亚战争（英国等与俄国争夺巴尔干半岛的战争）中英国军队自1854年4月至1856年3月的逐月死亡人数(Nightingale 1858)；其中，右图为1854年4月至1855年3月的死亡人数，左图为1855年4月至1856年3月的死亡人数。玫瑰图不仅清楚展示了这两年军队死亡人数的变化，而且更重要的是，她将每个月中三种死亡情况也分别用不同颜色标记出来：蓝色表示死于可预防的疾病、红色表示死于战争伤害、黑色表示死于其它原因。这样我们可以清楚知道军队伤亡原因的结构，尤其是“绝大多数士兵死于可预防的疾病”（图中最高的花瓣）。凭借这一条重要信息，她让英国政府意识到，真正影响战争伤亡的并非战争本身，而是由于军队缺乏有效的医疗护理!

1.4 拿破仑的俄罗斯远征

1812年6月24日，拿破仑率领的691,501人的大兵团—同时也是欧洲历史上集结的最大规模的部队—开赴莫斯科。但等他们到达那里，看到的只是一座空城。城里的人都被遣散，所有的供给也被中断。由于没有正式的投降，拿破仑觉得俄国人从他那儿剥夺了一场传统意义上的胜利。

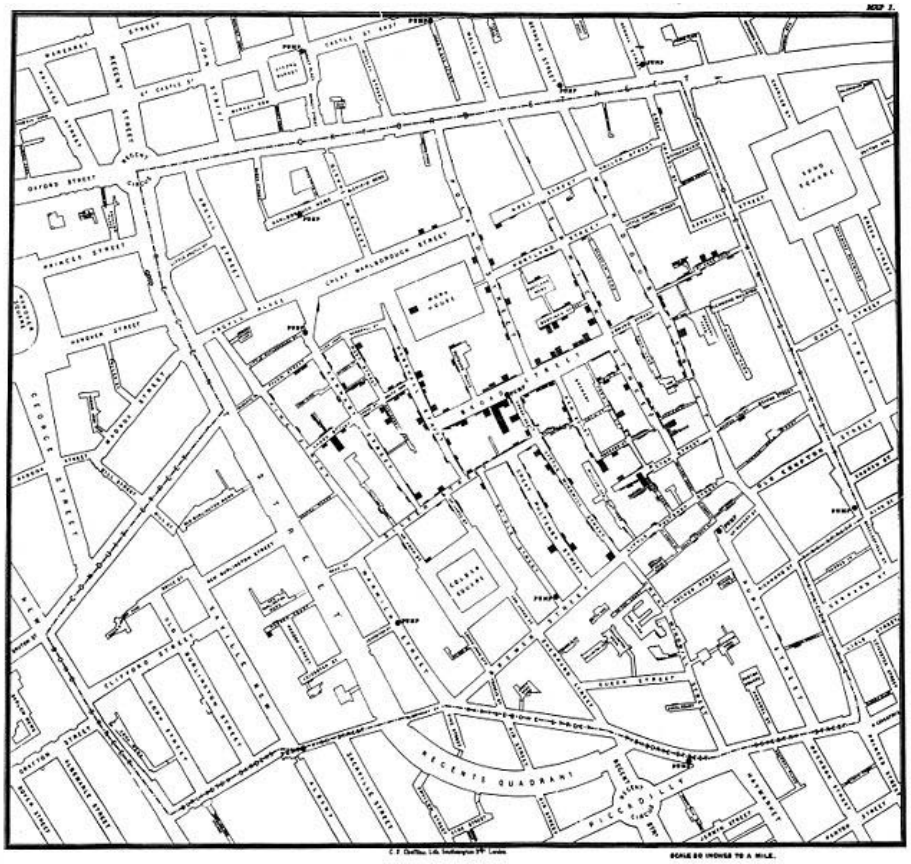


图 *1.3: 1854*年英国*Broad*大街大规模爆发霍乱，当时了解微生物理论的人很少，人们不清楚霍乱传播途径，而“瘴气传播理论”是当时的主导理论；*John Snow*对这种理论表示了怀疑，于*1849*年发表了关于霍乱传播理论的论文，本图即其主要依据。图中心东西方向的街道即为*Broad*大街，黑点表示死亡的地点。这幅图形揭示了一个重要现象，就是死亡发生地都在街道中部一处水源（水井）周围，市内其它水源周围极少发现死者。进一步调查他发现这些死者都饮用过这里的井水。图片来源：*http://upload.wikimedia.org/wikipedia/commons/2/27/Snow-cholera-map-1.jpg*

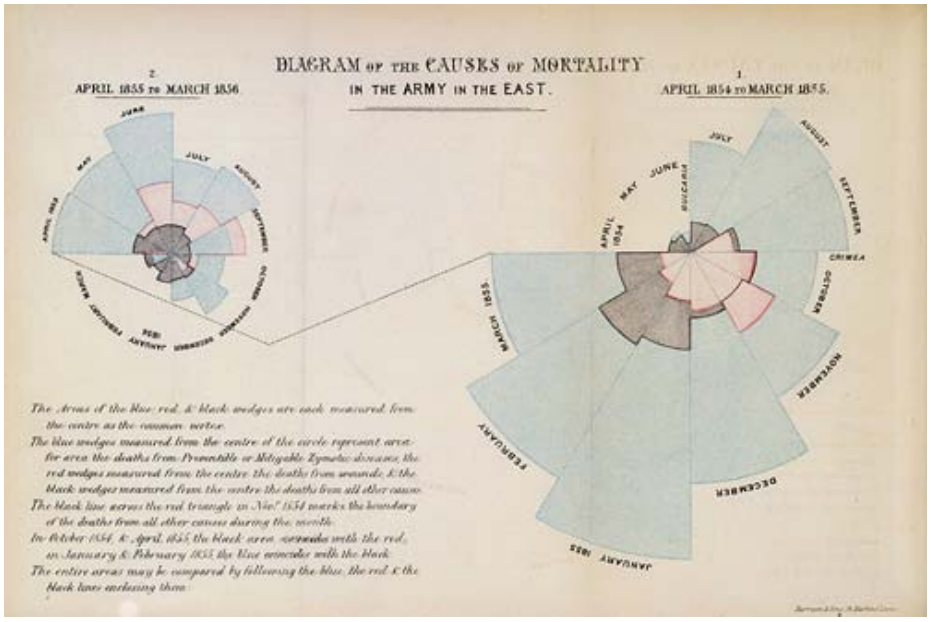


图 *1.4:* 南丁格尔的极坐标面积图：两幅图分别是*1854*年和*1855*年的军队伤亡人数，一年*12*个月恰好可以将极坐标分为*12*等分，每一瓣代表一个月。图中用颜色标记出了三种死亡原因。南丁格尔的重大贡献在于使得英国政府意识到真正影响战争伤亡的并非战争本身，而是由于军队缺乏有效的医疗护理，导致大量的士兵死于可预防的疾病。*1857*年，在她的努力下，英国皇家陆军卫生委员会成立。同年，军医学校成立。图片来源：*http://en.wikipedia.org/wiki/Florence_Nightingale*

军队不得不撤退。在归程中，给军队提供补给几乎是不可能的，主要是因为天气过于恶劣。马匹因为缺少粮草而变得虚弱，所有的马要么饿死，要么被饥饿的士兵拿去果腹。没有了坐骑，法国骑兵们成了步兵，大炮和马车被迫丢弃，部队没了装甲。饥饿与疾病带来惨重的伤亡，而逃兵增速也直线上升。大军团的小分队在Vyazma，Krasnoi和Polotsk也被俄国人击溃。法国军队在渡贝尔齐纳河时遭到俄军两面夹击，伤亡惨重，这也是法军在俄国遭遇的最后一场灾难。1812年12月14日，大军团被驱逐出俄国领土。在这场远征俄罗斯的战役中，拿破仑的士兵只有大约22,000人得以幸存。

这一历史事件被Charles Joseph Minard用一张二维平面图形记录了下来，Minard是一位法国工程师，他以在工程和统计中应用图形而闻名。图1.5就是他的著名作品：在一张二维图形中，他成功地展示了如下信息：

- 军队的位置和前进方向，以及一路上军队的分支和汇合情况
- 士兵数目的减少（图形顶端最粗的线条表示最初渡河的422,000人，他们一路深入到俄国领土，在莫斯科停下来的时候还有100,000人左右。从右到左，他们朝西走回头路，渡过Niemen河的时候，仅仅剩下10,000。随着大部队和余部会师（比如在渡贝尔齐纳河之前），图中显示的数字降中也有升）
- 撤退时的气温变化（参见图的下半部分，可知当时气候条件极其恶劣）

这幅图形在统计图形界内享有至高无上的地位，被Edward Tufte[1]称为“有史以来最好的统计图形”。

1.5 小结与开始

在前面四节中，我们看到了具有历史意义的四幅统计图形，它们融入了前人的智慧与艺术，有些甚至具有重大社会价值。当然我们不能苛求每一幅统计图形都能达到那样的效果，但至少我们了解到了统计图形在揭示特殊现象或规律上的功能，这种功能是数据本身不能替代的。试想，若只是将每一个霍乱死者的数据列在纸上，那么要观察出霍乱发生的规律是何其艰难。

[1]Tufte是统计图形和信息可视化领域的领军人物，人称“数据达芬奇”。

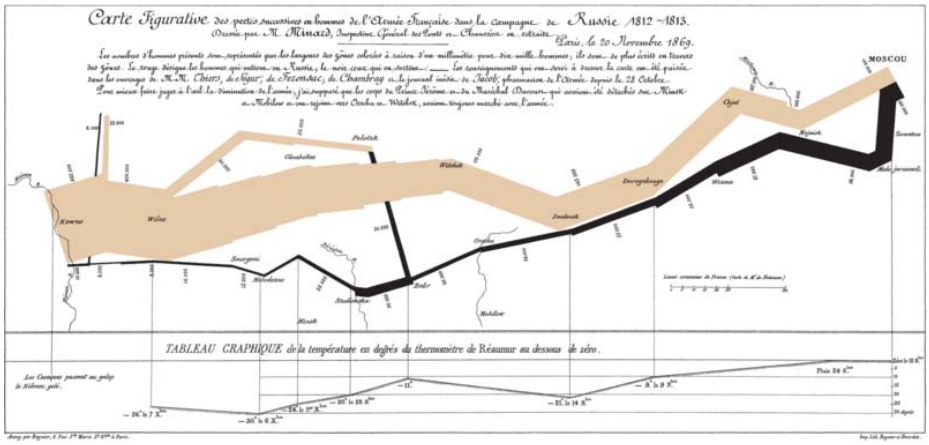


图 *1.5: Minard*绘制的地图，展现了*1812*年拿破仑的大军团进军俄国的路线（上半部分）和撤退时的气温变化（下半部分）。这一历史事件中，法军数量的急剧减少以及恶劣的气候条件一览无遗，法国科学家*Étienne-Jules Marey*称“该图所展现出的雄辩对历史学家的笔是一种极大的挑战”。图片来源：*http://upload.wikimedia.org/wikipedia/commons/2/29/Minard.png*

统计图形领域还有大批卓有成就的研究者，为统计图形的发展做出了不少贡献。早在上个世纪八九十年代，国外已经有比较全面的图示书籍文献资料，如前文提到的“数据达芬奇”(Tufte 1992, 2001)，Wainer and Thissen (1981)，Wilkinson (2005)，以及贝尔实验室的Cleveland (1985, 1993)等，其中尤其是Cleveland在数据可视化和统计图示方面撰写了大量的论文，还提出了不少原创图形类型，感兴趣的读者可以访问他在贝尔实验室的个人主页`http://cm.bell-labs.com/cm/ms/departments/sia/wsc/`。

关于统计图形的历史总结，Friendly (2008)是一份非常详尽的资料，该文档整理、记载了自17世纪以前至今数百年历史中较有影响力的统计图形。

如今统计图形的使用已经比较普遍，饼图、条形图都已经不是什么新鲜内容，但是一方面统计图形的价值并没有被很好地体现出来，另一方面人们对统计图形的了解和使用也被统计软件所限，而不能随心创造图形。我们来看这样一组事实(谢益辉 2008)：

> 以期刊《统计研究》在2006年12月~2007年11月期间共12个月的所有论文作为统计对象，剔除部分非学术研究型论文之后，挑选论文总数为168篇，其中使用表格的论文篇数为136篇（81.43%），表格总数为528个，而使用图形的论文仅有63篇（37.72%），若将仅仅使用示意图（非统计图形）、条形图和折线图的论文排除在外，使用其它图形的论文仅剩下9篇。

这可算国内统计图形应用现状的一个缩影。为了改变这种局面、发掘出统计图形在数据分析中应有的潜力，我们特别撰写这本小书，供广大统计研究者参考。我们的目的并非仅限于如何作出漂亮的统计图形，而是在作图的同时，强调图背后更重要的工作，那就是“数据分析与统计图形的有机结合”。传统的统计分析大约可以分为三类：

- 描述性统计分析：Descriptive Statistical Analysis
- 推断性统计分析：Inferential Statistical Analysis
- 探索性统计分析：Exploratory Statistical Analysis

前两类统计分析往往都是从既定的统计模型、方法的角度入手，而探索统计分析则主要借助图形对数据进行探索性分析，这对于数据分析的手段是一种重要的拓展（姑且称之为“图形统计分析”）；然而要使用这种手段，则必须清楚了解如何制图以及现有图形有哪些种类，这样才能真正开发出统计图形的价值。

其实，“图形统计分析”也不是一个新概念，平常的统计图示已经或多或少用到了这样的思想，只是我们往往更倾向于数理意义上的统计模型分析，而不会把图形统计分析作为主要分析手段，当然，由于图形的表达限制以及统计图形的普及程度，也使得它不可能替代模型分析，但无论如何，我们对统计图形在统计分析中的地位应该加深认识，不仅是因为这是一个信息爆炸的时代、大量的信息让我们无法在短时间内获取核心信息，更重要的是，目前在国内仍有大量的统计图形未被开发介绍出来，图形种类过于单一，表达信息的效果大打折扣。

本书介绍统计图形的方式主要是从两方面入手，第一，阐明各种统计图形所用到的统计量；第二，与实例结合，解释图形中表现的统计量的实际含义。在本书的附录B中，我们也会介绍一些有用的作图技巧，用以辅助完善统计图形。

总的说来，要把图形提到“统计分析”的高度，就一定要搞清楚统计图形的来龙去脉，包括原始数据的来源和类型、统计量的计算、图形的构造与组合机制等，这与统计模型实际上没有本质区别：若不清楚模型的假设前提、计算原理以及相应的结果解释，同样也不能随便使用模型分析。除了图形本身之外，用好图形分析还需要一定的观察力，最简单的莫过于观察数据的分布状况、离群点、线性/非线性关系等表面观察，而更重要也是最本质的的莫过于洞察到种种规律或异常现象背后的深刻原因，至此，我们才达到了分析的目的。

第二章　工具

> “好了，好了，我的好伙计，就这么办吧。我们在这房子里共同生活了好几年，如果再蹲在同一座牢房里就更有意思了。华生，我跟你说实话。我一直有个想法：我要是当罪犯，一定是超一流的。这是我在这方面难得的一次机会。看这儿!”他从抽屉里拿出一个整洁的皮制小袋，打开来亮出里面几件闪亮的工具。“这是最新最好的盗窃工具，镀镍的撬棒，镶着金刚石的玻璃刀，万能钥匙，以及对付现代文明所需要的各种新玩意儿。我这儿还有在黑暗中使用的灯。一切都准备好了。你有走路不出声的鞋吗?”
>
> —柯南·道尔《查尔斯·密尔沃顿》

当今统计软件有许多种，如SPSS、SAS、S-Plus、Statistica、Stata、Systat甚至Excel等等，它们都有作图功能，那么我们面对这么多工具应该如何选择呢？我们认为主要的准则有两点：一是统计计算功能齐全，二是统计元素易于控制。

2.1　选择作图工具

在人们通常的观念中，图形往往代表着简单，然而“直观”与“简单”是两个不同的概念，图形的首要作用的确是直观展示信息，然而这里的“信息”未必是简单的。一幅优秀的统计图形背后也许隐藏着重要的统计量，而统计量是统计图形的最关键构成因素。

我们通常见到的所谓“统计图形”也许只是Microsoft Office Excel的产物，事实上，Excel的图形总体看来只有三种，第一种是表现绝对数值大

小，如条形图、柱形图、折线图等，第二种是表现比例，如饼图，第三种则是表示二维平面上的变量关系，如X-Y散点图。而统计学的核心研究对象是什么？答案当然是分布（Distribution）[1]。无论Excel的图形如何搭配色彩、怎样变得立体化，都跳不出上面三种类型的限制，而且不能全面妥善表达统计学的要义。可能正是因为这样的原因，统计图形界内的一位大家Leland Wilkinson才说“给统计刊物投稿时永远不要用Excel作图”。本书所要介绍的R语言所能表达的统计量种类极其丰富，甚至可以毫不夸张地说，任何理论上可以计算出来的统计量都能在R中很方便地以图形的方式表达出来。

除了统计量之外，我们也应对图形本身的组成元素给予足够的重视，这些元素包括点、线（直线、曲线、线段和箭头等）、多边形、文本、图例、颜色等，往往这部分工作都由常见的统计软件替我们做了—我们不必自己设计点、线等基本元素，但同时也就意味着我们几乎无法灵活运用这些元素（比如在图中添加点、线、文本标注等）。表面上看似计算机软件给我们省了不少麻烦，实际上，这种限制的弊端要大于那一点微不足道的“好处”。我们在实际工作中遇到不少这样的例子，比如往散点图中添加若干条不同的回归直线（根据某自变量不同分类的回归、或者是不同分位数的分位回归(Koenker and Bassett 1978)直线等）、在图中添加箭头或者文本标注甚至添加包含希腊字母、微积分符号、上下标的数学公式，等等，不一而足，对于这些简单问题，用传统的（商业）统计软件恐怕太难解决，原因就在于，它们让计算机替代了太多本可以由用户完成的工作。

相比之下，R语言汇集统计计算与统计图示两种功能于一身，灵活的面向对象（Object-Oriented，OO）编程方式让我们可以很方便地控制图形输出，从而制作出既精美又专业的统计图形。我们说图形并不意味着简单，指的是统计图形的构造可以很细致入微（包括统计量的选定和图形元素的设计），而不是指R作图的过程或程序很复杂。

要想真正精通统计图形，则应首先要练好基本功（例如对图形基础元素或构造作深入的了解），然后在此基础上通过各种抽象、提炼和认识最终上升到理性认识的境界（自如地表达统计量）。

[1]分布不仅包含一元变量的分布，而且（更重要的是）包括多元变量的分布，诸如“相关”、“概率”等概念都可以归为“分布”的范畴

2.2 R语言简介

“R”(R Development Core Team 2009)是一款优秀的统计软件，同时也是一门统计计算与作图的语言，它最初由奥克兰大学统计学系的Ross Ihaka和Robert Gentleman编写(Ihaka and Gentleman 1996)；自1997年R开始由一个核心团队（R Core Team）开发，这个团队的成员大部分来自大学机构（统计及相关院系），包括牛津大学、华盛顿大学、威斯康星大学、爱荷华大学、奥克兰大学等，除了这些作者之外，R还拥有一大批贡献者（来自哈佛大学、加州大学洛杉矶分校、麻省理工大学等），他们为R编写代码、修正程序缺陷和撰写文档。迄今为止，R中的程序包（Package）已经是数以千计，各种统计前沿理论方法的相应计算机程序都会在短时间内以软件包的形式得以实现，这种速度是其它统计软件无法比拟的。除此之外，R还有一个重要的特点，那就是它是免费、开源的！由于R背后的强大技术支持力量和它在统计理论及应用上的优势，加上目前国内学术和应用界内人士对R的了解相对较少，我们期望能够通过本书引进并推动它在国内的广泛使用。

R的功能概括起来可以分为两方面，一是统计计算（Statistical Computation），二是统计图示（Graphics）；一般而言，图形往往比较直观而且相对简易，因此本书从R图形入手，以期能给初学者提供一份好的R作图入门学习资料。

在启动R之后[2]，屏幕上会显示类似如下信息，告诉我们R是由很多贡献者一起协作编写的免费软件，用户可以在一定许可和条件（GPL）下重新发布它：

```
> cat(head(system("R -e R.version.string", TRUE)[-1],
+     15), sep = "\n")

R version 2.10.1 (2009-12-14)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
```

[2]Windows用户一般从程序快捷方式直接启动，Linux用户一般可以从终端敲入命令R启动

```
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

从技术上来讲，R是一套用于统计计算和图示的综合系统，它由一个语言系统（R语言）和运行环境构成，后者包括图形、调试器（Debugger）、对某些系统函数的调用和运行脚本文件的能力。R的设计原型是基于两种已有的语言：S语言[3](Becker *et al.* 1988)以及Sussman的Scheme[4]，因此它在外观上很像S，而背后的执行方式和语义是来自Scheme。

R的核心是一种解释性计算机语言，大部分用户可见的函数都是用R语言编写的，而用户也可以调用C、C++或者FORTRAN程序以提高运算效率。正式发行的R版本中默认包括了**base**（R基础包）、**stats**（统计函数包）、**graphics**（图形包）、**grDevices**（图形设备包）、**datasets**（数据集包）等基础程序包，其中包含了大量的统计模型函数，如：线性模型/广义线性模型、非线性回归模型、时间序列分析、经典的参数/非参数检验、聚类和光滑方法等，还有大批灵活的作图程序。此外，附加程序包（add-on packages）中也提供了各式各样的程序用于特殊的统计学方法，但这些附加包都必须先安装到R的系统中才能够使用(Hornik 2009)。

本书不会过多涉及到附加包，所介绍图形主要基于R自身的**graphics**包，当然也不可避免会使用**base**和**grDevices**等基础包中的函数[5]；在第五章中会使用一些附加包介绍特殊的统计数据和统计方法、模型涉及到的图形，例如分类数据（Categorical Data）会用到**vcd**包，生存分析会用到**survival**包。当我们需要调用附加包时，可以使用*library()*函数，例如加载**MSG**包：

```
1 > library(MSG)
```

R的官方网站`http://www.R-project.org`中对R有详细介绍，我们也可以从它在世界各地的镜像（CRAN：`http://CRAN.R-project.org`，全称**C**omprehensive **R** **A**rchive **N**etwork）下载R的安装程序和附加包，通常

[3] `http://cm.bell-labs.com/cm/ms/departments/sia/S/history.html`

[4] `http://www.cs.indiana.edu/scheme-repository/home.html`

[5] 这些基础包一般不用特别加载，R在启动的时候会自动加载进来，我们随时可以用*search()*函数来查看目前的工作环境中有哪些包已经被加载

我们可以用函数*install.packages()*安装附加包，Windows用户也可以从菜单中点击安装：Packages ⇒ Install Package(s))，注意，附加包都是从镜像上下载的，因此安装时要保证网络连接正常；当然我们也可以先将程序包下载到本地计算机上然后安装。

可能令读者感到不便的一点是，R不像别的统计软件那样有图形用户界面（GUI，即**G**raphical **U**ser **I**nterface）[6]，因此它不会显得很“傻瓜”，它的界面常常是命令行界面（CLI，即**C**ommand **L**ine **I**nterface），即：输入一些代码，R就会输出相应的运算结果或其它输出结果。因此，在正式使用R之前，我们有必要大致了解一下R的运作方式。

著名计算机科学家Nikiklaus Wirth曾经提出，程序语言的经典构成是“数据结构+算法”：数据结构是程序要处理的对象，算法则是程序的灵魂。R也不例外，它有自己独特的数据结构，这些数据结构尤其适应统计分析的需要，它们包括：向量（vector）、矩阵（matrix）、数据框（data frame）、列表（list）、数组（array）、因子（factor）和时间序列（ts）等，当然，数值、文本和逻辑数据都可以在R中灵活使用，附录A.1中给出了一些简单的数据操作例子，请读者通过阅读该章熟悉R的数据对象；至于算法，我们暂时可以不去过多了解，因为R中已经包含了大量设计好的函数，对于一般的用户来讲都不必自行设计算法，除非有特殊或者自定义的算法，那么也可以根据R的语法规则编写程序代码。

对R的运作方式粗略了解之后，我们再回头看看GUI。一个软件的菜单、按钮、对话框等GUI组件不可能无限增多，否则软件会变得无比庞大臃肿，而繁杂的操作顺序的难度将很可能会超过使用程序命令行的难度，而且非开源的GUI隐藏了计算原理，除了程序的原始编写者，无人知道输出结果究竟是以怎样的方式计算出来。随着统计学的发展，各种新方法、模型必将不断涌现，我们现在不妨试想未来的统计软件用户界面将会变成什么样子。看看R的发展，可以体会到这种思路：菜单不可能无限增加，但是程序、函数都是可以无限增加的—道理很简单，因为它们不受计算机屏幕的限制。迄今为止，R的仓库（repository）中附加包的数量已经超过2000个，这还只是CRAN上的仓库，不算另外两个著名站点“Bioconductor”和“Omegahat”。在这样的大仓库中，我们可以找到最前沿的统计理论方法的实现，所需做的仅仅就是下载一个通常在几十K到几

[6]事实上也不是完全没有，比如John Fox的**Rcmdr**(Fox *et al.* 2009)包就是一个比较成熟和著名的R用户界面，我们可以在其中点菜单操作，但我们并不推荐用户过多依赖于GUI，因为GUI会让我们难以驾驭很多统计分析的细节处理，况且R语法命令也不是那样难学

百K的一个附加包。值得注意的是，R的主安装程序大小约为30M，相比之下，SPSS的安装程序已达六七百M，而SAS的安装程序早已达数G（六七张光盘），从程序大小角度便可知R语言的精炼。

关于R更深入的介绍，请参考官方发行的若干手册和网站上的大量学习材料，如官方的六本手册（均可从R的安装目录或者网站`http://cran.r-project.org/manuals.html`上找到）：

- An Introduction to R (R-intro)
- R Data Import/Export (R-data)
- R Installation and Administration (R-admin)
- Writing R Extensions (R-exts)
- R Internals (R-ints)
- The R Language Definition (R-lang)

其中R-intro手册附录A中给了一个很好的代码入门演示，推荐读者通过这个演示初步熟悉R语言；其它手册都比较偏重R的底层介绍，不适合对计算机程序没有深入了解的初学者阅读；另外还有Emmanuel Paradis编写的“R for Beginners”也是较好的入门教材[7]。鉴于目前R的中文资料并不多，我们也推荐R的初学者和爱好者通过访问“统计之都”中文网站（`http://cos.name`；R版块`http://cos.name/cn/forum/15`）共同学习、讨论和研究。

最后需要特别指出的是，R拥有一套完善而便利的帮助系统，这对于初学者也是很好的资源。若已知函数名称，我们可以简单用问号“`?`”来获取该函数的帮助，比如查询计算均值的函数帮助，只需要在命令行中敲入“`?mean`”，则会弹出一个窗口显示该函数的详细说明。大多数情况下“`?`”等价于*help()*函数；但有少数特殊的函数在查询其帮助时需要在函数名上加引号，比如“`?+`”就查不到加法的帮助信息，而“`?'+'`”或者“`help('+')`”则可顺利查到加法帮助信息，类似的还有`if`、`for`等。

一般来说，帮助窗口会显示一个函数所属的包、用途、用法、参数说明、返回值、参考文献、相关函数以及示例，这些信息是相当丰富的。当

[7]丁国徽已将几本官方手册翻译为中文，R-intro的中文翻译文档可以在官方网站上下载，其它中文手册可从`http://www.biosino.org/R/R-doc/`得到；R for Beginners也已由本书作者及其他几位合作者翻译为中文，可从“统计之都”获得：`http://cos.name`

然，更多情况下是我们并不知道函数名称是什么，此时也可以使用搜索功能，即函数*help.search()*（几乎等价于双问号“??”），例如我们想知道方差分析的函数名称，则可输入命令help.search('analysis of variance')，弹出的信息窗口会显示与搜索关键词相关的所有函数，如*aov()*等。

知道名称以后，接下来我们就可以通过前面讲到的“?”来查询具体函数的帮助信息。函数*help.start()*可以打开一个网页浏览器用来浏览帮助。如果这些帮助功能还不够用，例如有时候需要的函数在已经安装的包中找不到，那么可以到官方网站上搜索：http://www.r-project.org/search.html，网站上的邮件列表导航（Mailing List Archives）也是很有用的资源，其中有大批统计相关学科的著名教授以及世界各地的统计研究者和R爱好者在那里用邮件的方式公开回答各种关于R的问题。

我们在此如此强调帮助系统，目的在于告诉读者，要想学好R语言，除了阅读相关书籍资料，也应该自己多多动手利用信息资源解决自己的问题。

2.3 安装R语言

读者在进入下一章学习之前，可以先将R安装在自己的计算机上，以便阅读时可以随时打开R进行实际操作。R软件可以在多种操作系统上运行，包括Windows、Linux以及Mac OS X等，进入官方网站主页会发现左栏有“Download”一项（CRAN），点击进入便可以看到世界各地的R镜像，任意选择一个进入（比如选择美国加州大学伯克利分校http://cran.cnr.Berkeley.edu），我们就会看到R安装程序和源代码的下载页面，此时只需要根据自己的操作系统选择相应的链接进入下载即可，例如Windows用户应该选择“Windows (95 and later)”进入，然后下载基础安装包（base），其中R-*.*.*-win32.exe字样[8]的链接便是Windows安装程序；Linux用户则可根据具体的系统如RedHat、Ubuntu等选择对应的链接。

由于R是完全开放源代码的，所以我们可以自由修改代码以构建符合自己需要的程序，但是一方面这需要一定的其它程序语言技能（典型的如C语言），因为R的很多基础函数都是用C写的，另一方面，对于绝大多数用户，

[8] *.*.*表示版本号，例如2.3.1；本书的程序版本是2009年12月发布Windows 2.10.1版，从2000年2月至今，R已经更新了30多个版本，更新速度非常快

其实没有必要对基础包进行修改—既然是开源软件，其代码必然受到很多用户“监视”，这样就会最大程度减少程序错误、优化程序代码，而且我们也可以在R里面自定义函数，这些函数可以保存起来，以后同样还能继续使用。对比起来，现今的商业统计软件都将源代码和计算过程封闭在用户完全不知道的“黑匣子”中，而在用户界面上花大量的功夫，这对统计来说，毫无疑问并非长久之计。我们相信，随着读者对R的深入了解，一定能体会到这个软件的真正强大之处。

下面我们以一个简单的例子开始R图形之旅，见图2.1。

```
> # 生成x（从-3到3长度为200的数列）和y（正态密度）
> x = seq(-3, 3, length = 200)
> y = dnorm(x)
> # 建立图形框架
> plot(x, y, type = "n")
> # 设定5%和95%分位数之间的X点和相应密度值
> xx = seq(-1.65, 1.65, length = 100)
> yy = c(0, dnorm(xx), 0)
> xx = c(-1.65, xx, 1.65)
> # 灰色多边形
> polygon(xx, yy, col = "gray", border = NA)
> # 添加密度曲线
> lines(x, y)
> # 文本标注
> text(0, 0.05, "$P(-1.65 < X < 1.65) = 90\\%$")
```

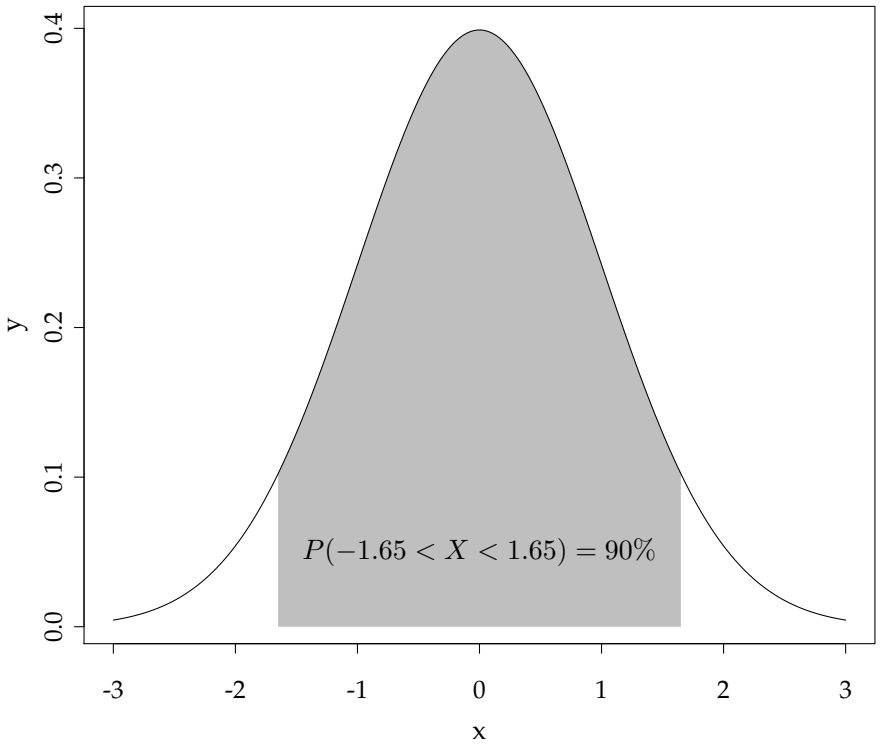


图 2.1: 正态分布密度曲线及其5%到95%分位数区间表示

第三章　细节

“那么，关于绳子的话题就谈到这里吧。”福尔摩斯微笑着说，“现在，我们来看看包裹纸吧。牛皮纸，带有一股明显的咖啡味。怎么，你还没有观察到？我想，肯定没有检查过。地址的字写得很零乱：‘克罗伊登十字大街*S*.库欣小姐收’，是用笔尖很粗的钢笔写的，或许是一支*J*字牌的，墨水很差。‘克罗伊登’一词原来是拼写的字母‘*i*’，后来才被改成字母‘*y*’的。而且，这个包裹是一个男人直接寄的—字体很明显是男人的字体—这个男人文化程度不高，对克罗伊登镇也不怎么熟悉。到目前为止，一切都顺利！这个纸盒子是一个半磅装甘露烟草盒。除了盒子右下角有两个指印外，再没有明显的痕迹。里面装的粗盐是用来保存兽皮和其他粗制商品的那种。埋在盐里的就是这奇怪的东西。”

—柯南·道尔《硬纸盒子》

统计图形都是通过相应的图形函数生成的，R默认的图形设置一般已经比较符合美学原理，但是往往由于其它方面的需要（如排版、强调某一部分），我们可能要对图形作一些细节性的微调，比如字体、字号、图形边距、点线样式等等。

R的图形参数可以通过函数*par()*预先全局设置，也可以在具体作图函数（如*plot()*、*lines()*等）中设置临时参数值；二者的区别在于前者的设置会一直起作用，除非将图形设备（参见附录B.5）关闭，而后者的设置只是临时性的，不会影响后面其它作图函数的图形效果。函数*par()*中涵盖了大部分图形参数，因此专用一节讲述。

3.1 *par()*函数的参数详解

函数*par()*可以用来设置或者获取图形参数，*par()*本身（括号中不写任何参数）返回当前的图形参数设置（一个list）；若要设置图形参数，则可用`par(tag = value)`的形式，其中tag的详细说明参见下面的列表，value就是参数值。

目前*par()*函数涉及到的图形参数大约有70个，这里只是选取其中40多个常用且较易理解的参数进行解释说明如下列表，其它参数请参阅R帮助`?par`。

adj 调整图中字符的相对位置；取值：长度为2的数值向量，分别表示字符边界矩形框的左下角相对坐标点`(x, y)`位置的调整，向量的两个数值一般都在$[0,1]$范围中（有些图形设备中也可以超出此范围），表示字符串以左下角为基准、根据自身的宽度和高度分别向左和向下移动的比例，默认为`c(0.5, 0.5)`。例如`c(0, 0)`表示整个字符（串）的左下角对准设定的坐标点，而`c(1, 0)`则表示字符串横向移动了自身宽度的距离，而纵向不受影响。具体示例参见图3.1左上图

ask 切换到下一个新的作图设备（通常是作一幅新图）时是否需要用户输入（敲回车键或点鼠标）；`TRUE`表示是；`FALSE`表示否。当有多幅图将逐一出现而需要按顺序一步步在图形设备上展示时很有用，这种情况下若设置ask为`TRUE`，那么作图时每一副新图的出现都要先等待用户输入，否则所有的图将会一闪而过

bg 设置图形背景色；关于颜色值的设置请参见4.1节

bty 设置图形边框样式；取值为字符`o`，`l`，`7`，`c`，`u`，`]`之一；这些字符本身的形状对应着边框样式，比如（默认值）`o`表示四条边都显示，而`c`表示不显示右侧边，参见图3.2四幅图的边框样式

cex 图上元素（文本和符号等）的缩放倍数；取值为一个相对于1的数值（默认为1）。具体的细节缩放可以通过如下参数设置（默认值均为1）：

cex.axis 坐标轴刻度标记的缩放倍数

cex.lab 坐标轴标题的缩放倍数

cex.main 图主标题的缩放倍数

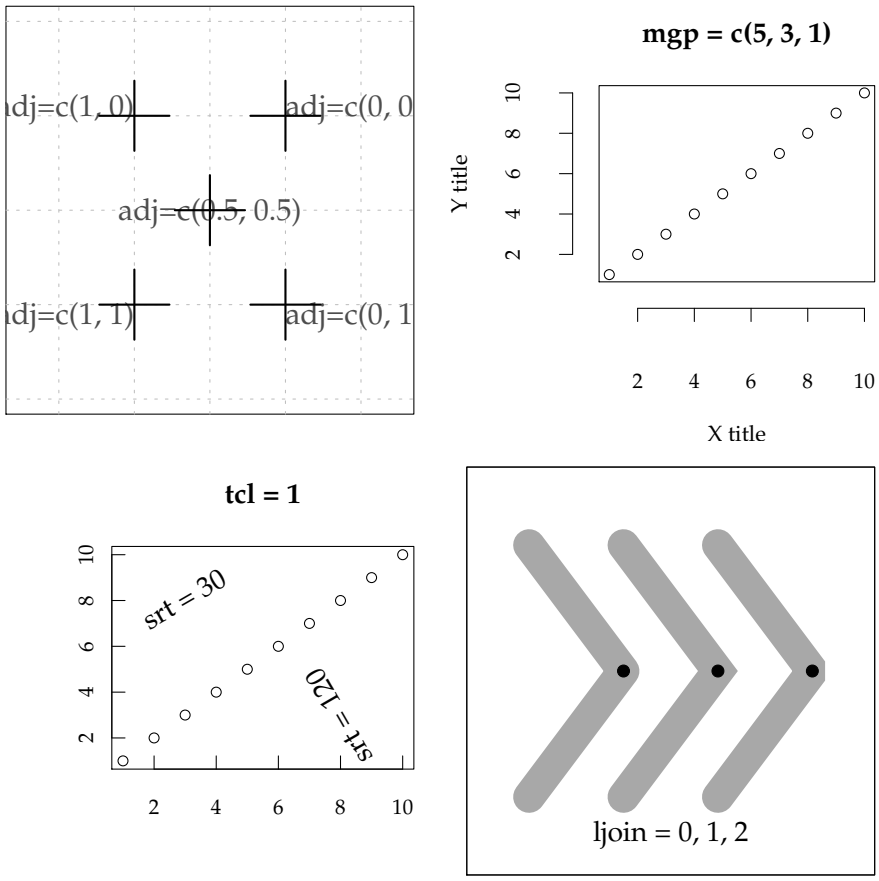


图 3.1: 左上：*adj*用于字符相对位置的调整，“+”表示真实坐标点的位置，通过一个长度为2的向量`c(x, y)`可以分别调整字符在横纵坐标上相对平移的位置；右上：坐标轴元素的边界距离，参数*mgp*的三个数值分别控制了坐标轴标题、坐标轴刻度数字以及坐标轴线到图形的距离；左下：*tcl*控制了坐标轴刻度线的方向和长度，*srt*控制了字符串的旋转角度；右下：线条相交处的样式。

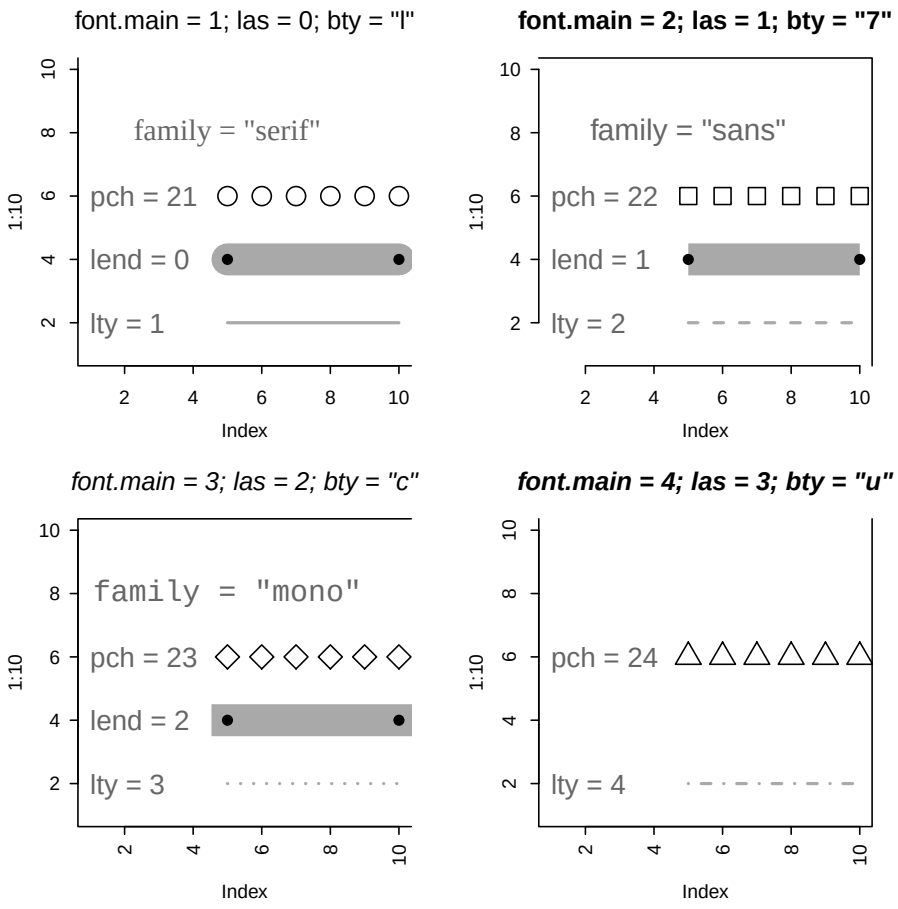


图 3.2: 四幅图形演示了不同的图形边框（上右开、上右闭、右开和上开）、字体样式（正常、粗体、斜体和粗斜体）、字体族（衬线、无衬线、等宽、符号）、坐标轴标签样式、点样式（圆圈、方框、菱形、三角）、线末端样式和线样式（实线、虚线、点线、点划线）。

cex.sub 图副标题的缩放倍数

col 图中符号（点、线等）的颜色；取值参见4.1节。与cex参数类似，具体的细节颜色也可以通过如下参数设置：

col.axis 坐标轴刻度标记的颜色

col.lab 坐标轴标题的颜色

col.main 图主标题的颜色

col.sub 图副标题的颜色

family 设置文本的字体族（衬线、无衬线、等宽、符号字体等）；标准取值有：`serif`，`sans`，`mono`，`symbol`，参见图3.2坐标`(2, 8)`处的文本；`family = 'symbol'`的情况没有显示出来

fg 设置前景色（若后面没有指定别的颜色设置，本参数会影响几乎所有的后续图形元素颜色，若后续图形元素有指定的颜色设置，那么只是影响图形边框和坐标轴刻度线的颜色）；颜色值参见4.1节。

font 设置文本字体样式；取值为一个整数；通常1、2、3、4分别表示正常、粗体、斜体和粗斜体[1]，4.6节有进一步的介绍，参见图3.2的图主标题字体

font.axis 坐标轴刻度标签的字体样式

font.lab 坐标轴标题的字体样式

font.main 图主标题的字体样式

font.sub 图副标题的字体样式

lab 设置坐标轴刻度数目（R会尽量自动“取整”[2]）；取值形式`c(x, y, len)`：`x`和`y`分别设置两轴的刻度数目，`len`目前在R中尚未生效，因此设置任意值都不会有影响（但用到lab参数时必须写上这个参数）

las 坐标轴标签样式；取0、1、2、3四个整数之一，分别表示“总是平行于坐标轴”、“总是水平”、“总是垂直于坐标轴”和“总是竖直”。仔细观察图3.2中四幅图的不同坐标轴标签方向

[1]对于添加文本，*text()*函数及其vfont参数可以设置更为详细的字体族和字体样式；参见这两个演示：`demo(Hershey)`和`demo(Japanese)`，前者演示Hershey向量字体，后者演示日语的表示。

[2]尽量向0.5、1或10的幂次靠近

lend 线条末端的样式（圆或方形）；取值为整数0、1、2之一（或相应的字符串'round', 'mitre', 'bevel'），注意后两者的细微区别[3]

lheight 图中文本行高；取值为一个倍数，默认为1

ljoin 线条相交处的样式；取值为整数0、1、2之一（或相应的字符串'round', 'mitre', 'bevel'），分别表示画圆角、画方角和切掉顶角，观察图3.1的三个直角的顶点

lty 线条样式：0 ⇒不画线，1 ⇒实线，2 ⇒虚线，3 ⇒点线，4 ⇒点划线，5 ⇒长划线，6 ⇒点长划线；或者相应设置如下字符：'blank', 'solid', 'dashed', 'dotted', 'dotdash', 'longdash', 'twodash'；还可以用由十六进制的数字组成的字符串表示线上实线和空白的相应长度，如'F624'，详细解释请参见4.3一节。

lwd 线条宽度；默认为1

mar 设置图形边界空白宽度；按照“下、左、上、右”的顺序，默认为c(5, 4, 4, 2) + 0.1

mex 设置坐标轴的边界宽度缩放倍数；默认为1，本参数会影响到mgp参数

mfrow, mfcol 设置一页多图；取值形式c(nrow, ncol)长度为2的向量，分别设置行数和列数，参见附录B.2

mgp 设置坐标轴的边界宽度；取值长度为3的数值向量，分别表示坐标轴标题、坐标轴刻度线标签和坐标轴线的边界宽度（受mex的影响），默认为c(3, 1, 0)，意思是坐标轴标题、坐标轴刻度线标签和坐标轴线离作图区域的距离分别为3、1、0；参见图3.1右上方小图

oma 设置外边界（Outer Margin）宽度；类似mar，默认为c(0, 0, 0, 0)，当一页上只放一张图时，该参数与mar不好区分，但在一页多图的情况下就容易可以看出与mar的区别

pch 点的符号；pch = 19⇒实圆点、pch = 20⇒小实圆点、pch = 21⇒圆圈、pch = 22⇒正方形、pch = 23⇒菱形、pch = 24⇒正三角尖、pch = 25⇒倒三角尖，其中，21-25可以填充颜色（用bg参数），参见图4.1

[3]仔细观察图3.2中宽线条中黑点的位置，在画线时，这些线条的起点和终点（分别用图中的两个黑点表示）都是选择同样的坐标位置!

pty 设置作图区域的形状；默认为'm'：尽可能最大化作图区域；另外一种取值's'表示设置作图区域为正方形

srt 字符串的旋转角度；取一个角度数值，参见图3.1左下方小图中分别旋转30°和120°的字符串

tck 坐标轴刻度线的高度；取值为与图形宽高的比例值（0到1之间）；正值表示向内画刻度线，负值表示向外；默认为不使用它（设为NA}），而使用tcl参数

tcl 坐标轴刻度线的高度；取一个与文本行高的比例值；正负值意义类似tck，默认值为-0.5，即向外画线，高度为半行文本高；观察图3.1左下角小图的坐标轴刻度线

usr 作图区域的范围限制，取值长度为4的数值向量c(x1, x2, y1, y2)，分别表示作图区域内x轴的左右极限和y轴的下上极限；注意，若坐标取了对数（参见xlog, ylog两个参数），那么实际上设置的极限都是10的相应幂次

xaxs, yaxs 坐标轴范围的计算方式；默认'r'：先把原始数据的范围向外扩大4%，然后用这个范围画坐标轴；另外一种取值'i'表示直接使用原始数据范围；实际上还有其它的坐标轴范围计算方式，但是鉴于它们目前在R中都尚未生效，所以暂不加介绍

xaxt, yaxt 坐标轴样式；默认's'为标准样式；另外一种取值'n'意思是不画坐标轴

xlog, ylog 坐标是否取对数；默认FALSE

xpd 对超出边界的图形的处理方式；取值FALSE：把图形限制在作图区域内，出界的图形截去；取值TRUE：把图形限制在图形区域内，出界的图形截去；取值NA：把图形限制在设备区域内。这些区域的说明参见下文和图3.3

整个作图设备实际上可以分为三个区域，分别是："作图区域（Plot Region）"、"图形区域（Figure Margin）"和"设备区域（Device Region）"，这三个区域也对应着两种边界："图形边界（Figure Margin）"和"外边界（Outer Margin）"，这些概念对于初学者来说可能会感到迷惑，然而图3.3是

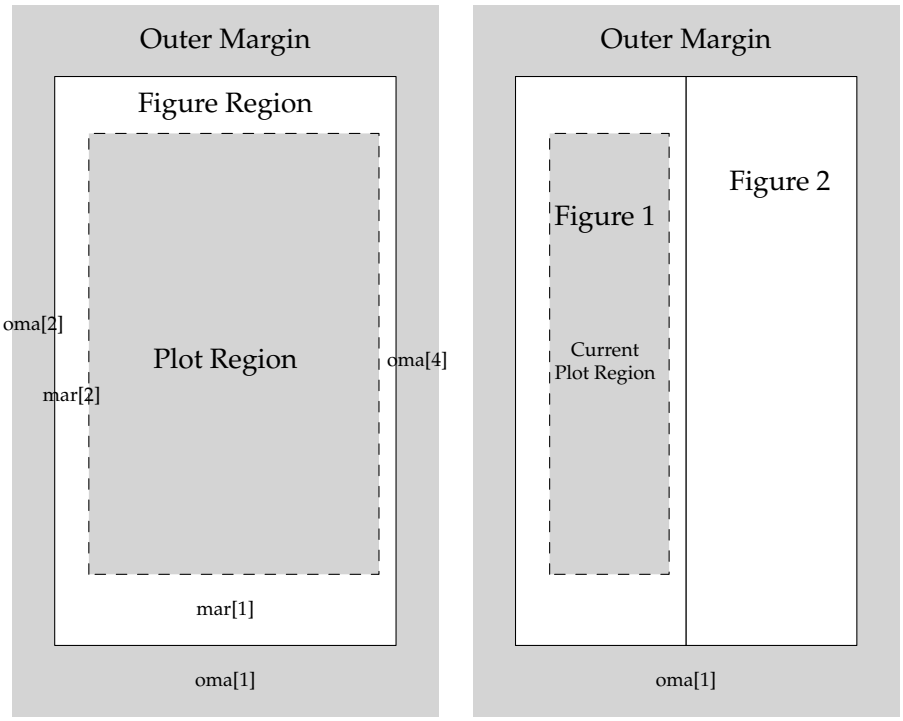


图 *3.3:* 图形的各种区域和边界说明：作图区域（当前作图区域）、图形区域和设备区域；图形边界和外边界。

一个很好的说明。R中的图形都是作在一个图形设备中，最常见的图形设备就是一个图形窗口，也可以在其它设备中（参见附录B.5）；整个设备内的区域就称为设备区域，也就是图3.3中最大的灰色区域，图形区域是设备区域内的白色实框方形区域，最里面的灰色虚框区域就是作图区域，我们的图形实体部分就作在这个区域；从设备区域的边界向内，到图形区域之间这一段称为外边界（用oma参数设定），图形区域边界再向内到作图区域的边界称为图形边界（用mar参数设定）。图3.3的左图是一页一图的展示，右图则是一页多图的展示，该展示更清楚地说明了oma与mar参数的区别，因为一页一图的情况下，外边界和图形边界完全融合在一起，很难分辨。

下面列表中的九组参数只能通过*par()*函数调用[4]，而在其它作图函数中不可设置（否则会导致错误或者被忽略）：

- ask
- fig, fin
- lheight
- mai, mar, mex, mfcol, mfrow, mfg
- new
- oma, omd, omi
- pin, plt, ps, pty
- usr
- xlog, ylog

介绍完上面的参数之后，我们顺便提一下关于*par()*的常用技巧。本节开头提到过，这个函数会“永久性”改变作图设置，我们有时并不想要这种功能，特别是在一幅图作完之后到准备下一幅图时，我们可能希望之前的参数可以被“还原”回来，此时，我们就需要在一幅图开始之前先把作图参数保存到一个对象中，比如`op = par()`，然后我们可以在作这幅图的过程中用*par()*函数任意更改设置以适合需要，作完这一幅图之后，我们再

[4]与此对应的是，有些参数同样可以在别的作图函数中调用，如las参数：`plot(..., las = 1)`；但要提醒读者注意，在其它函数中即使调用与*par()*相同的参数，也可能会有不同效果，典型的如col、pch等参数，请注意查看相应函数帮助

用par(op)语句把之前保存的参数设置“释放”出来，这样，中间过程对图形参数的更改就不再会影响到下一幅图。当然，也可以每作完一幅图都把图形设备关掉，然后再作下一幅图，这样也能达到目的，只是稍显麻烦而已，尤其是有时候对一幅图形反复重作、调整、比较，那时不断关闭、打开图形设备就显得更繁琐了。

3.2 *plot()*及相关函数的参数说明

R中最普通的作图函数就是*plot()*函数，它是一个泛型函数（R中的泛型函数与Java、C++等语言可能有所不同，附录A.1中会对它进行介绍），可以接受很多不同类的对象作为它的作图对象参数；我们这里要解释的只是其中的图形参数，而非作图对象参数。

先介绍*plot()*的通用参数：

type 图形样式类型，有九种可能的取值，分别代表不同的样式：'p'⇒画点；'l'⇒画线[5]；'b'⇒同时画点和线，但点线不相交；'c'⇒ 将type = 'b'中的点去掉，只剩下相应的线条部分；'o'⇒同时画点和线，且相互重叠，这是它与type = 'b'的区别；'h'⇒画铅垂线；'s'⇒ 画阶梯线，从一点到下一点时，先画水平线，再画垂直线；'S'⇒也是画阶梯线，但从一点到下一点是先画垂直线，再画水平线；'n'⇒作一幅空图，没有任何内容，但坐标轴、标题等其它元素都照样显示（除非用别的设置特意隐藏了）。图3.4的九幅图清楚说明了这九种类型

main 主标题；也可以在作图之后用数*title()*添加上，参见4.6节

sub 副标题；同上

xlab x轴标题；同上

ylab y轴标题；同上

asp 图形纵横比y/x；通常情况下这个比率不是1，有些情况下需要设置以显示更好的图形效果，例如需要从角度表现直线的斜率：若asp不等于1，那么45° 的角可能看起来并不像真实的45°

[5]注意是字母l（表示“line”），不是数字1！同样后面的字母o（表示“overplotted”）也不要误认为是数字0!

```
> par(mfrow = c(3, 3), mar = c(2, 2.5, 3, 2))
> for (i in c("p", "l", "b", "c", "o", "h", "s", "S",
+     "n")) {
+     plot(c(1:5, 5:1), type = i, main = paste("Plot type: \"",
+         i, "\"", sep = ""), xlab = "")
+ }
```

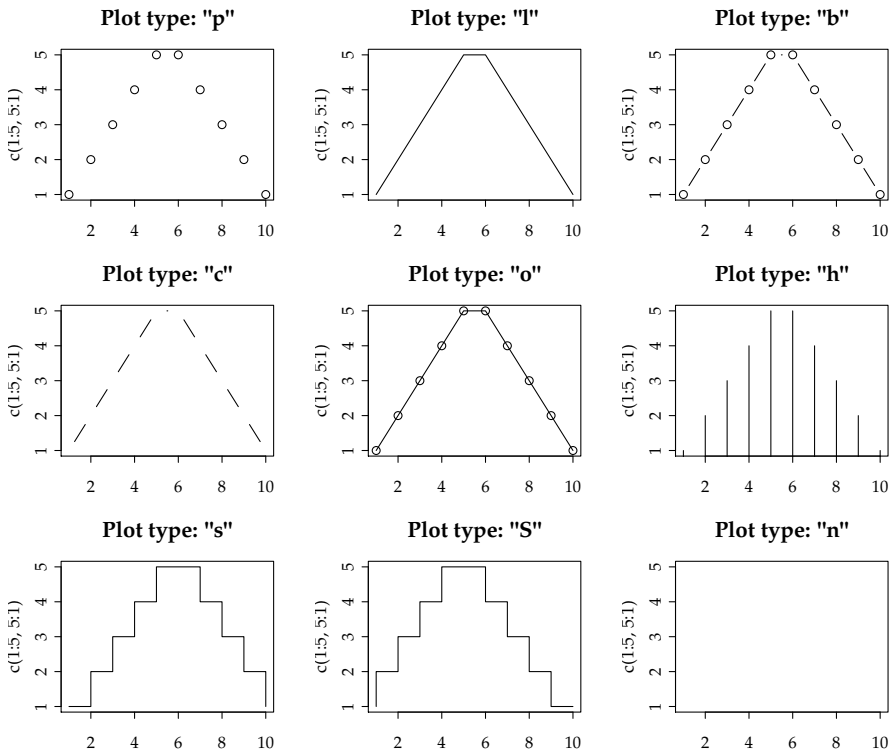


图 3.4: plot()作图的九种样式类型：点、线、点线（不相接）、擦掉点的线、点线（相接）、垂线、阶梯（水平起步）、阶梯（垂直起步）、无。

然后我们看看默认的散点图函数*plot.default()*。对于一般的散点图（两个数值变量之间），我们只需要调用*plot()*即可，如`plot(x, y)`，而不必写明`plot.default(x, y)`，原因就是*plot()*是泛型函数，它会自动判断传给它的数据类型从而采取不同的作图方式。*plot.default()*的参数当然包含了前面介绍的*plot()*中那些参数，此外还有：

x, y 欲作散点图的两个向量；如果y缺失，那么就用x对它的元素位置（`1:n`的整数）作散点图

xlim, ylim 设置坐标系的界限，两个参数都取长度为2的向量，它们的作用类似*par()*中的usr参数[6]

log 坐标是否取对数，`TRUE`或者`FALSE`

ann 一些默认的标记是否显示，如坐标轴标题和图标题

axes 是否画坐标轴；注意只会影响到是否画出坐标轴线和刻度，不会影响坐标轴标题

frame.plot 是否给图形加框；可以查阅*box()*函数，作用类似但功能更详细

panel.first 在作图前要完成的工作；这个参数常常被用来在作图之前添加背景网格（参见4.5节）或者添加散点的平滑曲线，比如`panel.first = grid()`

panel.last 作图之后要完成的工作；与上一个参数类似

... 其它常用参数如下：

col, pch, cex, lty, lwd 这些参数的意思与*par()*中的参数基本相同，有所区别的是，*par()*中这些参数只能设置一个单值，而这里可以对它们设置一个向量，这个向量的值将依次运用到各个元素上，若向量长度短于元素个数，那么向量会被循环使用，直到所有的元素都被画出来，事实上，向量的循环使用也是R图形参数的一大特点

bg 背景色；注意与*par()*不同的是，这里设置的只是可以画背景色的点[7]的背景色，而不是设置整幅图形的背景色!

[6]但我们可以通过`par()$usr`获得一幅图的坐标系界限，而这里的两个参数就没有这个功能了，因为一般来说作图函数不会返回任何值（或者说返回值为空：`NULL`）

[7]在4.2节中说明了什么类型的点可以画背景色

至此，我们基本上已经介绍完所有常用的图形参数，但这些参数的作用没有必要全都烂熟于心；本章可以仅作为参考资料，需要时查阅即可。后面的章节中，我们就会看到一些参数在图形元素和统计图形中的微调作用（但也许不是微小作用）。

第四章　元素

他从信封里拿出一页四折叠的半张*13×17*英寸的信纸。他把信纸打开铺在桌上，中间有一行铅字拼贴成的句子:

> 若你看重自己生命的价值或还有理性，那就远离沼地。

只有“沼地”两个字是用墨水写的。

“现在，”亨利·巴斯克维尔爵士说，“福尔摩斯先生，也许您可以告诉我，这是什么意思，到底是谁对我的事这么感兴趣呢?”

……“可是，亲爱的华生，两者之间的联系非常紧密，短信中的每个单字都是从这个长句中抽出来的。例如: ‘你’、‘你的’、‘生’、‘命’、‘理性’、‘价值’、‘远离’等，你现在还看不出这些字是从那里面弄来的吗?”

“天啊! 太对了! 唉呀，您可聪明绝顶!”亨利爵士喊了起来。

—柯南·道尔《巴斯克维尔的猎犬》

任何一幅统计图形都是由最基础的图形元素构成的，这些元素我们并不陌生，无非就是颜色、点、线（直线、曲线、线段甚至箭头）、矩形、任意多边形、文本以及图例。R提供了相应的一系列函数，用以向已有的图形中添加图形元素。事实上，R的所有作图函数分为两类，一类是高层函数（high-level），用以生成新的图形，另一类就是低层函数（low-level），这一类指的正是绘制图形元素的这些基础函数。

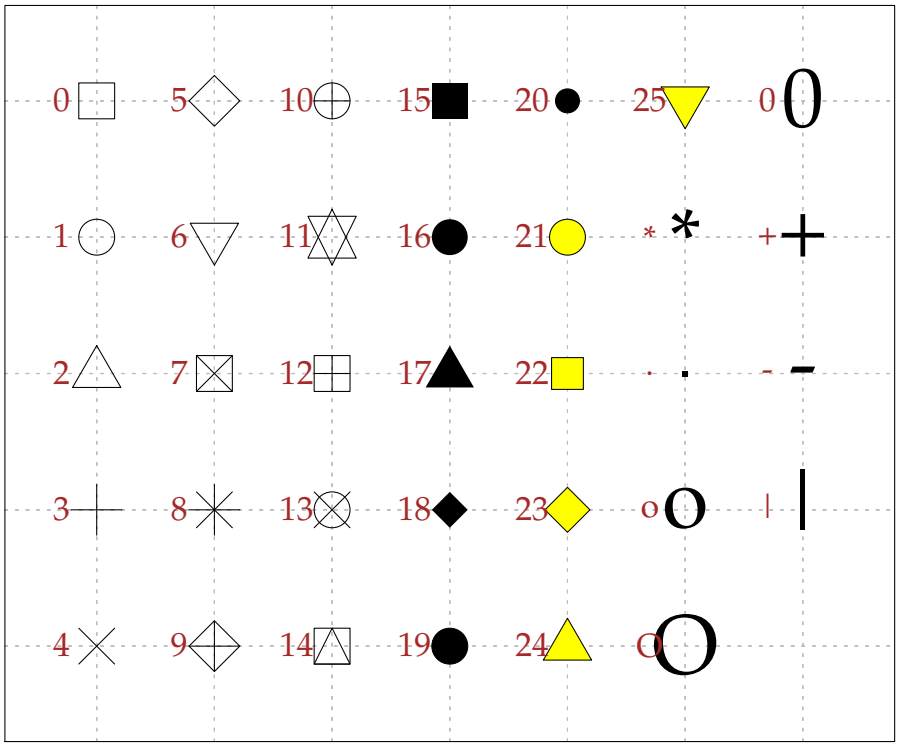


图 *4.1:* 点的类型：*pch*参数取值从*1*到*25*及其它符号（注意：*21~25*的点可以填充背景颜色）

4.1 颜色

4.2 点

4.3 曲线、直线、线段、箭头、X-样条

4.4 矩形、多边形

4.5 网格线

4.6 标题、任意文本、周边文本

4.7 图例

4.8 坐标轴

第五章　图库

> 他解释说：“你要明白，我认为人的大脑原本像一间空空的屋子，必须有选择地用一些家具填满它。只有笨蛋才把他碰到的各种各样的破烂都塞进去。这样的话，那些可能用得上的知识就被挤了出来；或者，充其量也只是把那些破烂同其它东西混杂在一块儿。结果，在需要时却难得找到了。因此，一个善于工作的人，对于将什么东西纳入自己的头脑里是非常仔细的。他只会容纳那些工作时用得着的工具，而且又将这些工具分门别类，安排得井然有序。如果认为这间屋子的墙壁富有弹性，可以随意扩展，那就大错特错了。毫无疑问，总有一天，当你增加点滴知识时，却把从前熟悉的知识给忘记了。因此，不要让无用的信息挤掉那些有用的信息，这一点是至关重要的。”
>
> —柯南·道尔《血字的研究》

本章中，我们结合R语言中的相关函数以及数据实例对各种统计图形依次作出介绍。从第5.1节到5.21节的所有图形都是基于**graphics**包所作，其后的图形均来自于其它函数包。

5.1　直方图

直方图（Histogram）是展示连续数据分布最常用的工具，它本质上是对密度函数的一种估计。在介绍作图方法之前我们有必要先了解一下它的基本数学思想，本节仅作简要介绍，详细的数学理论参见Scott (1992)。

我们知道，对于连续随机变量来说，其密度函数即为分布函数的导数：

```
> par(mfrow = c(2, 2), mar = c(2, 3, 2, 0.5), mgp = c(2,
+     0.5, 0))
> data(geyser, package = "MASS")
> hist(geyser$waiting, main = "(1) freq = TRUE", xlab = "waiting")
> hist(geyser$waiting, freq = FALSE, xlab = "waiting",
+     main = "(2) freq = FALSE")
> hist(geyser$waiting, breaks = 5, density = 10, xlab = "waiting",
+     main = "(3) breaks = 5")
> hist(geyser$waiting, breaks = 40, col = "red", xlab = "waiting",
+     main = "(4) breaks = 40")
```

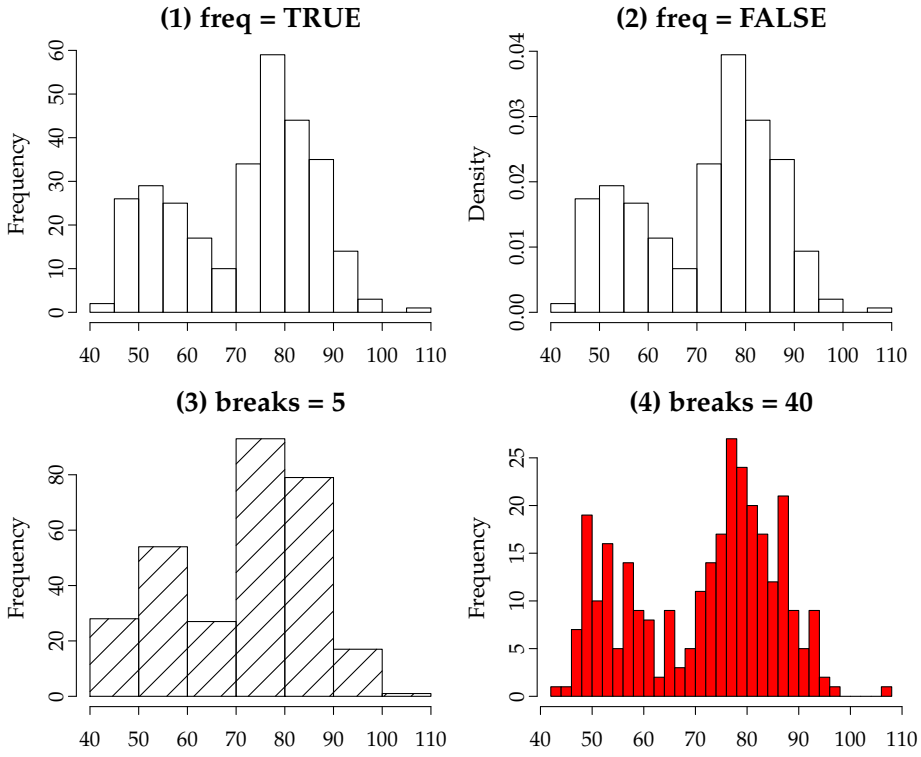


图 5.1: 喷泉间隔时间直方图:（1）使用默认参数值（作频数图）;（2）概率密度直方图;（3）减小区间段数，直方图看起来更平滑（偏差大，方差小）;（4）增大区间段数，直方图更突兀（偏差小，方差大）。

```
> # 注意：以下两行代码足够添加密度曲线，本代码只是为了美观效果
> # hist(geyser$waiting, probability = TRUE, main = '')
> # lines(density(geyser[['waiting']]))
> par(mar = c(1.8, 3, 0.5, 0.1), mgp = c(2, 0.5, 0))
> data(geyser, package = "MASS")
> hst = hist(geyser$waiting, probability = TRUE, main = "",
+     xlab = "waiting")
> d = density(geyser$waiting)
> polygon(c(min(d$x), d$x, max(d$x)), c(0, d$y, 0),
+     col = "lightgray", border = NA)
> lines(d)
> ht = NULL
> brk = seq(40, 110, 5)
> for (i in brk) ht = c(ht, d$y[which.min(abs(d$x -
+     i))])
> segments(brk, 0, brk, ht, lty = 3)
```

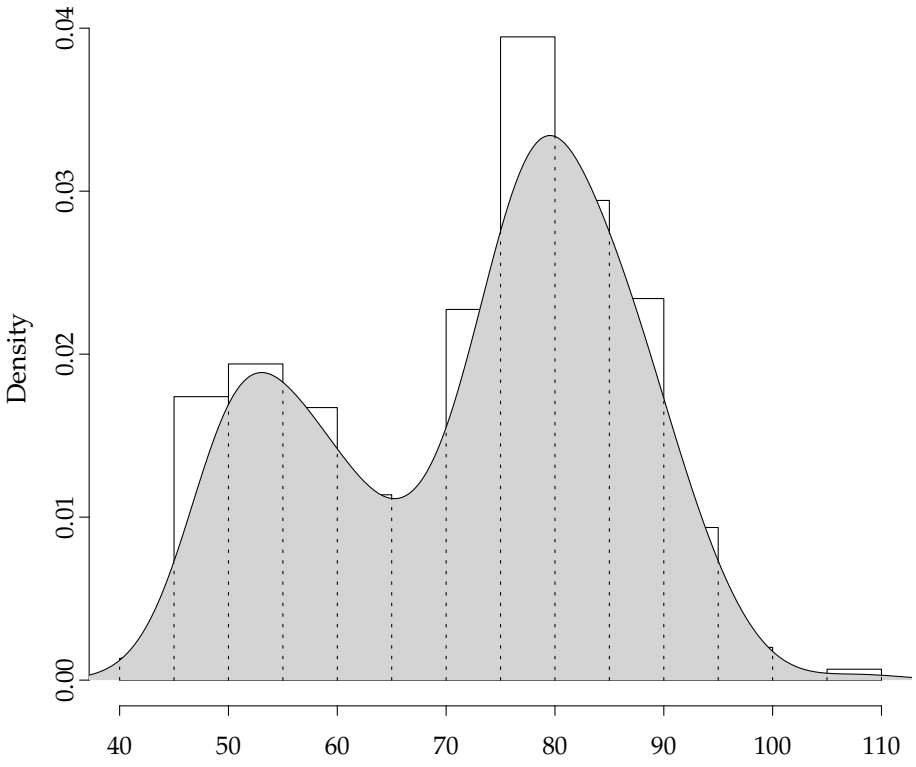


图 *5.2:* 直方图与密度曲线的结合：借助函数density()可以计算出数据的核密度估计，然后利用低层作图函数lines()将核密度估计曲线添加到直方图中。注意本图中其它低层作图函数的使用。

$$f(x) = F'(x) = \lim_{h \to 0} \frac{F(x+h) - F(x)}{h} \tag{5.1}$$

因此我们不妨自然而然地从分布函数的估计出发得到密度函数的估计。当我们拿到一批数据$X_1, X_2, \ldots, X_n$时，我们最容易想到的分布函数估计就是经验分布函数：

$$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}(X_i \leq x) \tag{5.2}$$

其中$\mathbf{I}(\cdot)$为示性函数；结合公式5.1和5.2以及示性函数的性质，我们可以直接得到以下密度函数估计：

$$\hat{f}_n(x) = \lim_{h \to 0} \frac{1}{n} \sum_{i=1}^{n} \frac{\mathbf{I}(x < X_i \leq x+h)}{h} \tag{5.3}$$

公式5.3实际上已经给出了直方图作为密度函数估计工具的基本思想：划分区间并计数有多少数据点落入该区间。实际数据不可能无限稠密，因此$h \to 0$的条件往往是不可能实现的，于是我们退而求其次，只是在某一些区间段里面估计区间上的密度。首先我们将实数轴划分为若干宽度为h的区间（我们称h为“窗宽”）：

$$b_1 < b_2 < \cdots < b_j < b_{j+1} < \cdots;\ b_{j+1} - b_j = h,\ j = 1, 2, \cdots \tag{5.4}$$

然后根据以下直方图密度估计表达式计算区间$(b_j, b_{j+1}]$上的密度估计值：

$$\hat{f}_n(x) = \frac{1}{nh} \sum_{i=1}^{n} \mathbf{I}(b_j < X_i \leq b_{j+1});\ x \in (b_j, b_{j+1}] \tag{5.5}$$

最后我们将密度估计值以矩形的形式表示出来，就完成了直方图的基本制作。当然我们没有必要使用这样原始的方式制作直方图，R中提供了*hist()*函数，其默认用法如下：

```
> usage(hist, "default")

hist(x, breaks = "Sturges", freq = NULL, probability = !freq,
    include.lowest = TRUE, right = TRUE, density = NULL,
    angle = 45, col = NULL, border = NULL, main = paste("Histogram of",
        xname), xlim = range(breaks), ylim = NULL, xlab = xname,
```

```
ylab, axes = TRUE, plot = TRUE, labels = FALSE, nclass = NULL,
...)
```

其中，x为欲估计分布的数值向量；breaks决定了计算分段区间的方法，它可以是一个向量（依次给出区间端点），或者一个数字（决定拆分为多少段），或者一个字符串（给出计算划分区间的算法名称），或者一个函数（给出划分区间个数的方法），区间的划分直接决定了直方图的形状，因此这个参数是非常关键的；freq和probability参数均取逻辑值（二者互斥），前者决定是否以频数作图，后者决定是否以概率密度作图（这种情况下矩形面积为1）；labels为逻辑值，决定是否将频数的数值添加到矩形条的上方；其它参数诸如density、angle、border均可参见低层作图函数“矩形”（*rect()*，4.4节）。

我们以黄石国家公园喷泉数据geyser(Venables and Ripley 2002)为例。图5.1展示了喷泉喷发间隔时间的分布情况。（1）和（2）中的直方图看起来形状完全一样，区别仅仅是前者为频数图，后者为密度图，二者在统计量上仅相差一个常数倍，但密度直方图的一个便利之处在于它可以方便地添加密度曲线，用以辅助展示数据的统计分布（图5.2即为一个示例）；（3）和（4）的区别在于区间划分段数，我们可以很清楚看出区间划分的多少对直方图的直接影响。关于区间划分的一些讨论可以参考Venables and Ripley (2002)，这里我们需要特别指出的是，直方图的理论并非想象中或看起来的那么简单，窗宽也并非可以任意选择，不同的窗宽或区间划分方法会导致不同的估计误差，关于这一点，Excel的直方图可以说是非常不可靠的，因为它把区间的划分方法完全交给了用户去选择，这样随意制作出来的直方图很可能会导致大的估计误差、掩盖数据的真实分布情况。另外一点需要提醒的是关于直方图中的密度曲线，SPSS软件在绘制直方图时会有选项提示是否添加正态分布密度曲线，这也是完全的误导，因为数据不一定来自正态分布，添加正态分布的密度曲线显然是不合理的，相比之下，图5.2的做法才是真正从数据本来的分布出发得到的密度曲线。

直方图函数在作图完毕之后会有一些计算返回值，这些值对于进一步的作图或者分析很有用，例如区间划分端点、频数（或密度）、区间中点等等，这些信息可以被灵活应用在图形定制上（例如图B.3）。

5.2 茎叶图

```
> stem(islands)

  The decimal point is 3 digit(s) to the right of the |

   0 | 000000000000000000000000000000111111222338
   2 | 07
   4 | 5
   6 | 8
   8 | 4
  10 | 5
  12 |
  14 |
  16 | 0
```

```
> stem(islands, scale = 2)

  The decimal point is 3 digit(s) to the right of the |

   0 | 000000000000000000000000000000111111222338
   1 |
   2 |
   3 | 07
   4 |
   5 | 5
   6 | 8
   7 |
   8 |
   9 | 4
  10 |
  11 | 5
  12 |
  13 |
  14 |
  15 |
  16 |
  17 | 0
```

```
> # 或者尝试stem(islands, width = 20)
> # 看看茎叶图的不同效果
```

图 *5.3:* 本图展示了世界上主要的*48*块大陆面积的分布状况，可以明显看出，这些面积数据是严重右偏的，意即：少数陆地块的面积非常大，而大多数陆地块的面积相对来说都很小。事实上，主要是七大洲的大陆块面积非常大，而其它岛屿诸如海南岛、帝汶岛、九洲岛等面积都相对较小。

茎叶图（Stem-and-Leaf Plot）与直方图的功能类似，也是展示数据密度的一种工具，但相比之下茎叶图对密度的刻画显得非常粗略，而且对原始数据通常会作舍入处理，它只是在早期计算机尚不发达时对于手工整理数据来说比较方便。茎叶图的整体形状如同植物的茎和叶，对于一个数据，通常取其10^n部分为茎（n视所有数据的数量级而定），剩下的尾数为叶，放置于茎旁，这样每隔$m10^n$就对数据作一次归类汇总，将落入区间$[km10^n,\ (k+1)m10^n]$的数据汇集为叶子（k, m为整数，m通常取1，$k = 1, 2, 3, \cdots$），我们不妨称这种区间为一个“节”，节的长度与直方图的“窗宽”本质上是同样的概念。显然，叶子越长则表明该节上数据频数越高。

R中茎叶图的函数为*stem()*，其用法为：

```
> usage(stem)

stem(x, scale = 1, width = 80, atom = 1e-08)
```

参数scale控制着m，即节与节之间的长度（scale越大则m越小）；width控制了茎叶图的宽度，若叶子的长度超出了这个设置，则叶子会被截取到长度width，然后以一个整数表示后面尚有多少片叶子没有被画出来。

下面我们以**datasets**包中islands数据为例说明茎叶图的作法。该数据记录了世界上各大陆地块的面积大小，原始数据前10条如下（单位：千平方英里）：

```
> head(islands, 10)

    Africa   Antarctica         Asia    Australia Axel Heiberg
     11506         5500        16988         2968           16
    Baffin        Banks       Borneo      Britain      Celebes
       184           23          280           84           73
```

可以看出，以上数据中最大的数量级为10^4，而大部分数据的数量级集中在10^1，因此茎上的数量级取作10^3相对比较合适—更大的数量级会导致茎的节数非常少、对分布的刻画过于粗略，而更小的数量级会导致节数过多，使得茎叶图几乎退化为数据的原始表示，这样也难以看出数据的集中趋势。图5.3展示了48块大陆块面积的分布，该茎叶图窗宽为2×10^3，图中注明了原始数据小数点位置在“|”后面三位数处，因此我们从图中“还原”原始数据时，需要用（“茎的区间”+“叶”）$\times 10^3$。我们以图5.3为例说明一下茎叶图的制作过程及其相应解释。首先我们将原始数据除以10^3，并四舍五入到小数点后的一位数：

```
> # 去掉陆地名称以便显示数据
> unname(sort(round(islands/1000, 1)))
```

```
 [1]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
[13]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
[25]  0.0  0.0  0.0  0.0  0.0  0.1  0.1  0.1  0.1  0.1  0.1  0.2
[37]  0.2  0.2  0.3  0.3  0.8  3.0  3.7  5.5  6.8  9.4 11.5 17.0
```

然后从0到18×10^3、以2×10^3为窗宽，分段整理数据，每一段（节）中依次放置落入该段的数据的小数位，堆砌起来便形成了茎叶图的叶子。例如11.5落入了$[10, 12]$的区间，我们就将尾数5放在10的右边；类似地，17.0在$[16, 18]$之间，我们将0放在16右边；关于茎叶图顶部的一长串0的解释此处不再赘述。

图5.4是利用泊松分布随机数生成的茎叶图，由于窗宽为1，不存在舍入问题，所以图形可以还原到原始数据，请读者自行对应数据观察茎叶图。

经过前面的说明，现在我们不妨将茎叶图简单理解为横放着的直方图，只是茎叶图通常都以某个便利的整数为窗宽，不如直方图那样精细。此外，茎叶图曾经的优势（简单、可手工绘制）在今天这个计算机时代也显得并不突出，因此，除非特殊情况，我们建议主要使用直方图作为密度函数估计工具。

5.3 箱线图

箱线图（Box Plot或Box-and-Whisker Plot）主要是从四分位数的角度出发描述数据的分布，它通过最大值（Q_4）、上四分位数（Q_3）、中位数（Q_2）、下四分位数（Q_1）和最小值（Q_0）五处位置来获取一维数据的分布概况。我们知道，这五处位置之间依次包含了四段数据，每段中数据量均为总数据量的1/4。通过每一段数据占据的长度，我们可以大致推断出数据的集中或离散趋势（长度越短，说明数据在该区间上越密集，反之则稀疏）。

R中相应的函数为*boxplot()*，其用法如下：

```
> # 默认用法
> usage(boxplot, "default")
```

```
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
    notch = FALSE, outline = TRUE, names, plot = TRUE, border = par("fg"),
```

```
> # lambda为10的泊松分布随机数
> (x = rpois(50, 10))
```

```
 [1]  6 13 12  6 20  9  6  9  8 12 11  8 10 16 11 11 14  5 10 10
[21]  9 16  4  7  6 12 12 13  6  6  6  9  8  9  7 11 13  9 10 16
[41]  6 15  8  9  4 12 11 10  7  8
```

```
> stem(x, scale = 2)
```

```
  The decimal point is at the |

   4 | 00
   5 | 0
   6 | 00000000
   7 | 000
   8 | 00000
   9 | 0000000
  10 | 00000
  11 | 00000
  12 | 00000
  13 | 000
  14 | 0
  15 | 0
  16 | 000
  17 |
  18 |
  19 |
  20 | 0
```

图 5.4: $\lambda = 10$的泊松分布随机数茎叶图：可以看出数据密度在10附近最高，这与理论相符。注意这幅图可以完全还原到原始数据。

```
    col = NULL, log = "", pars = list(boxwex = 0.8, staplewex = 0.5,
        outwex = 0.5), horizontal = FALSE, add = FALSE,
    at = NULL)
```

```
> # 公式用法
> usage(boxplot, "formula")
```

```
boxplot(formula, data = NULL, ..., subset, na.action = NULL)
```

因为*boxplot()*是一个泛型函数，所以它可以适应不同的参数类型。目前它支持两种参数类型：公式（formula）和数据，后者对我们来说可能更容易理解（给一批数据、作相应的箱线图），而前者在某些情况下更为方便，后面我们会举例说明。参数x为一个数值向量或者列表，若为列表则对列表中每一个子对象依次作出箱线图；range是一个延伸倍数，决定了箱线图的末端（须）延伸到什么位置，这主要是考虑到离群点的原因，在数据中存在离群点的情况下，将箱线图末端直接延伸到最大值和最小值对描述数据分布来说并不合适（图形缺乏稳健性），所以R中的箱线图默认只将图形延伸到离箱子两端range $\times$ $(Q_3 - Q_1)$处，即上下四分位数分别加/减内四分位距（Interquartile Range，简称IQR $\equiv Q_3 - Q_1$）的倍数，超过这个范围的数据点就被视作离群点，在图中直接以点的形式表示出来；width给定箱子的宽度；varwidth为逻辑值，若为TRUE，那么箱子的宽度与样本量的平方根成比例，这在多批数据同时画多个箱线图时比较有用，能进一步反映出样本量的大小；notch也是一个有用的逻辑参数，它决定了是否在箱子上画凹槽，凹槽所表示的实际上是中位数的一个区间估计，其计算式为$Q_2 + / - 1.58\text{IQR}/\sqrt{n}$(McGill *et al.* 1978; Chambers *et al.* 1983)，区间置信水平为95%，在比较两组数据中位数差异时，我们只需要观察箱线图的凹槽是否有重叠部分，若两个凹槽互不交叠，那么说明这两组数据的中位数有显著差异（P值小于0.05）；horizontal为逻辑值，设定箱线图是否水平放置；add设置是否将箱线图添加到现有图形上（例：图5.8）；其它参数诸如设置箱子颜色、位置、更详细的宽度等参见?boxplot。

绘制单个箱线图时只需要给*boxplot()*传入一个数值向量即可，如：boxplot(rnorm(100))；这里我们主要使用公式型的参数，以**datasets**包中的杀虫剂数据InsectSprays为例；该数据有两列，第一列为昆虫数目，第二列为杀虫剂种类（ABCDEF），这里是随机抽取的5列数据：

```
> InsectSprays[sample(nrow(InsectSprays), 5), ]
```

```
> par(mar = c(2, 2.5, 0.2, 0.1))
> boxplot(count ~ spray, data = InsectSprays, col = "lightgray",
+     horizontal = TRUE, pch = 4)
```

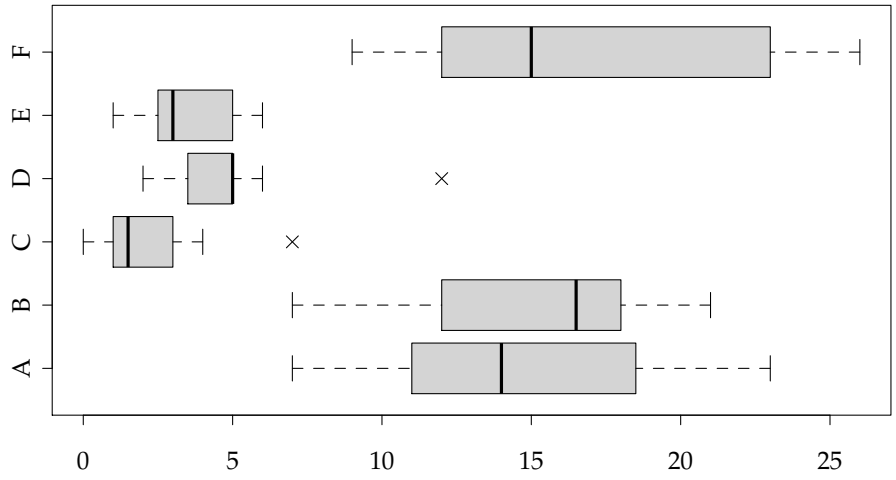


图 5.5: 昆虫数目箱线图：六种杀虫剂下昆虫的数目分布。

```
   count spray
39    12     D
66    16     F
55     1     E
16    11     B
19    17     B
```

为了了解杀虫剂的效果，我们需要对各种杀虫剂下昆虫的数目作出比较。图5.5是一个简单的箱线图展示，不难看出，除了B和D对应的昆虫数据呈左偏形态外，其它组均有右偏趋势，看起来各组数据的平均水平差异比较明显；另外注意观察图中的两个离群点（以“×”表示）。总体看来，C的效果最好。事实上，我们可以对这个数据作方差分析，检验杀虫剂类型对昆虫数目是否有显著影响：

```
> insects.aov = aov(count ~ spray, data = InsectSprays)
> summary(insects.aov)
```

```
             Df Sum Sq Mean Sq F value    Pr(>F)
spray         5 2668.8  533.77  34.702 < 2.2e-16 ***
Residuals    66 1015.2   15.38
```

```
> par(mar = c(2, 2.5, 0.2, 0.1))
> x = rnorm(150)
> y = rnorm(50, 0.5)
> boxplot(list(x, y), names = c("x", "y"), col = 2:3,
+     horizontal = TRUE, notch = TRUE, varwidth = TRUE)
> wilcox.test(x, y)
```

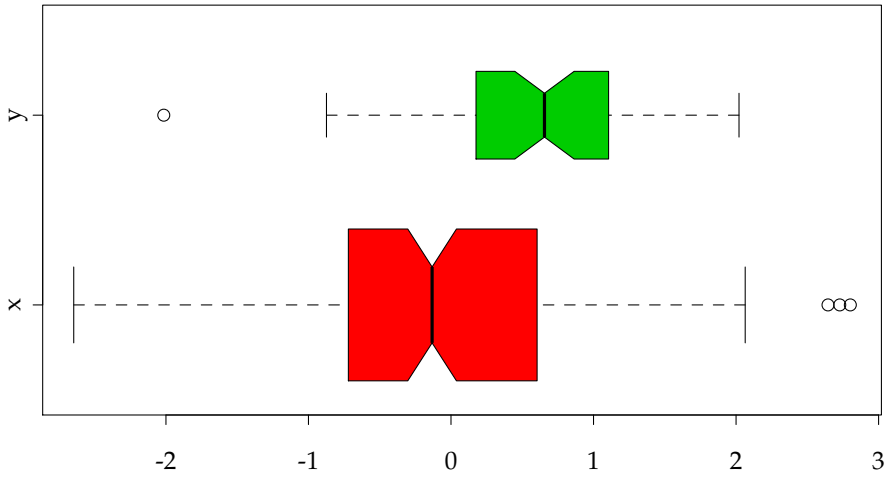


图 *5.6:* 箱线图的凹槽与统计推断：从凹槽不交叠的情况来看，两样本中位数有显著差异。

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

上述分析告诉我们杀虫剂类型有显著影响（P值接近于0），也印证了我们对图形的观察。

最后我们再以一个模拟数据的例子展示箱线图凹槽的功能。这里我们分别从正态分布$N(0,1)$和$N(0.5,1)$中各自产生150和50个随机数，然后作箱线图比较两组数据中间位置的差异。图5.6为一次模拟的结果，图中的凹槽表明了两组数据的中位数有显著差异，Wilcoxon秩和检验也证实了这一结论。此外，该图还使用了varwidth参数以表明两组数据样本量的大小不同。

5.4 条形图

如同前面1.5节中曾经提到的，条形图目前是各种统计图形中应用最广

```
> par(mfrow = c(2, 1), mar = c(3, 2.5, 0.5, 0.1))
> death = t(VADeaths)[, 5:1]
> barplot(death, col = heat.colors(4))
> barplot(death, col = heat.colors(4), beside = TRUE,
+     legend = TRUE)
```

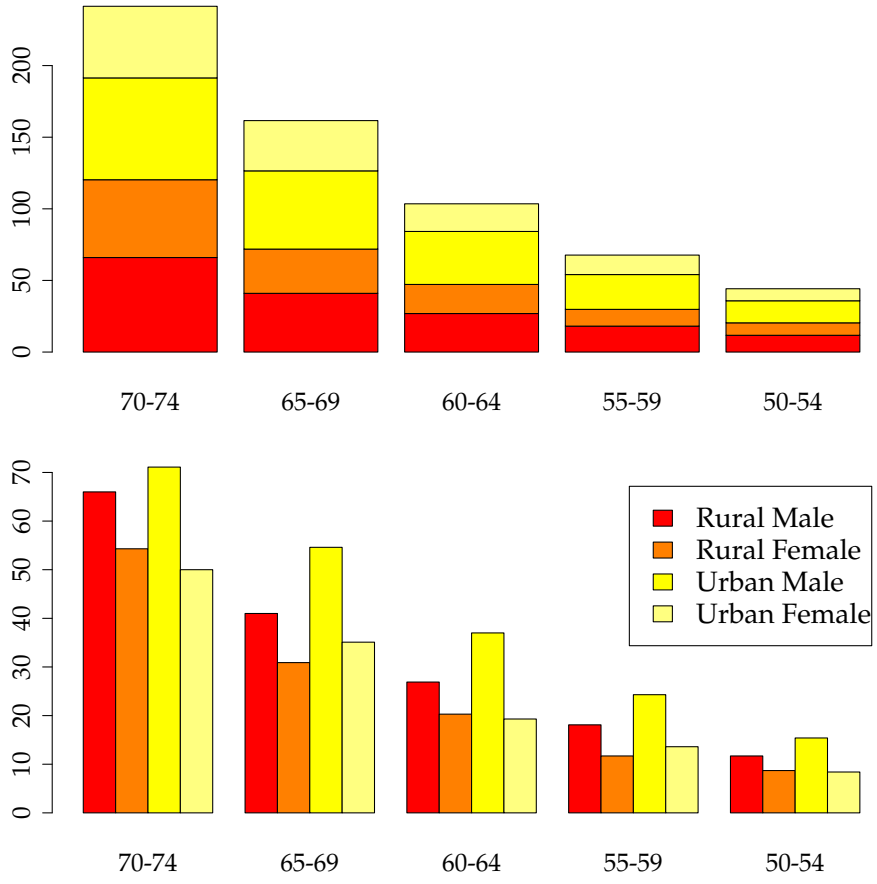


图 5.7: 弗吉尼亚死亡率数据条形图：堆砌和并列的条形图效果。

泛的，但条形图所能展示的统计量比较贫乏：它只能以矩形条的长度展示原始数值，对数据没有任何概括或推断。

R中条形图的函数为*barplot()*，用法如下：

```
> usage(barplot, "default")

barplot(height, width = 1, space = NULL, names.arg = NULL,
    legend.text = NULL, beside = FALSE, horiz = FALSE, density = NULL,
    angle = 45, col = NULL, border = par("fg"), main = NULL,
    sub = NULL, xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
    xpd = TRUE, log = "", axes = TRUE, axisnames = TRUE,
    cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
    inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
    add = FALSE, args.legend = NULL, ...)
```

条形图的主要参数是height，它指定了长条的长度，这个参数可以接受一个数值向量或者一个数值矩阵作为参数，前者容易理解，后者稍有些复杂，当传入一个矩阵时，条形图针对矩阵的每一列画图，若beside为FALSE，则矩阵每一列占据一条的位置，该条由若干矩形堆砌而成，这些矩形的长度对应着矩阵的行数据，若beside为TRUE，这些矩形则并排排列而非堆砌；width可以设置条的宽度；space用以设置条之间的间距；names.arg为条形图的标签，即每一条的名称；legend.text参数在height为矩阵时比较有用，可以用来添加图例；horiz用以设置条形图的方向（水平或垂直）；density、angle等参数可以参考矩形的章节（4.4节）；plot为逻辑值，决定是否将条形图添加到现有图形上。

图5.7展示了参数beside和legend.text（下图）的效果。该图以1940年弗吉尼亚州分年龄组、分地区和分性别死亡率数据VADeaths为基础，展示了各组之间死亡率的差异，其中堆砌的条形图容易比较各年龄组总死亡率的大小，显然年龄越高死亡率越大，而并列的条形图容易比较组内的城乡和性别差异，一般说来，男性死亡率高于女性，农村男性死亡率低于城市男性，但女性的城乡差异没有明显规律。由于人眼对长度比比例更敏感（例如在区分城乡和性别差异时，图5.7的上图就不如下图直观），所以我们制图时要考虑清楚我们想展示的是数据的哪一方面，即：将最关键的信息用最能激发视觉感知的形式表现出来。

```
> # 弗吉尼亚州死亡数据
> VADeaths
```

	Rural Male	Rural Female	Urban Male	Urban Female
50-54	11.7	8.7	15.4	8.4
55-59	18.1	11.7	24.3	13.6
60-64	26.9	20.3	37.0	19.3
65-69	41.0	30.9	54.6	35.1
70-74	66.0	54.3	71.1	50.0

5.5 饼图

5.6 关联图

5.7 Cleveland点图

5.8 条件密度图

5.9 协同图

5.10 一元函数曲线图

5.11 四瓣图

5.12 颜色图

5.13 马赛克图

5.14 散点图矩阵

Scatter plot matrix

图 *5.8: 2005*年中国*31*地区五大国民素质特征分布温度计图

5.15 三维透视图

5.16 因素效应图

5.17 坐标轴须

5.18 棘状图

5.19 带状图

5.20 向日葵散点图

5.21 符号图

5.22 QQ图

有人问PP图怎么画，这是个问题么？留做习题。

5.23 地图

5.24 小提琴图

5.25 脸谱图

5.26 平行坐标图

5.27 调和曲线图

5.28 习题

1. 在第5.22节中我们了解了如何画QQ图，与之对应的还有一种图形叫PP图（Probability-Probability），它也是一种检验数据分布是否和理论分布吻合的图形工具，原理和QQ图类似：对数据的实际概率分布值和理论概率分布值作散点图即可（也可以选择性地添加一条直线）。编写函数画出`geyser$waiting`的PP图（理论分布选择正态分布，均值和标准差用矩估计获得），并评价该数据的正态性。[1]

[1]本题源于COS会员thinkyou的提问：`http://cos.name/cn/topic/18521`

第六章　系统

> “我不想用各种说法和怀疑来影响你，华生，”他说，“我只要求你将各种事实尽可能详尽地报告给我，至于归纳推理就留给我好了。”
>
> “哪些事实呢？”我问道。
>
> “与该案可能有关的任何事实，无论是多么地间接，特别是年轻的巴斯克维尔与邻里的关系或与查尔兹爵士暴卒有关的任何新的问题。”
>
> —柯南·道尔《巴斯克维尔的猎犬》

本章是否要考虑移到最后？……

6.1　网格图形

除了基础图形系统之外，R还自带另一套图形系统即grid（网格图形）。grid包主要由Paul Murrell开发维护，它的设计初衷是为了克服基础图形系统中图形元素不能动态修改的弱点，例如当一个矩形被画出来之后就无法仅仅更改这个矩形的颜色，而只能重画整幅图形，并更改矩形颜色。例如：

```
> # 创建一个名叫rect0的矩形
> grid.rect(name = "rect0")
> # 修改它的填充颜色为红色
> grid.edit("rect0", gp = gpar(fill = "red"))
> # 修改为蓝色，不需要重新用grid.rect()画矩形
> grid.edit("rect0", gp = gpar(fill = "blue"))
> # 若用基础图形系统，则需要这样
```

```
> plot(0:1, 0:1)
> rect(0, 0, 1, 1, col = "red")
> # 为了改变颜色，重画整幅图形
> plot(0:1, 0:1)
> rect(0, 0, 1, 1, col = "blue")
> # 当然，这里可以用新的矩形覆盖旧的，但旧矩形仍然存在
```

6.2 lattice图形

lattice(Sarkar 2010)是基于**grid**包的一套统计图形系统，它的图形设计理念来自于Cleveland (1993)的Trellis图形[1]，其主要特征是根据特定变量（往往是分类变量）将数据分解为若干子集，并对每个子集画图。就像数理统计中的条件期望、条件概率一样，**lattice**的图形也是一种“条件作图”。

6.3 ggplot2图形

上一节中**lattice**虽然灵活，但它无穷无尽的选项往往让用户感到迷茫。**ggplot2**(Wickham 2009)从易用性出发，结合了基础图形系统的简便以及**grid**和**lattice**的灵活，并以“The Grammar of Graphics”一书(Wilkinson 2005)的理论为支撑，构建了一套实用而且美观的图形系统。

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

第七章　模型

> “嗯，人的身高十有八九可以从他的步幅的长度推测出来，这极容易推算，但是让我把枯燥的数字摆出来算给你看实在是毫无用处。我在屋外的粘土路上和屋内的尘土上找到了那人的脚印，此外我还有一个法子验证我的计算：当一个人在墙上写字时，他会本能地将字写在视线以上的地方，而那个血字正好离地面六英尺。这推算实在是简单得像儿戏一般。”
>
> 我问：“那他的年龄呢？”
>
> “好的，如果一个人不费吹灰之力就能一步跨出四英尺半，他不可能年老体衰，因为花园小径上的泥坑恰好长四英尺半，他显然是一步跨过去的，漆皮靴则是绕过去的，所以这也没有什么神秘之处。我只不过是将我在那篇文章中推崇的一些观察和演绎的规则应用在日常生活中罢了。你还有什么不明白的地方吗？”
>
> —柯南·道尔《血字的研究》

在1.5小节中我们曾经提到三种基本的统计分析方式，其中包括推断统计分析和探索统计分析，表面上看来，统计模型在前者中占主导地位，而统计图形在后者中往往起着非常重要的作用，然而本章要探讨和揭示的则是以探索为目标的统计图形与以推断为目标的统计模型之间的联系。

7.1　线性回归模型

- 一元回归：散点图
- 回归诊断：方差齐性假设、线性假设、独立性假设、正态性假设、离群点

7.2 方差分析

- 类似回归
- 箱线图等

7.3 非参数回归模型

7.3.1 局部加权回归散点平滑法

- http://cos.name/2008/11/lowess-to-explore-bivariate-correlation-by-yihui/

7.4 稳健回归模型

- 离群点

7.5 广义线性模型

- 回归诊断

7.6 分类数据模型和列联表

- 马塞克图

7.7 混合效应模型

- Gelman讲“爷为啥不关心多重比较的问题”：http://www.stat.columbia.edu/~gelman/presentations/multiple_minitalk2.pdf（随机效应模型究竟做了什么？）

7.8 主成分分析和因子分析

- 成分方差

- Biplot

7.9 聚类分析

- 谱系图
- K-Means的散点图

7.10 判别分析

- 成分的散点图

7.11 对应分析

7.12 多维标度分析

- 高维数据的二维（低维）表示

7.13 时间序列模型

经典ARMA、谱分析等

7.14 生存分析

7.15 空间统计学

7.16 数据挖掘和机器学习

7.16.1 分类与回归树

7.16.2 Bootstrap

7.16.3 支持向量机

第八章　数据

> “哦，你还不知道吧?”他笑了，大声说道。“很惭愧，我写了几篇专论，都是技术方面的。比方说，有一篇叫《论各种烟灰的辨别》。此文列举了一百四十种雪茄烟。卷烟、烟斗烟丝的烟灰，并附有彩色插图，以说明烟灰的区别。这是刑事审判中常常出现的重要证据，有时还是案件的重要线索。举例说，如果你断定某一谋杀案系一个抽印度雪茄的男人所为，显然缩小了侦查范围。在训练有素的人看来，印度雪茄的黑灰与‘鸟眼’牌的白灰的区别，正如白菜和土豆的区别一样大。”
>
> —柯南·道尔《四签名》

本章中我们主要从数据类型的角度对第五章中所叙述的统计图形进行一个比较全面的梳理与总结，即：什么样的数据适合用什么样的统计图形展示，以及数据中的信息以怎样的方式表达。本章中的数据类型主要分为离散数据和连续数据以及它们的混合，在各种数据类型中，我们分别对低维和高维数据的图示作出总结。

8.1　离散数据

8.1.1　一维数据

- 条形图

8.1.2　多维数据

- 马塞克图

8.2 连续数据

8.2.1 一维数据

- 直方图
- 箱线图

8.2.2 二维数据

- 散点图

8.2.3 高维数据

- 平行坐标图
- 降维

8.3 混合数据

8.3.1 一维数据

8.3.2 二维数据

8.3.3 高维数据

附录 A　程序初步

如第二章所讲，R的编程方式是面向对象（Object-Oriented）的，这里我们把R中的数据类型简要介绍一下，以便读者能熟练操纵数据；此外，我们也简要介绍一下R编程中的选择与循环语句以及输入输出的操作。

A.1　对象类型

在R的系统中，几乎任何东西都是对象。使用对象的好处在于它们都可以重用（Reuse）。例如我们可以建立并拟合一个回归模型（不妨称之为`fit`），这个对象中包含了若干子对象，在后面的计算中我们随时可以调用这个对象中的子对象，如残差向量（`fit$residuals`或`resid(fit)`）、系数估计（`fit$coefficients`或`coef(fit)`）等。面向对象的编程方式尤其在涉及到大量计算的工作中会大显身手，刚才我们提到的只是做一个回归模型，看起来优势并不明显，但如果我们想用某个因变量针对1000个自变量分别作回归，然后看看回归系数的t值或者AIC值的分布情况等等，这时“对象”操作的便利性就充分体现出来了，相比之下，读者不妨考虑用SPSS或其它软件如何完成类似的任务及其难度。掌握了R的对象之后，在R的世界编程基本就可以畅通无阻了。

A.1.1　向量

向量（vector）是最简单的数据结构，它是若干数据点的简单集合，如从1到10的数字：

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
```

通常我们可以用函数*c()*拼接一些数字或字符生成一个向量，如：

```
> c(7.11, 9.11, 9.19, 1.23)
```

```
[1] 7.11 9.11 9.19 1.23
```

我们可以将一个向量赋值给一个变量：

```
> (x <- c(7.11, 9.11, 9.19, 1.23))
```

```
[1] 7.11 9.11 9.19 1.23
```

注意R中赋值符号可以是<-或=，或->（从左往右赋值），或者使用*assign()*函数进行赋值。向量的运算一般都是针对每一个元素的运算，如：

```
> 1/x
```

```
[1] 0.1406470 0.1097695 0.1088139 0.8130081
```

```
> x + 1
```

```
[1]  8.11 10.11 10.19  2.23
```

实际上以向量的形式进行元素运算是R语言计算的重要特征。通过中括号和下标值可以提取向量中的元素或者改变相应位置的元素：

```
> x[c(1, 4)]
```

```
[1] 7.11 1.23
```

```
> # 将x赋值给tmp
> (tmp = x)
```

```
[1] 7.11 9.11 9.19 1.23
```

```
> tmp[1] = 10
> tmp
```

```
[1] 10.00  9.11  9.19  1.23
```

利用现有的向量可以继续利用*c()*生成新的向量：

```
> (y = c(x, 12.19))
```

```
[1]  7.11  9.11  9.19  1.23 12.19
```

向量的长度可以用*length()*获得：

```
> length(y)
```

```
[1] 5
```

我们还可以用*names()*给向量的每一个元素命名：

```
> names(x) = LETTERS[1:length(x)]
> x
```

```
   A    B    C    D
7.11 9.11 9.19 1.23
```

对于有名称的向量，我们可以用名称提取向量的元素（获取数据子集的方式通常有三种：整数下标、子对象名称以及逻辑值）：

```
> x[c("B", "A")]
```

```
   B    A
9.11 7.11
```

函数*sort()*可以对向量排序（顺序或倒序）：

```
> sort(x)
```

```
   D    A    B    C
1.23 7.11 9.11 9.19
```

因为很多统计量的计算是针对一维数据的，所以用向量操作起来会非常方便，例如计算样本方差$\sum_{i=1}^{n}(x_i - \bar{x})^2/(n-1)$：

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

```
[1] 14.03027
```

```
> # 自带函数var()作为对比
> var(x)
```

```
[1] 14.03027
```

当然R提供了现成的方差函数*var()*，我们不必将代码写得那么复杂，从上面的输出可以看出，直接根据公式写的代码和方差函数计算的结果是一样的。另外，向量操作可以节省显式循环的使用，如果在C语言或VB等其它程序语言中，我们只能使用几段循环来计算方差数值，因为其中涉及到两个求和函数。

使用函数*seq()*和*rep()*可以生成规则的序列，前者提供了等差数列的功能，后者可以将向量或元素重复多次从而生成新的向量，如：

```
> # 冒号表示步长为1或-1的序列
> 10:1

 [1] 10  9  8  7  6  5  4  3  2  1

> # 步长为0.2
> seq(7, 9, 0.2)

 [1] 7.0 7.2 7.4 7.6 7.8 8.0 8.2 8.4 8.6 8.8 9.0

> # 生成向量长度为6
> seq(7, 9, length.out = 6)

[1] 7.0 7.4 7.8 8.2 8.6 9.0

> rep(2, 10)

 [1] 2 2 2 2 2 2 2 2 2 2

> # 整个向量重复5次
> rep(1:3, 5)

 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3

> # 每个元素重复5次
> rep(1:3, each = 5)

 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3

> # 每个元素分别重复1、2、3次
> rep(1:3, 1:3)

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

向量除了可以是数值型之外，还可以是逻辑值、字符等，如：

```
> (z = (x < 5))

    A     B     C     D
FALSE FALSE FALSE  TRUE

> # letters和LETTERS是R中的字符常量
> letters

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
[16] "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

注意逻辑值的TRUE和FALSE可以简写为T和F，但强烈建议这两个逻辑值不要使用简写[1]。我们也可以用逻辑向量提取向量的元素，如：

```
> # 满足“非z”的元素
> x[!z]
```

```
   A    B    C
7.11 9.11 9.19
```

R中有三种特殊的值：缺失值NA，非数值NaN和无穷大/无穷小Inf/-Inf。非数值通常由无意义的数学计算产生，如0/0；注意分子不为0而分母为0时，结果是无穷大。

A.1.2 因子

因子（factor）对应着统计中的分类数据，它的形式和向量很相像，只是因子数据具有水平（level）和标签（label），前者即分类变量的不同取值，后者即各类取值的名称。

因子型数据可以由*factor()*函数生成，如：

```
> (x = factor(c(1, 2, 3, 1, 1, 3, 2, 3, 3), levels = 1:3,
+     labels = c("g1", "g2", "g3")))
```

```
[1] g1 g2 g3 g1 g1 g3 g2 g3 g3
Levels: g1 g2 g3
```

我们可以对因子型数据求频数、将其转化为整数或字符型向量。注意整数是因子型数据的本质：它本身以整数存储，但表现出来是字符，原理就是把整数对应的标签显示出来，这种存储方式在很多情况下可以大大节省存储空间（存整数往往比存字符串占用空间小）。

```
> table(x)
```

```
x
g1 g2 g3
 3  2  4
```

```
> as.integer(x)
```

[1]倾向简写这两个逻辑值的人容易倾向简写任何字符，比如变量名；据作者的经验，很多人的程序出错就在这里，因为使用了T或F作为变量名，而在程序的某些地方又将T或F作为逻辑值对待，如这段条件语句x = 1; T = NULL; ...; if ((x > 0) == T) ...会出错，因为条件(x > 0) == T返回的结果是长度为0的逻辑值，不符合if语句的要求。

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

```
> as.character(x)
```

```
 [1] "g1" "g2" "g3" "g1" "g1" "g3" "g2" "g3" "g3"
```

因子型数据在分类汇总时比较有用，例如在*tapply()*中：

```
> y = 1:9
> # x和y并列放的样子
> data.frame(y, x)
```

```
  y  x
1 1 g1
2 2 g2
3 3 g3
4 4 g1
5 5 g1
6 6 g3
7 7 g2
8 8 g3
9 9 g3
```

```
> # x的每一组的均值
> tapply(y, x, mean)
```

```
      g1       g2       g3
3.333333 4.500000 6.500000
```

因子型数据中有一种特殊的类型，就是有序因子，它与普通的因子型数据的区别在于各个分类之间是有顺序的，即统计上所说的定序变量；相关函数为*ordered()*。

A.1.3 数组和矩阵

数组和矩阵是具有维度属性（dimension）的数据结构，注意这里的维度并非统计上所说的“列”，而是一般的计算机语言中所定义的维度（数据下标的个数）。矩阵是数组的特例，它的维度为2。数组和矩阵可以分别由*array()*和*matrix()*函数生成。注意矩阵中所有元素必须为同一种类型，要么全都是数值，要么全都是字符，后面我们会介绍数据框，它会放宽这个要求。这里给出一些示例说明数组和矩阵的用法：

```
> # 3维数组示例
> (x = array(1:24, c(3, 4, 2)))
```

```
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

```
> # 维数
> dim(x)
```

```
[1] 3 4 2
```

```
> # 第3维第1个位置上的元素
> x[, , 1]
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> x[1, , 2]
```

```
[1] 13 16 19 22
```

```
> # 矩阵示例
> (x = matrix(1:12, nrow = 2, ncol = 6))
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
```

```
> x[1, 5]
```

```
[1] 9
```

```
> # 逻辑比较: 是否小于5?
> x < 5
```

```
     [,1] [,2]  [,3]  [,4]  [,5]  [,6]
[1,] TRUE TRUE FALSE FALSE FALSE FALSE
[2,] TRUE TRUE FALSE FALSE FALSE FALSE
```

```
> # 以逻辑矩阵为下标挑选出小于5的元素
> x[x < 5]
```

```
[1] 1 2 3 4
```

```
> # 各个元素的平方
> x^2
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    9   25   49   81  121
[2,]    4   16   36   64  100  144
```

```
> # 矩阵乘法 X * X'（t()为转置函数）
> x %*% t(x)
```

```
     [,1] [,2]
[1,]  286  322
[2,]  322  364
```

```
> # 一个随机方阵
> (x = matrix(sample(9), 3))
```

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

```
> # 求逆
> solve(x)
```

```
           [,1]        [,2]       [,3]
[1,] -0.3333333  0.33333333 -0.3333333
[2,]  0.1666667  0.02564103 -0.1794872
[3,]  0.1666667 -0.20512821  0.4358974
```

```
> # 还有eigen(x)求特征根与特征向量，用qr(x)对矩阵作QR分解，等等
> # 我们还可以用cbind()和rbind()等函数将几个向量拼接成矩阵
```

矩阵相对于数组来说在统计中应用更为广泛，尤其是大量的统计理论都涉及到矩阵的运算。顺便提一下，R包**Matrix**(Bates and Maechler 2010)在处理（高维）稀疏或稠密矩阵时效率会比R自身的矩阵运算效率更高。

A.1.4 数据框和列表

数据框（data frame）和列表（list）是R中非常灵活的数据结构。数据框也是二维表的形式，与矩阵非常类似，区别在于数据框只要求各列内的数据类型相同，而各列的类型可以不同，比如第1列为数值，第2列为字符，第3列为因子，等等；列表则是更灵活的数据结构，它可以包含任意类型的子对象。数据框和列表分别可以由*data.frame()*和*list()*生成。

```
> # 生成一个数据框，前两列为数值型，后一列为字符型
> data.frame(x = rnorm(5), y = runif(5), z = letters[1:5])
```

```
          x         y z
1  1.3821417 0.4399990 a
2 -1.2701430 0.8551456 b
3 -0.5165447 0.3063028 c
4  0.1550370 0.1824483 d
5  1.9079489 0.8478041 e
```

```
> # 包含四个子对象的列表
> (Lst = list(name = "Fred", wife = "Mary", no.children = 3,
+     child.ages = c(4, 7, 9)))
```

```
$name
[1] "Fred"

$wife
[1] "Mary"

$no.children
[1] 3

$child.ages
[1] 4 7 9
```

```
> # 用名称提取子对象
> Lst$child.ages
```

```
[1] 4 7 9
```

```
> # 用整数下标提取子对象
> Lst[[2]]
```

```
[1] "Mary"
```

```
> # 甚至可以试试list(first = Lst, second = 1:10)之类的语句
```

矩阵的本质是（带有维度属性的）向量，数据框的本质是（整齐的）列表。

A.1.5 函数

R中也可以自定义函数，以便编程中可以以不同的参数值重复使用一段代码。定义函数的方式为：`function(arglist) expr return(value)`；其中arglist是参数列表，expr是函数的主体，*return()*用来返回函数值。例如我们定义一个求峰度$\sum_{i=1}^{n}(x_i-\bar{x})^4/(n\cdot \mathrm{Var}(x))-3$的函数*kurtosis()*如下：

```
> kurtosis = function(x, na.rm = FALSE) {
+     # 去掉缺失值
+     if (na.rm)
+         x = x[!is.na(x)]
+     return(sum((x - mean(x))^4)/(length(x) * var(x)^2) -
+         3)
+ }
```

该函数有两个参数，数据向量x和是否删除缺失值na.rm，后者有默认值`FALSE`；下面我们用均匀分布的随机数测试一下：

```
> # 理论上均匀分布的峰度应该小于1，以下为一个模拟结果
> kurtosis(runif(100))
```

```
[1] -1.308220
```

注意，R中有时候不必用函数*return()*指定返回值，默认情况下，函数主体的最后一行即为函数返回值。

最后，因为本书的很多作图函数都是泛型函数，我们也补充一点泛型函数（generic function）的作用原理。实际上泛型函数是根据第一个参数的类（class）去调用了相应的“子函数”。以*plot()*为例，我们用*methods()*查看一下它有哪些作图方法：

```
> methods("plot")
```

```
 [1] plot.Date*            plot.HoltWinters*
 [3] plot.POSIXct*         plot.POSIXlt*
 [5] plot.TukeyHSD         plot.acf*
 [7] plot.correspondence*  plot.data.frame*
 [9] plot.decomposed.ts*   plot.default
[11] plot.dendrogram*      plot.density
[13] plot.ecdf             plot.factor*
[15] plot.formula*         plot.hclust*
[17] plot.histogram*       plot.isoreg*
[19] plot.lda*             plot.lm
```

```
[21] plot.mca*            plot.medpolish*
[23] plot.mlm             plot.ppr*
[25] plot.prcomp*         plot.princomp*
[27] plot.profile*        plot.profile.nls*
[29] plot.ridgelm*        plot.spec
[31] plot.spec.coherency  plot.spec.phase
[33] plot.stepfun         plot.stl*
[35] plot.table*          plot.ts
[37] plot.tskernel*

   Non-visible functions are asterisked
```

可以看出，这些函数都是以*plot.**的形式定义的，其中*便是类的名称。泛型函数的基本原理就是：传给*plot()*的第一个参数是何种类，则调用何种函数进行作图。例如iris的类是data.frame，那么plot(iris)就会调用*plot.data.frame()*作图（散点图矩阵）。下面的代码有助于进一步说明这种原理：

```
> class(iris)

[1] "data.frame"

> plot(iris)
> x = density(faithful$waiting)
> class(x)

[1] "density"

> plot(x)
```

我们也可以对现有的泛型函数进行扩充，使它们适应我们自定义的类。例如*print()*也是一个常用的泛型函数，当我们在R控制台中键入一个对象以显示其内容时，实际上是调用了该函数以打印出对象的内容。现在我们定义一个*print.yihui()*，使得类名称为yihui的对象在屏幕上打印出来时数值为真实数值的2倍：

```
> # 本例告诉我们，眼见未必为实!
> x = 1:5
> class(x)

[1] "integer"

> class(x) = "yihui"
> x
```

```
[1] 1 2 3 4 5
attr(,"class")
[1] "yihui"
```

```
> print.yihui = function(x) print.default(unclass(2 *
+     x))
> x
```

```
[1]  2  4  6  8 10
```

```
> str(x)
```

```
Class 'yihui'  int [1:5] 1 2 3 4 5
```

至此，读者应该能够明白有些作图函数既可以直接接受数据作为参数又可以接受公式作为参数的原因了，如*boxplot()*。

A.2 操作方法

在了解对象的几种基本类型之后，我们需要知道如何对这些对象进行简单的四则运算之外的操作。在计算机程序和算法中，最常见结构的就是选择分支结构和循环结构，通过这样的程序语句，我们可以进一步控制和操纵对象；同时，在执行计算机程序时，我们也常常需要一些输入输出的操作。

A.2.1 选择与循环

一般来说，计算机程序都是按代码先后顺序执行的，而有时候我们希望代码能够按照一定的判断条件执行，或者将一个步骤执行多次，此时我们就需要选择和循环结构的程序。

R提供了如下一些实现选择和循环的方法：

- `if(cond) expr if(cond) cons.expr else alt.expr`
- `for(var in seq) expr`
- `while(cond) expr`

cond为条件（其计算结果为逻辑值TRUE或FALSE），expr为一段要执行的代码，通常放在大括号{}中，除非代码只有一行。

选择结构的函数还有*ifelse()*和*swith()*，请读者自行查阅帮助文件。关于选择和循环，这里仅给出一个简单的综合示例，不再详加说明：

```
> # x初始为空
> x = NULL
> for (i in 1:10) {
+     rnd = rnorm(1)
+     # 如果随机数在[-1.65, 1.65]范围内则放到x中去
+     if (rnd < 1.65 & rnd > -1.65)
+         x = c(x, rnd)
+ }
> x
 [1]  0.4726341 -0.5426061 -0.4470215  1.2056402  0.1281449
 [6] -0.4468570  1.3277169 -0.3790596 -0.9302357 -0.1723591
```

A.2.2 输入与输出

对于统计程序来说，输入的一般是数据，而输出一般是图形和表格，它们在R中都容易实现。前者可以参考*read.table()*系列函数，后者则可以使用图形设备（附录B.5）以及*write.table()*系列函数。具体使用方法请查阅这些函数的帮助文件。

A.3 习题

1. 变量选择问题：某多元线性回归模型包含了20个自变量，根据某些经验，自变量的个数在3个左右是最合适的。要求建立所有可能的回归模型，并根据AIC准则找出最优模型以及相应的自变量。

   提示：所有回归模型数目为组合数$C_{20}^{3}=1140$，可使用函数*combn()*生成所有组合，用*lm()*建模，并用*AIC()*提取AIC值。

2. DNA序列的反转互补[2]：给定一个DNA序列字符串，要求将它的字符序列顺序颠倒过来，然后A与T互换，C与G互换。例如原序列为“TTGGTAGTGC”，首先将它顺序反转为“CGTGATGGTT”，然后互补为“GCACTACCAA”。

   提示：可以采用*substr()*暴力提取每一个字符的方式，然后用繁琐的循环和选择语句实现本题的要求；但利用R的向量化操作功能会大大加

[2]本题源于COS会员iiiiiiiiii的提问：http://cos.name/cn/topic/18471

速计算速度，可采用的函数有：拆分字符串*strsplit()*、反转向量*rev()*、拼接字符串*paste()*。仔细考虑如何利用名字和整数的下标功能实现序列的互补。

3. 接受—拒绝抽样（Acceptance-Rejection Sampling）：当我们不容易从某个分布中生成随机数的时候，若有另一个分布的密度函数$g(x)$能控制前一个分布的密度函数$f(x)$的核$h(x)$（此处“控制”的意思是$\exists M > 0,\ h(x)/g(x) \leq M,\ \forall x$，核$h(x)$正比于$f(x)$），而且我们很方便从$g(\cdot)$中生成随机数，那么我们可用“接受—拒绝抽样”的方式从$f(\cdot)$中生成随机数。步骤如下：

   (a) 从$g(\cdot)$中生成随机数x_i；

   (b) 从均匀分布$U(0,1)$中生成随机数u_i；

   (c) 若$u_i \leq h(x_i)/(M \cdot g(x_i))$，则记下$x_i$；

   重复以上步骤若干次，被接受的x_i的概率密度函数即为$f(\cdot)$。根据以上抽样步骤用适当的循环和选择语句生成服从以下分布的随机数：

$$f(\theta) = c\frac{\exp(\theta)}{1+\exp(\theta)}\frac{1}{\sqrt{2\pi}}\exp(-\theta^2/2)$$

   其中，c为使得$f(\theta)$积分为1的常数。

附录 B　作图技巧

本章我们介绍一些统计作图上的技巧，这些技巧对于数据分析来说也许没有显著的作用，但它们可以帮我们进一步调整、组织好我们的图形输出。本章的内容包括：数学公式的表示、一页多图的方法、离散变量的散点图示和各种图形设备的使用方法。

B.1　添加数学公式

由于统计理论中经常需要用到数学符号，所以向统计图形中添加一些数学说明不仅会使得图形看起来更专业，对图形背后的理论也是一种重要补充。

R的**grDevices**包中提供了一系列数学公式的表达符号，例如运算符（加减乘除乘方开方等）、比较符（等号不等号大于小于号等）、微积分符号、希腊字母（大小写α至ω）、上标下标等等。这些数学符号的使用与LaTeX数学公式非常类似，因此如果读者对LaTeX公式比较熟悉的话，用起R中的数学表达式来也会很顺手。

如果想向图中添加数学表达式的文本标签，只需要将文本设置为表达式（expression）的类型即可。图B.1展示了向正态曲线上添加正态分布密度函数表达式的方法，可以看到，表达式中的公式都是LaTeX与R的混合语法。另外，我们也可以设置符号的外形，如斜体、粗体等。详情参见`?plotmath`或者运行代码`demo(plotmath)`观看R提供的数学公式演示。

注意本书中的所有图形均由**tikzDevice**包(Sharpsteen and Bracken 2009)生成，其中的数学公式为原始LaTeX代码，其质量比R自身的数学公式质量高很多，因此图B.1并没有采用`demo(plotmath)`中的写法生成数学公式图上展示的代码为“伪代码”。

```
> # 本代码为“伪代码”，下图由pgfSweave生成
> plot(seq(-3, 3, 0.1), dnorm(seq(-3, 3, 0.1)), type = "l",
+     xlab = "x", ylab = expression(phi(x)))
> text(-3, 0.37, adj = c(0, 1), expression(phi(x) ==
+     frac(1, sqrt(2 * pi)) ~ e^-frac(x^2, 2)), cex = 1.2)
> arrows(-2, 0.27, -1.3, dnorm(-1.3) + 0.02)
> abline(v = qnorm(0.95), lty = 2)
> text(0, dnorm(qnorm(0.95)), expression(integral(phi(x) *
+     dx, -infinity, 1.65) %~~% 0.95))
```

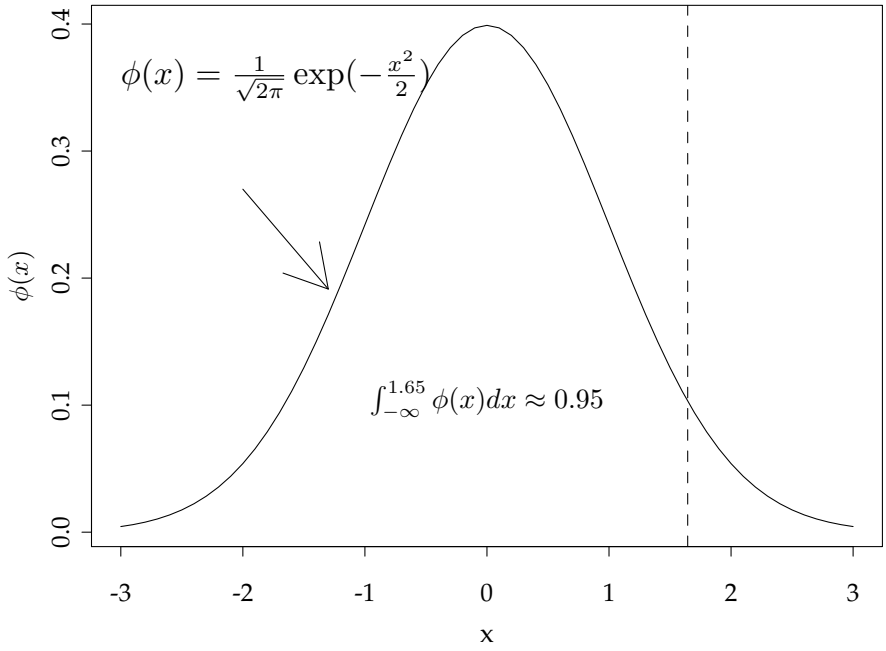


图 *B.1:* 正态分布密度函数公式的表示

B.2 一页多图

有时候我们需要将多幅图形放在同一页图中，以便对这些图形作出对比，或者使图形的排列更加美观。这种情况下，我们至少可以有三种选择：

B.2.1 设置图形参数

在前面3.1小节中我们曾经讲到过mfrow和mfcol两个参数，如果我们在*par()*函数中给这两个参数中的一者提供一个长度为2的向量，那么接下来的图形就会按照这两个参数所设定的行数和列数依次生成图形。本书的图形中有很多用到过这两个参数，如图3.4、5.4等，另外有一些统计图形函数也利用了这两个参数设置它们的图形版面，如四瓣图、协同图等。

这两个参数的限制在于它们只能将图形区域拆分为网格状，每一格的长和宽都分别必须相等，而且每一格中必须有一幅图形，不能实现一幅图形占据多格的功能。下面的两个函数则灵活许多。

B.2.2 设置图形版面

R提供了*layout()*函数作为设置图形版面拆分的工具，其用法如下：

```
> usage(layout)

layout(mat, widths = rep(1, ncol(mat)), heights = rep(1,
    nrow(mat)), respect = FALSE)

> usage(layout.show)

layout.show(n = 1)
```

其中mat参数为一个矩阵，提供了作图的顺序以及图形版面的安排；widths和heights提供了各个矩形作图区域的长和宽的比例；respect控制着各图形内的横纵轴刻度长度的比例尺是否一样；n为欲显示的区域的序号。

mat矩阵中的元素为数字1到n，矩阵行列中数字的顺序和图形方格的顺序是一样的。图B.2解释了这种顺序，该图的矩阵为：

```
> matrix(c(1, 2, 1, 3), 2)

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

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

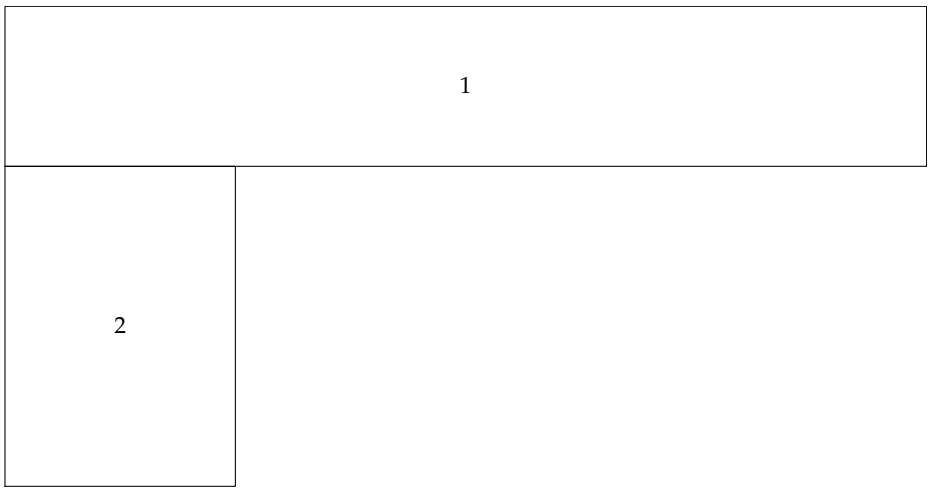


图 *B.2:* 函数layout()的版面设置示意图

由于这种设置，使得第1幅图占据了(1, 1)和(1, 2)的位置，接下来第2、3幅图分别在(2, 1)和(2, 2)的位置；加上长度和宽度的设置，便产生了图B.2的效果。

前面图5.5和5.19曾经使用该函数设置了图形版面，使得不同方格中的图形长宽不一样。图B.3用*layout()*安排展示了二元变量的边际分布以及回归直线。

B.2.3 拆分设备屏幕

R中还有另外一种拆分屏幕的方法，即*split.screen()*。这种方法比前两种方法更灵活，它不仅可以像前两种方法一样设定将作图区域拆分为若干行列，也可以随意指定作图区域在屏幕上的位置。该函数及相关函数用法如下：

```
> usage(split.screen)

split.screen(figs, screen, erase = TRUE)
```

```
> usage(screen)
```

```
> x = pmin(3, pmax(-3, stats::rnorm(50)))
> y = pmin(3, pmax(-3, x + runif(50, -1, 1)))
> xhist = hist(x, breaks = seq(-3, 3, 0.5), plot = FALSE)
> yhist = hist(y, breaks = seq(-3, 3, 0.5), plot = FALSE)
> top = max(c(xhist$counts, yhist$counts))
> layout(matrix(c(2, 0, 1, 3), 2, 2, byrow = TRUE),
+     c(3, 1), c(1, 3))
> par(mar = c(2, 2, 1, 1))
> plot(x, y, xlim = c(-3, 3), ylim = c(-3, 3), ann = FALSE)
> abline(lm(y ~ x))
> par(mar = c(0, 2, 1, 1))
> barplot(xhist$counts, axes = FALSE, ylim = c(0, top),
+     space = 0)
> par(mar = c(2, 0, 1, 1))
> barplot(yhist$counts, axes = FALSE, xlim = c(0, top),
+     space = 0, horiz = TRUE)
```

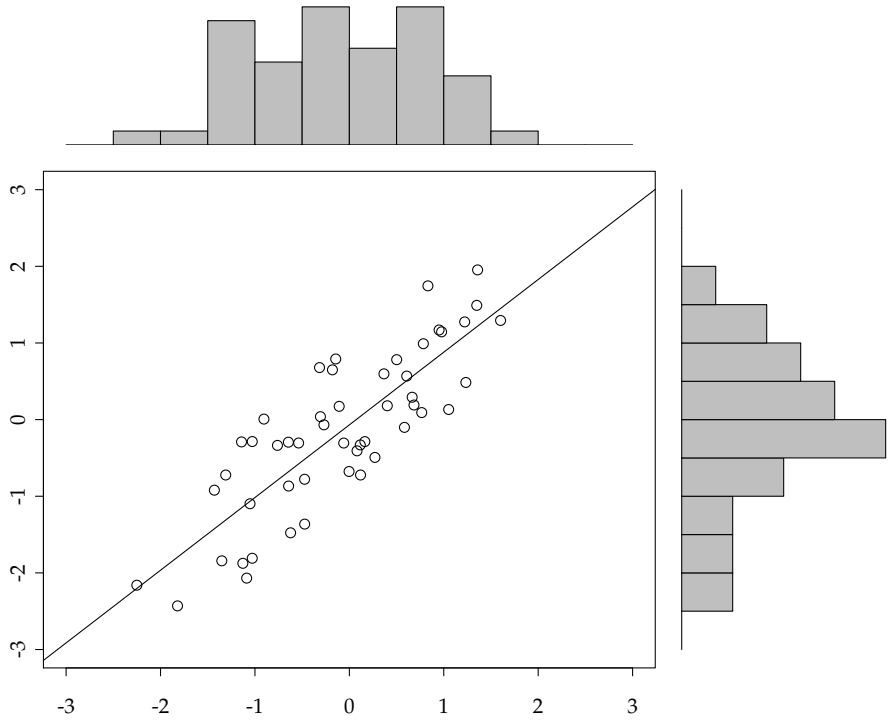


图 *B.3:* 回归模型中边际分布的展示：左下方图中展示了散点图和回归直线，上方和右方的直方图分别展示了自变量和因变量的密度分布。

```
screen(n = cur.screen, new = TRUE)
```

```
> usage(erase.screen)
```

```
erase.screen(n = cur.screen)
```

```
> usage(close.screen)
```

```
close.screen(n, all.screens = FALSE)
```

拆分后的屏幕由若干个区域构成，每个区域有一个编号，即screen，我们可以用函数*screen()*指定要作图的区域号，或者用*erase.screen()*擦除该区域的图形，而*split.screen()*的用法主要由figs参数控制，该参数既可以取值为一个长度为2的向量（指定行列的数目），也可以是一个4列的数值矩阵，制定图形区域的坐标位置，后一种用法比较灵活，它可以将图形作在屏幕的任意位置上，这里的4列矩阵分别给定区域横坐标的左和右以及纵坐标的下和上的位置，即给定了区域左下角和右上角的坐标，这样就可以划分出一块矩形作图区域来。注意这里的坐标值应该在[0, 1]范围内，整个屏幕左下角坐标为(0, 0)，右上角坐标为(1, 1)。

图B.4给出了用矩阵指定作图区域位置的示例，该矩阵的取值为：

```
> matrix(c(0, 0.1, 0.4, 0.3, 0.5, 0.8, 0.9, 1, 0, 0.2,
+     0.3, 0.5, 0.4, 0.7, 0.8, 1), 4, 4)
```

```
     [,1] [,2] [,3] [,4]
[1,]  0.0  0.5  0.0  0.4
[2,]  0.1  0.8  0.2  0.7
[3,]  0.4  0.9  0.3  0.8
[4,]  0.3  1.0  0.5  1.0
```

矩阵一共四行，因此制定了四个屏幕作图区域，四列给定了区域的位置，例如第1个区域的位置在点(0.0, 0.0)与点(0.5, 0.4)之间。该示例中，整个屏幕中划分出了4块有重叠的区域，并分别画出了4幅散点图。

拆分屏幕区域方法的灵活性还在于它可以在拆分的区域中继续拆分（类似于“递归”的做法），而前两节中提到的办法是无法做到这一点的，因此三种方法中这种方法的功能是最强大的，但大多数情况下我们其实用不着如此灵活的定制方法，网格式拆分已经足够使用。

```
> split.screen(matrix(c(0, 0.1, 0.4, 0.3, 0.5, 0.8,
+     0.9, 1, 0, 0.2, 0.3, 0.5, 0.4, 0.7, 0.8, 1),
+     4, 4))
> for (i in 1:4) {
+     screen(i)
+     par(mar = c(0, 0, 0, 0), mgp = c(0, 0, 0), cex.axis = 0.7)
+     plot(sort(runif(30)), sort(runif(30)), col = i,
+         pch = c(19, 21, 22, 24)[i], ann = FALSE,
+         axes = FALSE)
+     box(col = "gray")
+     axis(1, tcl = 0.3, labels = NA)
+     axis(2, tcl = 0.3, labels = NA)
+ }
```

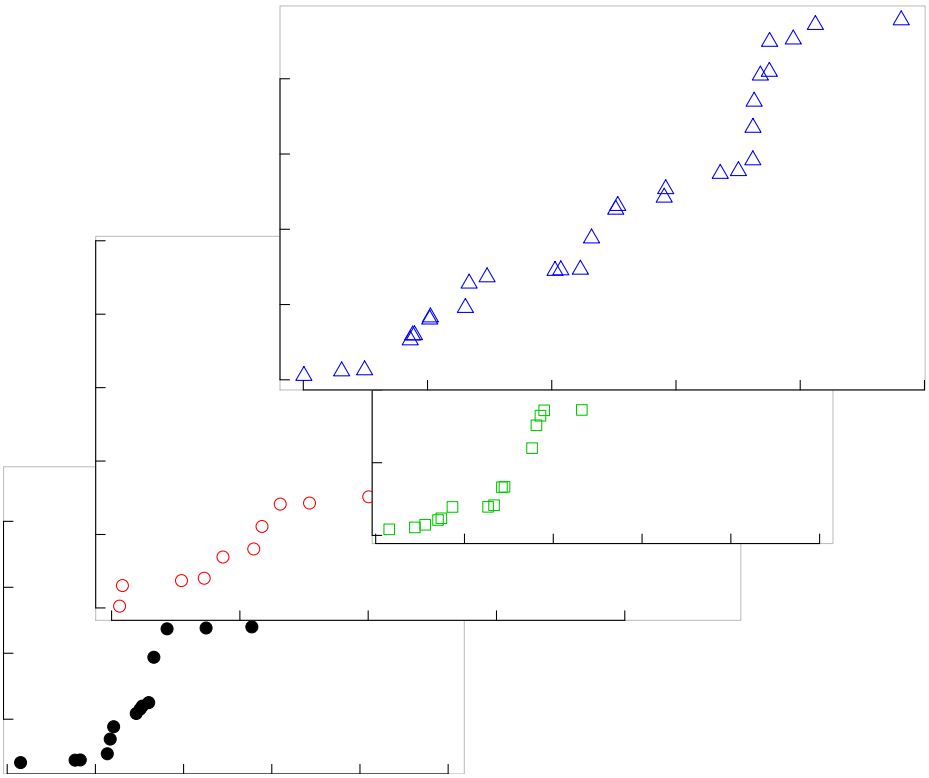


图 *B.4:* 拆分作图设备屏幕区域的示例：本图展示了如何用一个矩阵参数控制作图区域在屏幕上的的位置。

B.3 交互操作

R的图形设备可以支持简单的交互式操作，包括支持对鼠标和键盘输入的响应等，这主要由**graphics**和**grDevices**包中的以下几个函数来完成：

B.3.1 获取鼠标位置的坐标

graphics包中的函数*locator()*可以获取当前鼠标在图形坐标系统中的位置坐标，其用法为：

```
> usage(locator)

locator(n = 512, type = "n", ...)
```

当我们在图形窗口中创建了一幅图形后，我们可以调用该函数并通过点击鼠标获得坐标。参数n表示鼠标点击的次数，type为点击鼠标之后生成的图形类型，可以边点鼠标边画点或画线，后面的参数为一些图形参数，设定点或线的样式。

该函数在点击鼠标事件结束之后会返回一个包含坐标数据的列表，列表中x和y分别表示横坐标和纵坐标的位置。如下例：

```
> plot(1)
> # 任意点击三下鼠标
> locator(3)
> # 返回坐标（结果取决于用户点击的位置）
> # $x
> # [1] 0.6121417 0.8046955 1.2561452
> # $y
> # [1] 0.9562884 0.8710420 1.1648702
```

借助*locator()*返回的坐标数据，我们可以更方便地向图中添加一些图形元素，尤其是图例。因为R的图形设备大多都不支持图形元素的鼠标拖拽，所以事先使用*locator()*在图上“探探路”对画图还是很有帮助的。

B.3.2 识别鼠标附近的数据

graphics包中的函数*identify()*可以通过鼠标点击一幅散点图识别鼠标周围的数据点，并且可以给辨识出的数据添加标签，其默认用法如下：

```
> usage(identify, "default")
```

```
identify(x, y = NULL, labels = seq_along(x),
    pos = FALSE, n = length(x), plot = TRUE, atpen = FALSE,
    offset = 0.5, tolerance = 0.25, ...)
```

x和y给出散点图的原始数据，以便鼠标位置坐标与原始数据进行距离匹配，labels为数据的标签，默认用数据的序号1、2、3……。

当数据的散点图呈现出异常现象时，如存在离群点等等，我们可以很方便地通过*identify()*函数找出该数据的名称或者序号。

B.3.3 响应鼠标键盘的动作

grDevices包中的函数*getGraphicsEvent()*则提供了更灵活的交互，它可以捕获三种鼠标事件（鼠标按下、鼠标移动和鼠标弹起）和一种键盘事件（键盘输入）。用法如下：

```
> usage(getGraphicsEvent)

getGraphicsEvent(prompt = "Waiting for input",
    onMouseDown = NULL, onMouseMove = NULL, onMouseUp = NULL,
    onKeybd = NULL)
```

后面四个参数分别定义了鼠标和键盘事件所对应的行为（通过给定函数实现），具体解释和示例请参见其帮助文件，这里我们只是给出一个例子说明。图B.5演示了鼠标移动的效果：我们在黑色背景的窗口中画了一批数据点，然后通过鼠标的移动在鼠标周围生成一个矩形框，框内的点变成黄色且放大的样式，而框外的点为红色的小点。随着鼠标的移动，矩形框也会在屏幕上移动，从而会框住不同的点。

事实上当今已经有很多类似的交互式图形系统，例如GGobi系统(Cook and Swayne 2007)、Java的图形系统、OpenGL等，R中也有相应的基于这些系统的函数包如**rggobi**(Temple Lang *et al.* 2008)、**iplots**(Urbanek and Wichtrey 2008)、**rgl**(Adler and Murdoch 2008)等；感兴趣的读者可以去研究这些图形系统以及函数包。

B.4 分类变量散点图示

我们知道，因为分类变量只取有限的几个值，所以两个分类变量之间的散点图通常只是若干个网格点，而这些点本身并不能反映出该位置上真

```
> xx = runif(100)
> yy = runif(100)
> mousemove <- function(buttons, x, y) {
+     r = 0.2
+     idx = (x - r < xx & xx < x + r) & (y - r < yy &
+         yy < y + r)
+     plot(xx, yy, type = "n")
+     rect(-1, -1, 2, 2, col = "black")
+     # 鼠标周围画矩形
+     rect(x - r, y - r, x + r, y + r, border = "yellow",
+         lty = 2)
+     points(xx[idx], yy[idx], col = "yellow", cex = 2)
+     points(xx[!idx], yy[!idx], col = "red")
+     NULL
+ }
> mousedown <- function(buttons, x, y) {
+     "Done"
+ }
> par(mar = rep(0, 4), pch = 20)
> # plot(xx, yy, type = 'n')
> # getGraphicsEvent('Click mouse to exit', onMouseDown = mousedown,
> #     onMouseMove = mousemove)
> mousemove(, 0.2, 0.3)
> arrows(0.22, 0.27, 0.18, 0.33, 0.15, lwd = 4, col = "white")
```

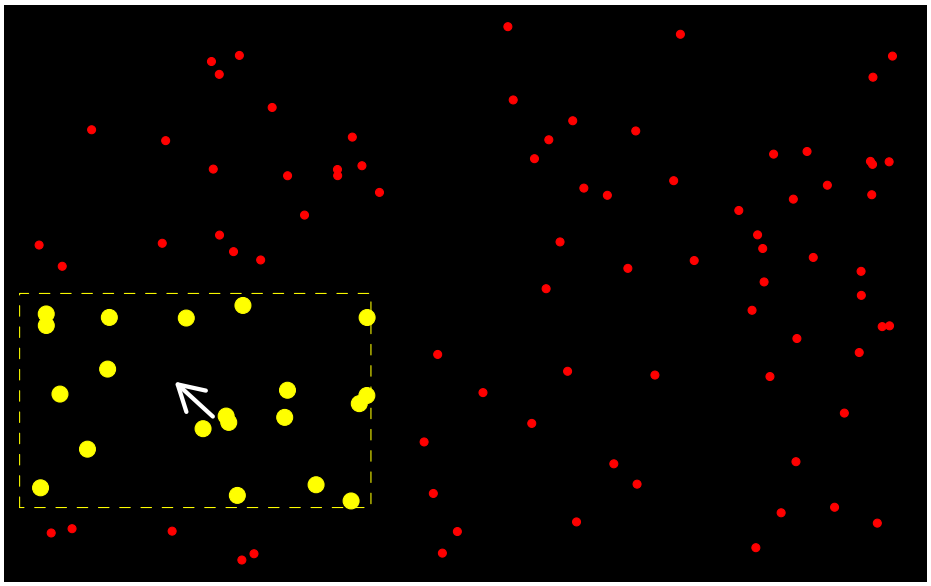


图 B.5: 鼠标在图形窗口中移动的效果图

```
> par(mfrow = c(2, 2), mar = c(2.5, 3, 2, 0.1), pch = 20,
+     mgp = c(1.5, 0.5, 0), cex.main = 1)
> x = sample(rep(1:2, c(12, 18)))
> y = rep(1:2, c(18, 12))
> plot(x, y, main = "(1) 原始散点图", xlim = c(0.8,
+     2.2), ylim = c(0.8, 2.2))
> plot(jitter(x), jitter(y), main = "(2) 随机打散后的散点图")
> points(x, y, cex = 3)
> sunflowerplot(x, y, main = "(3) 向日葵散点图",
+     xlim = c(0.8, 2.2), ylim = c(0.8, 2.2))
> mosaicplot(table(x, y), main = "(4) 马塞克图")
```

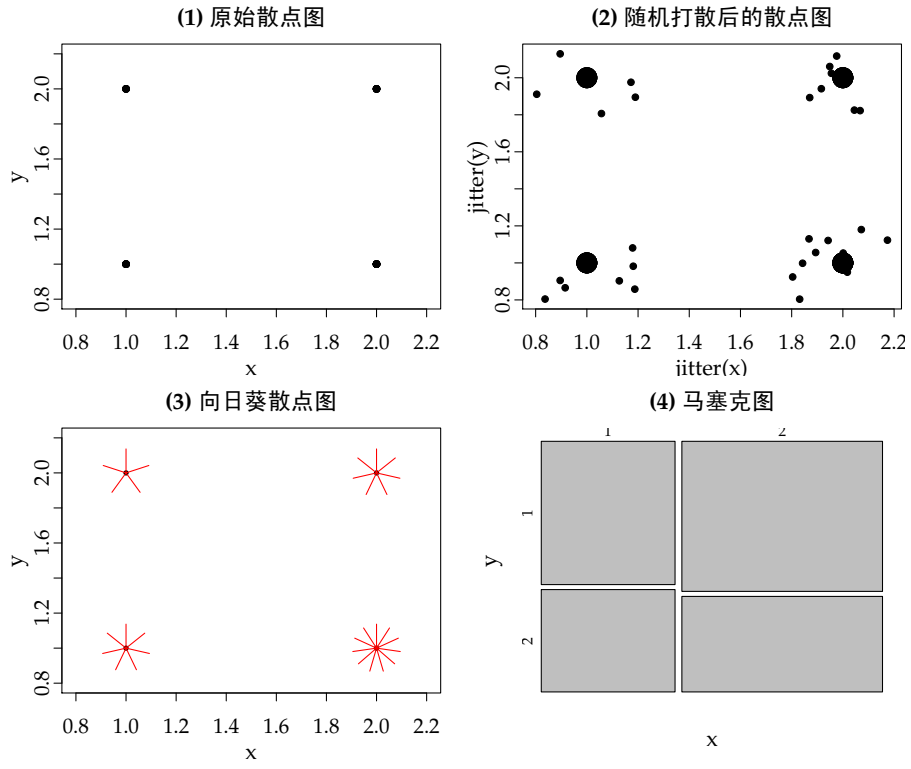


图 *B.6:* 分类变量的散点图示方法示例：原始散点图、打散方法、向日葵散点图和马赛克图。

正的频数。我们在第五章中提到过一些分类变量的图示方法，包括关联图（5.6节）、四瓣图（5.11节）和马赛克图（5.13节）等，不过它们都不是最直接的散点图，而是将频数表达在其它图形元素中。这里我们另外介绍两种散点图方法：向日葵散点图和随机打散方法。

B.4.1 向日葵散点图

关于向日葵散点图，在5.20小节中已经有详细介绍，这里我们再次强调一下它在展示分类变量散点图上的功效。如5.20小节中讲到的，向日葵散点图用向日葵的花瓣表示该处有多少个重复的数据点，而分类变量的散点图大多数情况下都会有重叠的数据点，因此分类变量尤其适合用向日葵散点图来表示。图B.6(3)给出了一个用向日葵散点图表示分类变量的示例。

B.4.2 随机打散方法

由于分类变量散点图的关键问题是重叠问题，因此我们不妨将重叠的数据稍微“打散”一些，然后再作散点图。关于打散方法，我们曾经在5.19小节中用到过，即*jitter()*函数。注意打散过的散点图不能严格按照点的坐标来解读，而是应该按聚集在一处的点的数目来解读频数。图B.6(2)给出了一个打散之后的分类变量散点图示例。

B.5 图形设备

利用**grDevices**包中的若干图形设备，我们可以将R的图形输出为各种格式的文件，包括位图文件（BMP、JPEG、PNG、TIFF）和矢量图文件（PDF、EPS）以及TEX或LATEX文件。本书中除了第一章中的历史图形以外，其它大部分图形都是使用**tikzDevice**包(Sharpsteen and Bracken 2009)中的*tikz()*图形设备生成的（其本质是LATEX）。

基本的图形设备函数有位图设备*bmp()*、*jpeg()*、*png()*和*tiff()*，以及矢量图设备*postscript()*和*pdf()*，打开图形设备之后，所有的R图形都会被生成在该图形设备中，而不会再在窗口中显示，直到图形设备被关闭。详细信息请读者自行查阅相应的帮助文件。

注意位图设备可以支持在图形中使用中文或其它CJK字符，但是在矢量图设备中使用中文字符时则需要设定字体族参数family，否则中文不会被

显示出来（例如简体中文应该用`pdf(family = 'GB1')`）。关于非标准字符在图形设备中的使用，请参考Murrell and Ripley (2006)。

最后补充关于图形的一点基础知识：位图文件的图形是由一个个像素点构成的，因此放大之后会变成晶格状从而不太清晰，而矢量图是由内部的数值矢量构成，这些矢量仅仅定义图形元素的始末位置以及其它属性，放大之后清晰度不变。例如一条直线在位图中由若干个点组成，而在矢量图中则是由两个点构成（给定起点和终点），图形放大之后位图的点之间可能会出现空隙，而矢量图随着放大会自动填充两点之间的空隙。为了得到高质量的打印输出，大多数情况下我们建议使用矢量图。

附录 C　统计动画

作者的个人兴趣之一是统计学动画，这里也简要介绍一下动画的基本原理以及它与统计学理论的内在联系。animation包(Xie 2009)利用R基础图形系统生成了一系列统计学动画；从这个程序包的展示中，读者可以更加深刻认识到对图形元素的控制是一门具有广泛意义的艺术。

附录 D　本书R包

为了配合本书的写作，作者特意编写了一个R包名为**MSG**（Modern Statistical Graphics的缩写），该包目前可从作者主页下载，在本书完成后也会发布到CRAN供读者下载。这里简要介绍一下它包含的函数和数据。

D.1　函数说明

usage() 显示一个函数的用法。该函数可以根据指定的函数名称获取函数的参数列表，并将其以整齐的方式打印出来，如：

```
> # plot函数的用法
> usage(plot)

plot(x, y, ...)
```

```
> # plot的默认用法，即plot.default
> usage(plot, "default")

plot(x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
    log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
    ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
    panel.last = NULL, asp = NA, ...)
```

```
> # plot在画经验分布函数的时候的用法，即plot.ecdf
> usage(plot, "ecdf")

plot(x, ..., ylab = "Fn(x)", verticals = FALSE,
    col.01line = "gray70", pch = 19)
```

```
> # usage自身的用法
> usage(usage)

usage(FUN, class = NULL)
```

D.2 数据说明

PlantCounts 植物数目与海拔高度的数据，共两列，记录了某一海拔高度上植物数目。

参考文献

谢益辉(2008). “统计图形在数据分析中的应用.” In 张波(ed.), 统计学评论. 中国财政经济出版社.

Adler D, Murdoch D (2008). *rgl: 3D visualization device system (OpenGL)*. R package version 0.80, URL `http://rgl.neoscientists.org`.

Bates D, Maechler M (2010). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 0.999375-37, URL `http://CRAN.R-project.org/package=Matrix`.

Becker RA, Chambers JM, Wilks AR (1988). *The New S Language*. Wadsworth & Brooks/Cole.

Chambers JM, Cleveland WS, Kleiner B, Tukey PA (1983). *Graphical Methods for Data Analysis*. Wadsworth & Brooks/Cole.

Cleveland WS (1985). *The Elements of Graphing Data*. Monterey, CA: Wadsworth.

Cleveland WS (1993). *Visualizing Data*. Hobart Press.

Cook D, Swayne DF (2007). *Interactive and Dynamic Graphics for Data Analysis With R and GGobi*. Springer. ISBN 978-0-387-71761-6.

Fox J, with contributions from Liviu Andronic, Ash M, Boye T, Calza S, Chang A, Grosjean P, Heiberger R, Kerns GJ, Lancelot R, Lesnoff M, Ligges U, Messad S, Maechler M, Muenchen R, Murdoch D, Neuwirth E, Putler D, Ripley B, Ristic M, , Wolf P (2009). *Rcmdr: R Commander*. R package version 1.5-4, URL `http://CRAN.R-project.org/package=Rcmdr`.

Friendly M (2008). *Milestones in the history of thematic cartography, statistical graphics, and data visualization*. URL `http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf`.

Hornik K (2009). "The R FAQ." ISBN 3-900051-08-9, URL `http://CRAN.R-project.org/doc/FAQ/R-FAQ.html`.

Ihaka R, Gentleman R (1996). "R: A Language for Data Analysis and Graphics." *Journal of Computational and Graphical Statistics*, **5**(3), 299–314. ISSN 10618600.

Koenker R, Bassett G (1978). "Regression quantiles." *Econometrica*, **46**, 33–50.

McGill R, Tukey JW, Larsen WA (1978). "Variations of box plots." *The American Statistician*, **32**, 12–16.

Murrell P, Ripley B (2006). "Non-standard fonts in PostScript and PDF graphics." *R News*, **6**(2), 41–47. URL `http://cran.r-project.org/doc/Rnews/Rnews_2006-2.pdf`.

Nightingale F (1858). "Notes on Matters Affecting the Health, Efficiency, and Hospital Administration of the British Army." *Technical report*.

Playfair W (1786). *The Commercial and Political Atlas: Representing, by Means of Stained Copper-Plate Charts, the Progress of the Commerce, Revenues, Expenditure and Debts of England during the Whole of the Eighteenth Century*.

Playfair W (1801). *The statistical breviary*. London: T. Bensley.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org`.

Sarkar D (2010). *lattice: Lattice Graphics*. R package version 0.18-3, URL `http://CRAN.R-project.org/package=lattice`.

Scott DW (1992). *Multivariate Density Estimation. Theory, Practice and Visualization*. New York: Wiley.

Sharpsteen C, Bracken C (2009). *tikzDevice: A Device for R Graphics Output in PGF/TikZ Format*. R package version 0.4.8, URL `http://CRAN.R-project.org/package=tikzDevice`.

Temple Lang D, Swayne D, Wickham H, Lawrence M (2008). *rggobi: Interface between R and GGobi*. R package version 2.1.9, URL `http://www.ggobi.org/rggobi`.

Tufte ER (1992). *Envisioning Information*. Cheshire, CT, USA: Graphics Press. ISBN 0-961-39211-8.

Tufte ER (2001). *The Visual Display of Quantitative Information*. 2nd edition. Cheshire, CT, USA: Graphics Press. ISBN 0-9613921-4-2.

Urbanek S, Wichtrey T (2008). *iplots: iPlots - interactive graphics for R*. R package version 1.1-2, URL `http://www.iPlots.org/`.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer. ISBN 0-387-95457-0.

Wainer H, Thissen D (1981). "Graphical Data Analysis." *Annual Review of Psychology*, **32**(1), 191–241.

Wickham H (2009). *ggplot2: elegant graphics for data analysis*. Springer New York. ISBN 978-0-387-98140-6. URL `http://had.co.nz/ggplot2/book`.

Wilkinson L (2005). *The Grammar of Graphics*. 2nd edition. Springer.

Xie Y (2009). *animation: Demonstrate Animations in Statistics*. R package version 1.1-0, URL `http://animation.yihui.name`.

Xie Y (2010). *MSG: Modern Statistical Graphics*. R package version 0.1-0, URL `http://yihui.name/cn/publication`.

索引