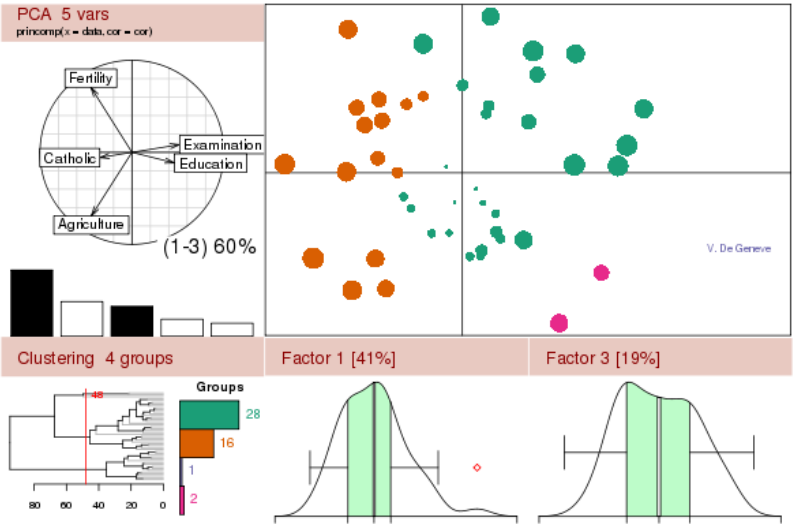
西南财经大学课程讲义

R 语言入门

司亚卿


PCA 5 vars
princomp(x = data, cor = cor)
Fertility
Examination
Catholic
Education
Agriculture
(1-3) 60%
V. De Geneve
Clustering 4 groups
Factor 1 [41%]
Factor 3 [19%]
Groups
28
16
1
2
80
60
40
20
0

目录

第 一 章　一门最值得学习的统计语言

1.1　R 的特点

此部分主要引用一个网络架构师的博客http://blog.fens.me/r-ideal/

(i) R 的基因

R 是统计学家发明的语言,天生具有统计的基因。

从我开始学习 R 语言,我就开始了知识的跨界思考。统计基于概率论,概率论又基于数学,用计算机的方式编程,解决某个领域的实际问题。简单一算,4 个学科知识的交集,决定着我们解决问题的能力。统计的基因,让 R 语言与众不同!

(ii) R 的发展

R 一直在小众领域成长着,最早也只有统计学家在用,主要用 R 来代替 SAS 做统计计算。时代在进步,随着大数据的爆发,R 终于在这一波浪潮中,被工业界所发现。然后,有越来越多的工程背景的人加入到这个圈子,对 R 计算引擎,R 的性能,R 的各种程序包进行改进和升级,让 R 获得了新生。

我们现在用到的 R 语言软件,已经越来越接近工业软件的标准了。由工

程师推动的 R 的发展速度,远远地超过了由统计学家推动的步伐。随着人们对数据分析要求的进一步增加,R 会以更快的脚步继续发展,将成为免费的、开源的、数据分析软件的代名词。

(iii) R 的应用前景

R 可以做所有 SAS 做的事情。R 应用最热门的领域有:

- 统计分析:包括统计分布,假设检验,统计建模
- 金融分析:量化策略,投资组合,风险控制,时间序列,波动率
- 数据挖掘:数据挖掘算法,数据建模,机器学习
- 互联网:推荐系统,消费预测,社交网络
- 生物信息学:DNA 分析,物种分析
- 生物制药:生存分析,制药过程管理
- 全球地理科学:天气,气候,遥感数据
- 数据可视化:静态图,可交互的动态图,社交图,地图,热图,与各种 Javascript 库的集成

总之 R 有着非常广阔的应用前景，而且 R 也将成为新一代的最有能力创造价值的工具。

(iv) R 的社区和资源

R 的发展，离不开 R 的社区支持。当然，我不得不承认 R 的官方社区，从 Web 页上看起来太简陋了，稍微调整一下 CSS 样式表，都会比现在好看很多。也许这种简单、无修饰也是统计学家的基因吧。

在 R 的社区中，我们可以下载到 R 语言软件，R 的第三方软件包，和 R 的其他支持软件。可以找到开发者论坛，R-Journal 列表，软件包列表，R 语言图书列表，R 用户组等的信息，同其他语言的社区资源一样丰富。

R 是自由软件，开发者可以开发自己的软件包，封装自己的功能，然后在 CRAN 上面发布。截止到 2014 年 2 月，共有 5236 个 R 包在 CRAN 上面发布。

可能很多人会说只有 5236 个包，数量太少了。这是因为 CRAN 是需要提

交申请的,R 语言小组审核,检查后再会发布的出来。而且审核非常严格的,高质量是发布一个新的 R 包基本要求。由于 CRAN 过于严格的审查,让很多的开发者选择在 RForge 上发布,还有些 R 包是基于 Github 发布的,我也在 github 上面发布了自己的 R 包:https://github.com/bsspirit/chinaWeather。

- R 官方地址:http://www.r-project.org/
- R 开发者论坛:http://r.789695.n4.nabble.com/
- CRAN:http://cran.rstudio.com/
- RForge: https://r-forge.r-project.org/

(v) R 的哲学

每种语言都有自己的设计理念和哲学,而我体会的 R 的哲学,就是"静下心做事情"。

R 不需要很长的代码,R 也不需要设计模式。一个函数调用,传几个参数,就能实现一个复杂的统计模型。我们需要思考,用什么模型,传什么参数,

而不是怎么进行程序设计。

我们可能会用 R 实现“从一个数学公式,变成一个统计模型”的过程,我们也可能会考虑“如何让一个分类器结果更准确”,但我们不会思考“时间复杂度是多少,空间复杂度是多少”。

R 的哲学,可以让你把数学和统计学的知识,变成计算模型,这也是 R 的基因所决定的。

(vi) R 的使用者

R 语言早期主要是学术界统计学家在用,在各种不同的领域,包括统计分析,应用数学,计量经济,金融分析,财经分析,人文科学,数据挖掘,人工智能,生物信息学,生物制药,全球地理科学,数据可视化等等。

近些年来,由互联网引发的大数据革命,才让工业界的人,开始认识 R,加入 R。当越来越多的有工程背景的人,加入到 R 语言使用者的队伍后,R 才开始像着全领域发展,逐步实现工业化的要求。

- RevolutionAnalytics 公司的 RHadoop 产品，让 R 可以直接调用 Hadoop 集群资源
- RStudio 公司的 RStudio 产品，给了我们对于编辑软件新的认识
- RMySQL, ROracle, RJDBC 打通了 R 和数据库访问通道
- rmongodb, rredis, RHive, rhbase, RCassandra 打通过 R 和 NoSQL 的访问通道
- Rmpi, snow 打通了单机多核并行计算的通道
- Rserve,rwebsocket 打通了 R 语言的跨平台通信的通道

R 不仅是学术界的语言，更将成为工业界必备的语言。

(vii) R 的思维模式

R 语言让我跳出了原有思维定式。使用 R 语言，我们应该从统计学的角度想问题，而不是计算机的思维模式。

R 语言是直接面向数据的语言。在我们的日常生活中，无论做什么事情都会产生数据，上网有浏览数据，买东西有消费数据，就算什么都不干，也会

受大气 PM2.5 的影响。利用 R 语言,我可以直接分析这些数据。

面向什么业务,就分析什么数据,不需要从产品经理向程序员的角色转换,不需要考虑有什么功能,更不需要考虑程序设计的事。

(viii) R 解决的问题

当数据成为生产资料的时候,R 就是为人们能运用生产资料创造价值的生产工具,R 语言主要解决的是数据的问题。

在很长期的历史时期,人类产生的数据都没有自互联网诞生以来产生的数据多;当 Hadoop 帮助人们解决了大数据存储的问题后,如何发现数据的价值,成为当前最火的话题。R 语言的统计分析能力,就是数据分析最好的工具。

所以,R 要解决的问题,就是大数据时代的问题,是时代赋予的任务。

(ix) R 的不足

前面说了太多 R 的优点了,R 也有很多不足之处。

- R 语言是统计学家编写的软件,并不如软件工程师编写的软件那么健壮。
- R 语言软件的性能,存在一些问题。
- R 语言很自由,语法命名不太规范,需要花时间熟悉。
- R 语言结合了很多数学、概率、统计的基础知识,学起来有一定门槛。

R 的这些不足,都是可以克服的。当有更多的工程背景的人加入的时候,R 语言会比现在更强大,帮助使用者创造更多的价值。

(x) 时代赋予 R 的任务

R 语言是在大数据时代被工业界了解和认识的语言,R 语言被时代赋予了,挖掘数据价值,发现数据规律,创造数据财富的任务。

R 语言也是帮助人们发挥智慧和创造力的最好的生产工具,我们不仅要学好 R 语言,还要用好 R 语言,为社会注入更多的创新的生产力。

所以,通过上面的几节内容所有的文字描述,我认为“R 是最值得学习的编程语言”。不论你还在读书,还是已经工作,掌握 R 语言这个工具,找最适合自己的位置,前途将无限量。

最后总结:R 是最特殊的,R 被赋予了与其他语言不同的使命。R 的基因决定了,R 将成为 2014 年,也可能是以后更长一段时间的主角。

1.2 R 的安装即其它准备工作

1.3 下载安装

R 很强大的,但是与其它软件 SAS、SPSS 或其它软件相比,R 体积很小(小于 50M,对比 SAS 9.2 差不多有 9G),而且 R 完全免费!

从官方网站http://www.r-project.org/可以下载并安装最新的版本。安装之后可以立即使用。R 自带的界面非常简洁,但功能足够齐全。也可以使用其它更友好的程序界面,比如 Rstudio(可以从http://www.rstudio.com/下载安装)

(i) 程序包

当你搜索 R 有很多程序包(library)可以安装。比如,若通过网上搜索到某个函数 RNASeq.Data() 的帮助文件如下图所示,则表明该函数是在一个名叫 MBCluster.Seq 程序包中 若该程序包尚未安装,则需要安装之后才能调用,语法是

```
install.packages("MBCluster.Seq") ## 安装程序包
library(MBCluster) ## 调用程序包
```

有时候会遇到安装权限的问题,无法安装到默认的路径,则可以指定安装路径,比如可以安装在 D 盘的 MyPackages 文件夹中。这时候需要指定具

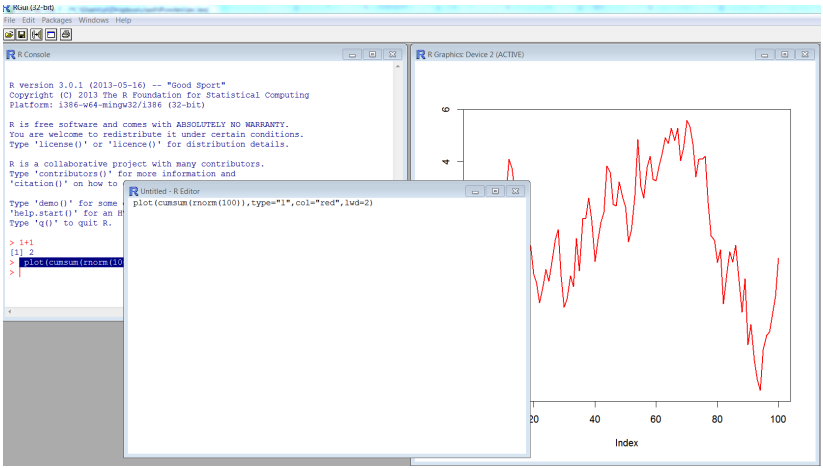


Figure 1.1: R 自带的界面

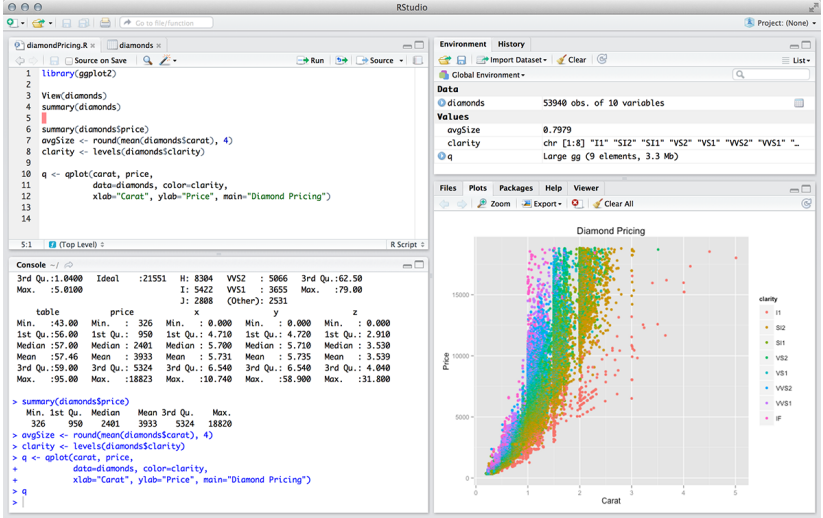


Figure 1.2: Rstudio 界面

RNASeq.Data {MBCluster.Seq}

Standardize RNASeq Data for Clustering

Description

RNASeq.Data is used to collect RNA-Seq data that need to be clustered.

Usage

```
RNASeq.Data(Count, Normalizer=NULL, Treatment,GeneID=NULL)
```

Arguments

Count	a GxP matrix storing the numbers of reads mapped to G genes in P samples. Non-integer

体的位置才能调用

```
install.packages("MBCluster.Seq",lib="D:/MyPackages/") ## 安装程序包
library("MBCluster.Seq",lib.loc="D:/MyPackages/")
```

有时候,某个特殊的程序包在 CRAN 服务器上找不到,这时候可以将程序文件下载到电脑上再安装。建议下载.tar.gz 格式的安装文件,因为这种格式支持 Windows,OS,Linux 等各种系统。

```
install.packages("MBCluster.Seq",lib="D:/MyPackages/") ## 安装程序包
library(pkgs="D:/MBCluster.Seq_1.0_tar.gz",lib.loc="D:/MyPackages/",sou
```

(ii) 设置路径

当前 R 的工作路径 (working directory) 是 R 默认寻找程序或文件的地址,可以通过 getwd() 来查看

```
> getwd()
```

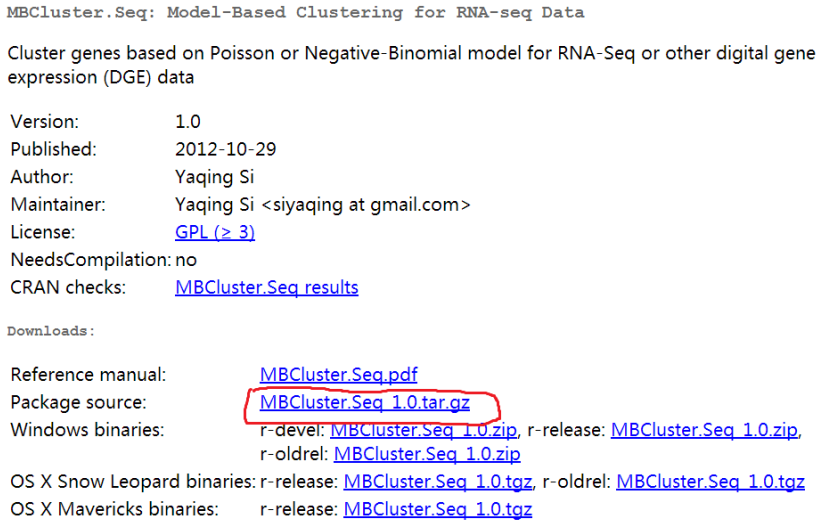


Figure 1.3: 网上搜到的 MBCluster.Seq 软件包的安装文件下载地址

```
[1] "C:/Documents"
```

程序表明当前工作路径是 C 盘的 Documents 文件夹。

有时候为了方便操作，需要改变 R 的默认工作路径，比如在 D:/data 中，最好重新通过 setwd() 设定路径

```
> setwd("D:/data")
> getwd()
[1] "D:/data"
```

上面的 getwd() 函数表明当前的工作目录已经发生改变。

1.4 从入门到精通

(i) R 高手秘籍

R 语言作为统计学一门语言,一直在小众领域闪耀着光芒。直到大数据的爆发,R 语言变成了一门炙手可热的数据分析的利器。随着越来越多的工程背景的人的加入,R 语言的社区在迅速扩大成长。现在已不仅仅是统计领域,教育,银行,电商,互联网. 都在使用 R 语言。要成为有理想的极客(geek),你不能停留在语法上,要掌握牢固的数学,概率,统计知识,同时还要有创新精神,把 R 语言发挥到各个领域。比如有个家伙就用 R 做了一个贪吃蛇游戏 但是你必须从一个菜鸟做起。

有人说 R 的门槛太高,初学者很容易放弃。一派胡言! 实际上,即使你是一个会用键盘的小学生,你就可以使用 R。比如你在运行框中敲入 1+1,就可以得到答案 2。这时候,R 可以当成一个功能强大的计算器。

```
> 1+1
```

Figure 1.4: 一个用 R 语言写的贪吃蛇程序

```
[1] 2
```

到了大学,有了微积分和线性代数的知识,你就可以用 R 做复杂的计算了;然后只要再具有初步的概率论和数理统计知识,你就可以做各种各样统计分析了。

所以你任何时候都可以学习 R。有多大的基础,就可以达到相应的级别。就像学习大多数软件(比如 SAS,Matlab)一样,下面是一条学习 R 的金玉良言:

学习秘诀第一条:不要试图把 R 学透,因为永远也不可能!你要做的是,遇到一个实际的问题,首先问问自己可不可以用 R 完成,然后去尝试。

一下子学太多,会让你绝望,所以要循序渐进。使用一个星期,你就熟悉它的界面了;使用一个月,你就熟悉它的语法了;使用一年,你就熟悉熟悉它的功能了;使用十年,你就是 R 高手了;使用一辈子,你就是 R 的骨灰级高手了!

现在能做统计活的软件很多,SAS,SPSS,Eviews,Matlab……,但是我建议你们优先学会 R。向你们推荐 R 不是出于私心,不是因为这门课程教 R 语

言(实际上本来是 SAS),而仅仅是因为它的好。希望你们能够遵循下面的忠告:

学习秘诀第二条:有意识地只用 R,最好对它产生依赖性。要拒绝其它软件的诱惑。

SAS 太贵了,你买不起;太难装了,你不想用;太不灵活了,用着烦! SPSS, E-views 太容易了,别的专业的同学都会怎么能体现出你是统计高手?! Matlab 的统计功能太不全了,有时候用不上! 其它的? 你也不会!

学习秘诀第三条:多多使用免费资源 买几本厚厚的参考书,碰到问题翻一翻,这种学习方法已经过时了。R 是一种免费的、开放性的编程语言,全世界有数以百万计的 R 高手分享他们的技巧、成果。除非你专门学习某个领域的技巧,比如 Hadoop 大数据、生物信息、机器学习、金融分析等领域,你应该使用这些资源是最有效率的途径是利用各种共享资源。

最常用的方法是用 R 自带的 help 文档。用 help() 或? 命令就可以立即看到这个帮助文档,比如若要查询对数函数 log() 的用法,可以直接输入下面的任意一条命令:

```
> ?log
> help(log)
```

R 的帮助文件是英文的，但一定要克服懒惰心理，认真看下去，其实非常好懂。看点有一般有几条：红色的需要重点关注。

- 描述 (Description): 描述某个命令/函数的功能
- 用法(Usage)：基本用法, 命令的格式。
- 参数(Arguments)：所需的参数，规定了参数的数据类型、格式等各种详细要求。
- 细节 (Details)：其它补充细节
- 取值(Values:) 命令/函数的输出的结果
- 方法 (S4/3 Methods)：R 如何运行这个命令/函数
- 来源 (Sources)：源代码，用何种语言写成等信息。
- 参考 (References)：参考文献
- 其它 (See also)：其它相似的函数
- 例子(Examples)：一些有用的例子

如果你觉得 help 文档不够具体，或者看不懂，就可以使用丰富的网络资

log {base}

Logarithms and Ex

Description 描述

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `lo`

`log1p(x)` computes *log(1+x)* accurately also for |x| << *1* (and less accurately when *x is approximately -1*).

`exp` computes the exponential function.

`expm1(x)` computes *exp(x) - 1* accurately also for |x| << *1*.

Usage 用法

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

Arguments 参数

`x`	a numeric or complex vector.
`base`	a positive or complex number: the base with respect to which logarithms are computed. Defaults to e=`exp(1)`.

Details 细节

All except `logb` are generic functions: methods can be defined for them individually or via the `Math` group generic.

`log10` and `log2` are only convenience wrappers, but logs to bases 10 and 2 (whether computed *via* `log` or the wrappers) will be computed more efficiently and accurately where supp

`logb` is a wrapper for `log` for compatibility with S. If (S3 or S4) methods are set for `log` they will be dispatched. Do not set S4 methods on `logb` itself.

All except `log` are primitive functions.

Value 取值

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf`, and `log(x)` for negative values of `x` is `NaN`. `exp(-Inf)` is `0`.

For complex inputs to the log functions, the value is a complex number with imaginary part in the range *[-pi, pi]*: which end of the range is used might be platform-specific.

S4 methods 方法

`exp`, `expm1`, `log`, `log10`, `log2` and `log1p` are S4 generic and are members of the `Math` group generic.

Note that this means that the S4 generic for `log` has a signature with only one argument, `x`, but that `base` can be passed to methods (but will not be used for method selection). On th

Source 来源

`log1p` and `expm1` may be taken from the operating system, but if not available there are based on the Fortran subroutine `dlnrel` by W. Fullerton of Los Alamos Scientific Laborator

References 参考

源。不幸的是，大多数资源仍然是英文的，百度不知道。这时候就要用英文搜索，可惜最好用的 google 得罪了中国的 GFW(Great Firewall)，所以你是没法用的。但微软的必应还可以用，打开www.bing.com, 输入适当的关键词，比如R: log function，就可以找到很多关于 log 函数用法的例子。其它资源也可以用相似的方法搜到，不要忘了关键词“R”。

网上还有大量的博客、论坛等学习资源，有时间多看看别人的例子，加以模仿，就熟悉了。

1.5 关于这门课

课程面向初学者。设定你没有任何 R 语言的基础。需要有微积分、线性代数、一些概率论、数理统计、线性模型的知识。主要内容拟包括：

- 矩阵、向量的运算
- 列表、数据框、因子等数据类型
- 数据的读写、存储、提取

- 自定义函数、条件、循环程序
- 画图: 点、线、方块、颜色、大小、字体等设定。
- 描述性统计量、常见的统计方法、图形
- 随机分布、随机数、概率值
- 简单统计检验、线性模型

这门课不指定教材。每个星期都有作业。暂时决定没有笔试,但有上机考试。

请随时登录课程网页www.sssidea.org/R/关注讲义、作业、通知等信息

第 二 章　向量、矩阵、函数

向量是 R 语言最基本的数据类型。R 的计算是基于向量的，正是因为如此，R 的运算才具有很高的效率。

2.1 向量、矩阵

(i) 变量和赋值

R 的变量不需要专门定义。变量的名字可以直接用由字母和数字(数字不能作为开头)组成的序列表示,因此 x,a3,A3,ab5c 都是合法的变量名字。字母分大小写,因此 x 和 X 是不同的变量。

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" 组成一个类似箭头的符合。"箭头"指向被表达式赋值的对象。因此下面的两个命令都表示把 1 这个数值赋予 x 这个变量

```
> x <- 1
> 1 -> x
```

在许多(大多数)情况下,赋值符号 <− 可以用等号 = 代替,也可以用函数 assign() 实现。下面的命令和前面的赋值命令等价:

```
> x = 1
> assign("x",1)
```

但是作为初学者,只用等号 = 就可以了。

若要显示某个变量的值,只需要输入变量的名称就可以了,比如上面我们已经创建了某个变量 x,但忘记了或只是想显示它的值,就可以键入

```
> x
[1] 1
```

于是就知道 x 的值是 1

(ii) 函数 c()

R 在已经命名的数据结构(data structure)上起作用的。其中,最简单的结构就是由一串有序数值构成的数值向量(vector)。假如我们要创建一个含

有五个数值的向量 x,且这五个值分别为 10.4,5.6,3.1,6.4 和 21.7,则 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,而它返回的值则是一个把这些参数首尾相连形成的向量。

(iii) 函数:

在 R 环境里面,单个的数值也是被看作长度为 1 的向量。比如

R 有一系列产生常用数列的工具。如 1:30 等价于向量 c(1,2,3,4,5)。5:1 形式的句法(construction)可用来产生一个逆向的数列。比较

```
> x=1:5
> x
[1] 1 2 3 4 5
```

```
> y=5:1
> y
[1] 5 4 3 2 1
```

(iv) 函数 seq()

函数 seq() 是数列 (sequence) 生成中最为常用的工具。它有五个参数，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，则和冒号运算符的效果完全一样了。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参数命名方式给定。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)，seq(to=30, from=1) 同 1:30 完全一样。seq() 随后的两个参数是 by=value 和 length=value；它们分别表示这个数列的步长和长度，二者不需要同时设定。如果二者都没有设定，默认值就是 by=1(步长为 1)。试比较

```
> seq(-2,2)
[1] -2 -1  0  1  2
> seq(2,-2)
[1]  2  1  0 -1 -2
> seq(2,-2,by=-.5)
[1]  2.0  1.5  1.0  0.5  0.0 -0.5 -1.0 -1.5 -2.0
> seq(from=2,to=-2,length=11)
 [1]  2.0  1.6  1.2  0.8  0.4  0.0 -0.4 -0.8 -1.2 -1.6 -2.0
> seq(from=-2,to=2,length=11)
 [1] -2.0 -1.6 -1.2 -0.8 -0.4  0.0  0.4  0.8  1.2  1.6  2.0
```

(v) 函数 rep()

还有一个相关的函数是 rep()。它可以用各种复杂的方式重复 (repeat) 一个对象。最简单的方式是 > 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) 这种方式把 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。

```
> x=1:3
> rep(x,each=5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
> rep(x,times=5)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

(vi) 向量的指标

一个向量有一个或很多值组成,若想知道某些位置的数值,可以用指标符号"方括号"[] 得到. 向量的元素下标取值是以 1 开始, 比如 x 的第 1 个或 2、3 位置上的取值

```
> x=5:10
> x[1]
[1] 5
```

```
> x[2:3]
[1] 6 7
```

x 除去某些位置的取值可以通过在 [] 加上减号 -得到,比如除去 x 的第 3 个元素可以得到第 1,2,4,5,6 位置的分量

```
> x=5:10
> x[-3]
[1] 5  6  8  9 10
```

也可以改变向量某个或某些位置的值,比如将 x 的第 3 个分量的改为 0 可以用如下命令

```
> x=5:10
> x[3]=0
> x
[1]  5  6  0  8  9 10
```

也可以一次取出几个位置的值

```
> i=c(2,4)
> x[i]
[1] 6 8
> x[-i]
[1]  5  0  9 10
```

(vii) 向量的运算

向量可以参加数学运算,规则是每一个分量都单独参加运算。比如

```
> x=1:3
> y=x+1
> y
[1] 2 3 4
> sin(x)
```

```
[1] 0.8414710 0.9092974 0.1411200
> sum(x)
[1] 6
```

常用的数学运算包括

```
+,-,*,/,^,%%,%/%                                  # 四则、同余
<,>,<=,>=,==,!=                                   # 比较、逻辑
log(x),log10(),log2(),exp(),expm1(),log1p()       # 对数、指数
sqrt(),sign(),factorial()                         # 开方、符号、阶乘
cos(),sin(),tan(),acos(),asin(),atan(),atan2()    # 三角函数
cosh(),sinh(),tanh(),acosh(),asinh(),atanh()      # 反三角函数
union(),append(),intersect()                      # 集合
max(),min(),which.max(),which.min()               # 最大值，最小值
identical(),all.equal,setdiff(),setequal()        # 比较
append(),rbind(),cbind()                          # 合并
sum(),prod(),cumsum(),cumprod()                   # 连加、连乘
length(),range(),round()                          # 长度，范围，取整
```

熟练掌握这些基本函数可以显著提高向量运算的效率

(viii) 向量元素的添加及合并

```
> v=c(1,2,3)
> v=c(v,55) #格式为新向量<-(原向量，新元素)
> v
[1]  1  2  3 55
> v1=c(1, 2, 3)
> v2=c(4, 5, 6)
> v3=c(v1,v2)
> v=c(1,2,3,3,3,4)
> append(v,10,after=3) #在第3个向量后面加入10
[1]  1  2  3 10  3  3  4
```

(ix) 向量排序

sort(): 输出排序后的结果;order(): 输出排序后的各个分量位置;rank(): 各个分量的排名;rev(): 将向量逆序重新排列

```
> a=c(3,9,0,12,19)
> sort(a)
[1]  0  3  9 12 19
> order(a)
[1] 3 1 2 4 5
> rank(a)
[1] 2 3 1 4 5
> rev(a)
[1] 19 12  0  9  3
```

2.2 矩阵

(i) 定义一个矩阵

一个矩阵可以通过改变某个向量的维数来定义,比如一个长度为 20 的向量可以变成 4×5 的矩阵

```
> x=1:20
> y=matrix(x,nrow=4,ncol=5)
> y
     [,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
```

可见矩阵是按照列将向量的各个元素依次排列。如果想按照行排列,就可以修改参数 byrow=TRUE。

```
> z=matrix(x,4,5,byrow=TRUE)
> z
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
```

(ii) 合并矩阵

rbind 和 cbind 是两个可以合并矩阵 (或向量) 的函数,分别按照行 (row) 和列 (column), 但合并的时候要保证行数或列数相等。继续上面的例子:

```
> rbind(y,z)
```

```
     [,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
[5,]    1    2    3    4    5
[6,]    6    7    8    9   10
[7,]   11   12   13   14   15
[8,]   16   17   18   19   20
> cbind(y,z)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    5    9   13   17    1    2    3    4     5
[2,]    2    6   10   14   18    6    7    8    9    10
[3,]    3    7   11   15   19   11   12   13   14    15
[4,]    4    8   12   16   20   16   17   18   19    20
```

(iii) 子矩阵

可以使用方括号提取矩阵的元素,y[i,j] 表示矩阵第 i 行第 j 列的那个元素,y[i,] 表示第 i 行,y[,j] 表示第 j 列。若在 i 或 j 前面加上负号 -,则表示除去这些行或列剩下的部分

```
> y[2,4]
[1] 14
> y[,4]
[1] 13 14 15 16
> y[2,]
[1]  2  6 10 14 18
> y[-2,]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    3    7   11   15   19
[3,]    4    8   12   16   20
> y[,-4]
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   17
[2,]    2    6   10   18
[3,]    3    7   11   19
[4,]    4    8   12   20
```

也可以提取一个子矩阵,比如

```
> i=c(1,3)
> j=c(2,3,5)
> y[i,j]
     [,1] [,2] [,3]
[1,]    5    9   17
[2,]    7   11   19
> y[-i,-j]
     [,1] [,2]
[1,]    2   14
[2,]    4   16
```

(iv) 给矩阵行或列起名

可以用 colnames 或 colnames 为矩阵的行或列命名,名字应该是一串数字或字符,注意名字的长度必须与行数或列数相等。

```
> rownames(y)=c("apple","orange","peach","grape")
> colnames(y)=c("yellow","red","blue","green","purple")
> y
       yellow red blue green purple
apple       1   5    9    13     17
orange      2   6   10    14     18
peach       3   7   11    15     19
grape       4   8   12    16     20
```

有了名字,就可以很容易地提取相应的行或列了

```
> y["apple",]
yellow    red   blue  green purple
```

```
     1      5      9     13     17
> y[,c("red","blue")]
       red blue
apple    5    9
orange   6   10
peach    7   11
grape    8   12
```

(v) 矩阵的运算

矩阵可以参加简单的数学运算,和向量一样(矩阵在 R 中实际上就是一个向量),运算的规则是,对简单的运算每一个元素分别参加计算。

```
> z^2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    9   16   25
[2,]   36   49   64   81  100
```

```
[3,]  121  144  169  196  225
[4,]  256  289  324  361  400
```

有了名字,就可以很容易地提取相应的行或列了

```
> y["apple",]
yellow    red   blue  green purple
     1      5      9     13     17
> y[,c("red","blue")]
       red blue
apple    5    9
orange   6   10
peach    7   11
grape    8   12
```

%*是矩阵相乘的运算符,一个矩阵可以左/右乘以另外一个矩阵

```
> m=matrix(1:20,5,4)
```

```
> m%*%z
     [,1] [,2] [,3] [,4] [,5]
[1,]  414  448  482  516  550
[2,]  448  486  524  562  600
[3,]  482  524  566  608  650
[4,]  516  562  608  654  700
[5,]  550  600  650  700  750
> z%*%m
     [,1] [,2] [,3] [,4]
[1,]   55  130  205  280
[2,]  130  330  530  730
[3,]  205  530  855 1180
[4,]  280  730 1180 1630
```

注意,一个长度为 n 的向量被视为一个 nx1 的矩阵,所以矩阵可以右乘一个向量,但不可左乘。另外,注意区分矩阵相乘%*%和一般的乘法 *。对一般的乘法 *,R 先将矩阵变成一个向量,然后依次将两个向量的元素相乘(较短的矩阵会自动重复直到补齐),最后将结果得到的向量变回原来矩阵的

形状。请运行如下代码,并思考其结果。

```
> v=1:5
> z%*%v
     [,1]
[1,]   55
[2,]  130
[3,]  205
[4,]  280
> z*v
     [,1] [,2] [,3] [,4] [,5]
[1,]    1   10   12   12   10
[2,]   12    7   40   36   30
[3,]   33   24   13   70   60
[4,]   64   51   36   19  100
> v%*%z
Error in v %*% z : non-conformable arguments
> v*z
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1   10   12   12   10
[2,]   12    7   40   36   30
[3,]   33   24   13   70   60
[4,]   64   51   36   19  100
```

(vi) 矩阵函数

常用的矩阵函数有如下几个

```
t()                                    ## 矩阵转置
det(),norm()                                ## 行列式、模
solve()                                ## 非奇异方阵的逆矩阵
diag(),upper.tri(),lower.tri(),        ## 对角、上三角、下三角
dim(),nrow(),ncol()              ## 维数、行数、列数
qr(),svd(),eigen()          ## QR分解、SVD分解、谱分解
rowMeans(),rowSums,colMeans(),colSums() ## 每行/列的平均值/和
```

请认真弄清各个函数的用法,非常有用。

(vii) aplly 和 sweep

apply 函数对一个矩阵按行(以 1 来表示)或者按列(以 2 来表示)进行计算,

```
> apply(z,1,sum)
[1] 15 40 65 90
> apply(z,2,sum)
[1] 34 38 42 46 50
```

sweep 函数将一个向量作用到一个矩阵上,按行(以 1 来表示)或者按列(以 2 来表示)进行计算

```
> v1=21:24
> v2=11:15
```

```
> sweep(z,1,v1,"+")    ## z 的第i行分别加上v1的第i个元素
     [,1] [,2] [,3] [,4] [,5]
[1,]   22   23   24   25   26
[2,]   28   29   30   31   32
[3,]   34   35   36   37   38
[4,]   40   41   42   43   44
> sweep(z,2,v2,"/")   ## z 的第j列分别除以v2的第j个元素
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.09090909 0.1666667 0.2307692 0.2857143 0.3333333
[2,] 0.54545455 0.5833333 0.6153846 0.6428571 0.6666667
[3,] 1.00000000 1.0000000 1.0000000 1.0000000 1.0000000
[4,] 1.45454545 1.4166667 1.3846154 1.3571429 1.3333333
```

2.3 非数值型数据

(i) 字符串

字符串(string)可以是用“”标示的一串字符,R 用一个向量存储,比如

```
> s=c("Sunday","Monday","Tuesday","Wendesday","Thursday","Friday","Satu
> s[2:3]
[1] "Monday"  "Tuesday"
```

可以将字符串进行大小写转换

```
> toupper(s)
[1] "SUNDAY"   "MONDAY"   "TUESDAY"  "WENDESDAY" "THURSDAY"  "FRIDAY"   "
> tolower(s)
[1] "sunday"   "monday"   "tuesday"  "wendesday" "thursday"  "friday"   "
```

可以检查每个字符串的长度

```
> nchar(s)
[1] 6 6 7 9 8 6 8
```

可以截取 s 每个字符串的前三个或后三个字符

```
> substr(s,1,3)
[1] "Sun" "Mon" "Tue" "Wen" "Thu" "Fri" "Sat"
> substr(s,nchar(s)-2,nchar(s))
[1] "day" "day" "day" "day" "day" "day" "day"
```

可以将 s 的每个字符串分解为数个较小的字符串。

```
> x=c(as = "asfef", qu = "qwerty", "stuff.blah.yech")
> # 在字符e处将每个字符串分解
> strsplit(x, "e")
$as
```

```
[1] "asf" "f"

$qu
[1] "qw"  "rty"

[[3]]
[1] "stuff.blah.y" "ch"

> strsplit("a.b.c", ".")
[[1]]
[1] "" "" "" "" ""

> ## 对上面的特殊字符“.”，字符串没有正确地分解
> ## 正确的方法是用fixed=TRUE来设置
> strsplit("a.b.c", ".", fixed = TRUE)
[[1]]
[1] "a" "b" "c"
```

可以将一些字符串合并成一个

```
> paste(s)
[1] "Sunday"   "Monday"   "Tuesday"  "Wendesday" "Thursday" "Friday"   "
> paste(s,collapse="_")
[1] "Sunday_Monday_Tuesday_Wendesday_Thursday_Friday_Saturday"
```

注意上面的两个命令的区别:第一个命令将 s 的每个字符串合并成为一个,所以等于什么也没做,第二个命令将 s 的所有字符串合并成一个,并以"_" 连接。对比下面的命令,paste 将 n 和 s 的第 i 个元素合并成一个,并且以":" 连接,所以共得到 7 个较长的字符串

```
> n=0:6
> paste(n,s,sep=":")
[1] "0:Sunday"   "1:Monday"   "2:Tuesday"  "3:Wendesday" "4:Thursday" "5
```

(ii) 因子

因子(factor)是一个对等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。无序的因子可以由任意字符串或数字(数字被当成字符串处理)组成,并可以用函数 factor()ordered() 转换为有序的因子

```
> grade=c("A","1",100,"100","a",1,"*",TRUE,"Z")
> factor(grade)
[1] A    1    100  100  a    1    *    TRUE Z
Levels: * 1 100 a A TRUE Z
> ordered(grade)
[1] A    1    100  100  a    1    *    TRUE Z
Levels: * < 1 < 100 < a < A < TRUE < Z
```

无序的因子没有水平 (levels) 之分,而有序的因子存在高低之分

```
> levels(grade)
```

```
NULL
> Grade=ordered(grade)
> levels(Grade)
[1] "*"    "1"    "100"  "a"    "A"    "TRUE" "Z"
```

可以将有序因子转换为正整数,而无序的不可以

```
> levels(grade)
> k=as.numeric(grade)
Warning message:
NAs introduced by coercion
> k=as.numeric(Grade)
> k
[1] 5 2 3 3 4 2 1 6 7
```

2.4 时间日期

R 可以很容易得到当前的时间和日期

```
> today=Sys.Date()
> today
[1] "2014-07-07"
> now=Sys.time()
> now
[1] "2014-07-07 20:52:44 CST"
```

日期和时间可以按照一定的格式进行转化

```
%a Day of the week
%d Day of the month (decimal number)
%m Month (decimal number)
%b Month (abbreviated)
```

```
%B Month (full name)
%y Year (2 digit)
%Y Year (4 digit)
```

例如

```
> format(now, "%a %b %d %X %Y")
[1] "周一 七月 07 20:52:44 2014"
> format(today, format="%Y %B %d %Y")
[1] "2014 七月 07 2014"
> format(today,format="%m")
[1] "07"
> format(today, format="%m-%d %y")
[1] "07-07 14"
> format(now, "%a %b %d %X %Y")
[1] "周一 七月 07 20:52:44 2014"
```

有些特殊的函数可以提取特定的信息,包括星期、月份、季度等

```
> weekdays(today, abbreviate=TRUE)
[1] "周一"
> weekdays(now, abbreviate = FALSE)
[1] "星期一"
> months(today)
[1] "七月"
> quarters(now)
[1] "Q3"
```

也可以将字符串按照一定的格式转换为日期或时间

```
> as.Date('1915-6-16')
[1] "1915-06-16"
> as.Date('1990/02/17')
[1] "1990-02-17"
> d=as.Date('1/15/2001',format='%m/%d/%Y')
> format(d, format="%Y %B %d")
[1] "2001 一月 15"
```

```
> thedate = ISOdate(2005,10,21,18,47,22)
> thedate
[1] "2005-10-21 18:47:22 GMT"
```

日期和时间是可以参加运算的。默认的起始时间是 1970 年 1 月 1 日 0 时 0 分 0 秒(不同系统可能不同)。日期的单位是天数,时间是秒数。

```
> secs=as.numeric(now)
> secs
[1] 1404737564
> day1=as.Date('1970-1-1')
> day2=as.Date('1970-1-17')
> as.numeric(day1)
[1] 0
> as.numeric(day2)
[1] 16
> days=day2-day1
> days
```

```
Time difference of 16 days
> as.numeric(days)
[1] 16
```

2.5 列表

R 的列表(list)是一个以对象组成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类型,如一个列表可以同时包括数值向量,逻辑向量,矩阵,复向量,字符数组,函数等等。下面的例子演示怎么创建一个列表:

```
> Lst=list(name="Fred", wife="Mary", no.children=3,child.ages=c(4,7,9))
> Lst
$name
[1] "Fred"
```

```
$wife
[1] "Mary"

$no.children
[1] 3

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

分量常常会被编号的(numbered),并且可以利用这种编号来访问分量。如果列表 Lst 有四个分量,这些分量则可以用 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。这里特别要注意一下 Lst[[1]] 和 Lst[1] 的差别。前者得到的是列表 Lst 中的第一个对象, 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。

```
> Lst[4]
$child.ages
```

```
[1] 4 7 9
> Lst[[4]]
[1] 4 7 9
> Lst[[4]][2]
[1] 7
```

列表的分量可以被命名,这种情况下可以通过名字访问。此时,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,或者用一个 $ 符号连接,比如

```
> Lst$child.ages
[1] 4 7 9
> Lst$child.ages[2]
[1] 7
```

可以通过 names() 修改列表分量的名字,比如将第一个分量的名字"name"改为"husband"

```
> names(Lst)
[1] "husband"     "wife"         "no.children" "child.ages"
> names(Lst)[1]="husband"
> Lst$husband
[1] "Fred"
```

2.6 数据框

数据框(data frame)是一个属于"data.frame" 类的列表。不过,对于可能属于数据框的列表对象有下面一些限制条件,

- 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.

比如

```
> city=c("DC","LA","NY")
> PI=c(11.3,10.2,13.0)
> FS=c(4.2,4.5,3.5)
> fin=data.frame(City=city,Person.Income=PI,Family.Size=FS)
> fin
  City Person.Income Family.Size
1   DC          11.3         4.2
2   LA          10.2         4.5
3   NY          13.0         3.5
```

与列表 list 相同(实际上,数据框本身就是一种特殊的列表),访问数据框特定的列可以用 $ 符号,并且还可以添加新的列

```
> fin$City
```

```
[1] DC LA NY
Levels: DC LA NY
> HI=fin$Person.Income*fin$Family.Siz
> fin$HHold.Income=HI
> fin
  City Person.Income Family.Size HHold.Income
1   DC          11.3         4.2        47.46
2   LA          10.2         4.5        45.90
3   NY          13.0         3.5        45.50
```

另外,一个有用的命令是 edit(), 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。注意,edit(x) 或 y=edit(x) 并不改变 x 的值,只有 x=edit(x) 才会将修改的结果保存到 x 中,比较下面的例子

```
> edit(fin)
  City Person.Income Family.Size HHold.Income
1   DC          11.3         4.2        47.46
```

```
2   LA          10.2          4.5         45.90
3   NY          12.0          3.5         45.50
> fin=edit(fin)
> fin
  City Person.Income Family.Size HHold.Income
1   DC          11.3          4.2         47.46
2   LA          10.2          4.5         45.90
3   NY          12.0          3.5         45.50
```

2.7 读写与合并

(i) 读取

一般 R 导入数据不是通过用代码来创建,而是从某些文件,比如 txt 或 excel 表格中导入。首先, 应该知道当前 R 的工作路径 (working directory) 再哪里,以方便程序寻找数据文件。请参考第一章的介绍学习如何通过

setwd() 和 getwd() 查看或改变默认工作路径

R 可以读取多种存储结构的数据,但很多人认为,最常用、最好用的是.txt 文件和.csv 文件(即逗号分隔文件,Comma Separated Values File)。虽然使用特定的程序包,R 可以读取 Excel 表格,但是由于 Excel 实际上是一种存储结构非常复杂的数据,所以最好在 MS Office 上"另存为……逗号分隔文件"。这种文件可以直接通过 Excel 浏览。

假设 D:/data 下有两个个文件, authors.csv 和 book.txt,则可以分别用 read.csv() 和 read.table() 读取,读取的结果分别存入了两个数据框 a 和 b 中。

```
> a=read.csv("authors.csv")
> b=read.table("books.txt",header=TRUE)
> a
   surname nationality deceased
1    Tukey          US      yes
2 Venables   Australia       no
3  Tierney          US       no
```

```
4   Ripley          UK       no
5   McNeil   Australia       no
> b
      name                          title     other.author
1    Tukey     Exploratory Data Analysis             <NA>
2 Venables Modern Applied Statistics ...           Ripley
3  Tierney                      LISP-STAT             <NA>
4   Ripley            Spatial Statistics             <NA>
5   Ripley         Stochastic Simulation             <NA>
6   McNeil     Interactive Data Analysis             <NA>
7   R Core          An Introduction to R Venables & Smith
```

read.csv() 和 read.table() 可以有很多参数可以调整，根据文件中数据的存储方式，比如数据之间用什么分割、如何换行等等，应该选用合适的参数才能正确读取。具体设置方法请自行学习。R 还有许多其它函数也可以读取数据，也请根据具体问题自行学习掌握。

(ii) 存储

对应的,可以用 write.csv() 或 write.table() 将数据写入文件。若给定的文件不存在,则创建新的文件;若已经存在,则会覆盖掉,因此应当小心设置文件名。

```
> write.csv(a,"authors-new.csv",row.names=FALSE)
> write.table(b,"books-new.txt",row.names=FALSE)
```

此时查看当前的工作目录 D:/data/,应该发现多了两个文件,分别是 authors-new.csv 和 books-new.txt。

另外一种方式是将数据用 save() 写入一种 R 专用的.RData 文件类型

```
> save(a,b,file="ab.RData")
```

将来若再次使用到这个数据,就可以用 load() 来直接载入。由于.Rdata 是 R 专用的数据类型,因此读取该文件类型的速度要快得多。

```
> load(file="ab.RData")
```

(iii) 合并

如果有两个数据框,就可以将二者根据一定的关键词合并为一个

```
> ab = merge(a,b, by.x = "surname", by.y = "name")
> ab
  surname nationality deceased                    title other.author
1  McNeil   Australia       no  Interactive Data Analysis       <NA>
2  Ripley         UK       no           Spatial Statistics       <NA>
3  Ripley         UK       no        Stochastic Simulation       <NA>
4 Tierney         US       no                    LISP-STAT       <NA>
5   Tukey         US      yes  Exploratory Data Analysis       <NA>
6 Venables   Australia       no Modern Applied Statistics ...     Ripley
```

可见只有当两个数据框都含有某个关键词时,该关键词对应的行才会被包含进合并后的数据框。若放松这个要求,可以通过设置 merge 的参数 all,all.x 或 all.y 选择包含哪些数据。例如

```
> ab2 = merge(a,b, by.x = "surname", by.y = "name",all=TRUE)
> ab2
  surname nationality deceased                    title    other.author
1  McNeil  Australia      no    Interactive Data Analysis          <NA>
2  Ripley        UK      no          Spatial Statistics          <NA>
3  Ripley        UK      no       Stochastic Simulation          <NA>
4 Tierney        US      no                   LISP-STAT          <NA>
5   Tukey        US     yes    Exploratory Data Analysis          <NA>
6 Venables  Australia     no Modern Applied Statistics ...        Ripley
7  R Core       <NA>    <NA>       An Introduction to R Venables & Smith
```

2.8 自定义函数

(i) 条件控制

if(...){...} 是基本的条件控制语句，即若某个条件满足，就会执行一段命令，否则就不做任何事情。比如下面这段话可以判断 a 是不是一个正数

```
> a=1
> if(a>0){
+   print("a is positive")
+ }
[1] "a is positive"
```

if 语句可以和 else 语句联合使用，如果条件满足执行第一段命令，否则制定 else 之后的命令。比如下面的语句可以找出 a 和 b 之间的最小值。

```
> a=1
```

```
> b=2
> if(a<b){
+  m=a
+ } else{
+   m=b
+ }
> m
[1] 1
```

(ii) 循环控制

for(... in ...){...} **是一个循环控制语句,比如下面的语句可以打印出** 1,2,3 **的平方**

```
> for(i in 1:3){
+   print(i^2)
+ }
```

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

也可以用 while(...){...} 来实现，此时一定要注意避免出现无限循环的情况。

```
> i=1
> while(i<4){
+   print(i^2)
+   i=i+1
+ }
[1] 1
[1] 4
[1] 9
```

关键字 break 可以用于结束任何循环，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。请思考下面的例子中，为什么程序

只打印出 4 和 5 的平方

```
> for(i in 1:10){
+   if(i<=3) next
+   if(i>3 & i<6) cat("the square of",i,"is",i^2,"\n")
+   if(i>=6) break
+ }
the square of 4 is 16
the square of 5 is 25
```

repeat{...} 也可以构成一个循环控制语句，但 break 是结束 repeat 循环的唯一办法。

```
> i=1
> repeat{
+   print(i^2)
+   i=i+1
+   if(i>3) break
```

```
+ }
[1] 1
[1] 4
[1] 9
```

(iii) 函数

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。

一个函数是通过下面的语句形式定义的:

```
fname=function(arg1,arg2,...){  ## arg1,arg2,...是函数输入的一个或几个参
 ...          ## some expressions
 return(...) ## 返回函数的取值
}
```

比如下面这个函数就可以通过给出直角三角形的两条直边,得到斜边的长度

```
> slopeLength=function(a,b){
+  y=sqrt(a^2+b^2)
+  return(y)
+ }
> slopeLength(3,4)
[1] 5
```

如果想让函数返回几个值,也是可以的。实际上,函数可以返回任意数据类型,包括字符串、因子、向量、函数、数据框、列表等等,并且可以将几种类型的数据合并成一个列表,然后一并返回。看下面的例子

```
> myTri=function(a,b){
+   z=sqrt(a^2+b^2)
+   e=c(a,b,z)
+   m="This is a right triangle"
```

```
+   s=list(Message=m,Area=a*b,Edge=e)
+   return(s)
+ }
> myTri(3,4)
$Message
[1] "This is a right triangle"

$Area
[1] 12

$Edge
[1] 3 4 5
```

(iv) 调试:print() 与 cat()

和其它编程语言类似,调试 R 程序一个常用的办法是在程序中加入打印语句,比如下面的一段程序

```
> for(i in -2:3){
+   if (i <=0 ) cat(i, "is not a positive number", "\n")
+   else cat("log(",i,")=",log(i),"\n")
+ }
-2 is not a positive number
-1 is not a positive number
0 is not a positive number
log( 1 )= 0
log( 2 )= 0.6931472
log( 3 )= 1.098612
```

上面的 cat() 函数将一串变量连接(concatenate)起来(上面的例子中包含了变量 i,字符串"is not a positive number", 和换行符"\n"),然后输入到屏幕上。

另外一个 print() 函数也可以实现类似的功能,但是 print() 只能打印一个变量,所以需要将几个变量用 paste() 函数连接起来。注意,print() 运行后会直接换行,所以不用在最后加上换行符"\n",这一点和 cat() 不同。

```
> for(i in -2:3){
+   if (i <=0 ){
+     m1=paste(i, "is not a positive number")
+     print(m1)
+   } else{
+     m2=paste("log(",i,")=",log(i),sep="")
+     print(m2)
+   }
+ }
[1] "-2 is not a positive number"
[1] "-1 is not a positive number"
[1] "0 is not a positive number"
[1] "log(1)=0"
[1] "log(2)=0.693147180559945"
[1] "log(3)=1.09861228866811"
```

另外一个有用的函数是 sprintf(),它和 C 语言中的 sprintf 函数是一样的,能够将一个数按照给定的格式答应出来,请自行通过 help(sprintf) 查看如

何设置格式,比较下面的结果

```
> sprintf("%f", pi)
[1] "3.141593"
> sprintf("%.3f", pi)
[1] "3.142"
> sprintf("%1.0f", pi)
[1] "3"
> sprintf("%5.1f", pi)
[1] "  3.1"
> sprintf("%05.1f", pi)
[1] "003.1"
> sprintf("%+f", pi)
[1] "+3.141593"
> sprintf("% f", pi)
[1] " 3.141593"
> sprintf("%-10f", pi) # left justified
[1] "3.141593  "
```

```
> sprintf("%e", pi)
[1] "3.141593e+00"
> sprintf("%E", pi)
[1] "3.141593E+00"
> sprintf("%g", pi)
[1] "3.14159"
> sprintf("%g",   1e6 * pi) # -> exponential
[1] "3.14159e+06"
> sprintf("%.9g", 1e6 * pi) # -> "fixed"
[1] "3141592.65"
> sprintf("%G", 1e-6 * pi)
[1] "3.14159E-06"
```

2.9 例题及习题

例 2.1 我们知道泰勒展开公式

$$f(x) = f(x_0) + \sum_{n=1}^{k} f^{(n)}(x_0)(x - x_0)^n + o((x - x_0)^k)$$

所以有如下近似公式 (k 是一个比较大的整数)

$$e^x \approx 1 + \sum_{n=1}^{k} \frac{x^n}{n!}$$

下面的例子可以将指数函数用泰勒级数逼近到任意的精度

```
> exp.taylor=function(x,k){
+   n=1:k
+   z=x^n/factorial(n)
+   y=1+sum(z)
+   return(y)
```

```
+ }
> exp.taylor(0.5,3)
[1] 1.645833
> exp.taylor(0.5,5)
[1] 1.648698
> exp.taylor(0.5,100)
[1] 1.648721
> exp(0.5)
[1] 1.648721
```

可见这是一个很好地近似。

1. 已知 a=c(3,9,0,12,19),b=c(2,5,6,9,0), 写出一段代码实现如下任务
   (a) 找到 a 中第 3 大的元素,即 9
   (b) 将 a,b 合并成一个 5x2 的矩阵 m,a 在 m 的第 1 列,b 在 m 的第 2 列
   (c) 使用 apply() 和函数 mean 找出 m 的各行、各列的平均值

2. 泰勒展开

   (a) 已知三角函数 sin(x) 的泰勒展开是

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots + \frac{(-1)^{k-1}x^{2k-1}}{(2k-1)!} + \cdots$$

   请定义一个 R 函数 sin.taylor(x,k)，使得它用 k 阶 Taylor 级数逼近 sin(x)

   (b) 定义一个函数 myfun(x,k), 使得它能够逼近 $log(1+x^2)$

3. 黎曼积分。对一个连续函数 $f(x), x \in [a,b]$，它的积分可以表示为黎曼和的极限。即，将区间 $[a,b]$ 分成 n 份 $a = a_0 < a_1 < \cdots < a_n = b$，则当分割足够小(即 $\max\{a_i - x_{i-1}\} \to 0$)的时候

$$\int_a^b f(x)\mathrm{d}x = \lim_{\max\{a_i - x_{i-1}\}\to 0} \sum_{i=1}^{n} f(x_i)(x_i - x_{i-1})$$

   根据这个公式，完成如下任务

   (a) 假设 $f(x) = x^2 + 1$，将区间 $[-2,1]$ 按照间隔 0.001 平分，求出黎曼和 $\sum_{i=1}^{n} f(x_i)(x_i - x_{i-1})$

(b) 编写一个 R 函数 Int, 补全下面的代码求函数 f 在区间 [xmin,xmax] 上的黎曼和,:

```
Int=function(f,xmin,xmax,dx){
      ## f 是一个函数
      ## xmin, xmax 是
    ## dx 是区间 [xmin,xmax] 的分割长度，即每个小区间长度为 dx
  ....   ## 一些代码
 s=... ## s 是 f 在 [xmin,xmax] 上的黎曼和，小区间的分割长度是
  return(s)
}
```

(c) 假设 $f(x) = x^2 + 1$,求出 Int(f,-2,1,0.001) 的值。

(d) 假设 $f(x) = sin(x)$,求出 Int(f,0,pi,0.1)、Int(f,0,pi,0.01) 和 Int(f,0,pi,0.0001) 的值

4. 逆矩阵。在 R 中 solve() 是一个计算矩阵逆的内置函数,根据下面的步骤提示,编写一个函数 InvMatrix(),使得它与 solve() 的功能类似,使得当 A 是非奇异方阵时,InvMatrix(A) 是 A 的逆矩阵;当 A 奇异

时,InvMatrix(A)=NA

第 三 章　绘图

R 绘图十分强大,特别是结合统计针对数据的绘图十分方便,这是它不断扩大市场份额的重要原因

3.1 命令

绘图命令一般分为两种,高级的和低级的。高级绘图函数是创建一个新的图形,通常包括坐标轴、标签、标题等

高级绘图命令

函数名	功能
plot(x)	以 x 的值为纵坐标,x 的序号值 1:length(x) 为横坐标画散
plot(x,y)	画 x 和 y 的二维图
pie(x)	饼图
boxplot(x)	箱线图
hist(x)	x 的频率(或密度)直方图
parplot(x)	x 的值的条形图
pairs(x)	x(数据框或矩阵)的各列间的二元图
coplot(x y|z)	固定 z 的值或范围,画 x 与 y 的二元图
matplot(x,y)	二元图,x 的第 i 列对应 y 的第 i 列
qqnorm(x);qqline(x)	x 的 Q-Q 图,正态分位数 -分位数图
image(x,y,z)	x,y,z 的三元图,z 的值以颜色表示
heatmap(x)	热图
contour(x,y,z)	等高图

低级绘图函数是在原有图形上添加新的图形元素,比如额外的点、线、标签等

低级绘图命令

函数名	功能
points(x,y)	添加 (x,y) 点
lines(x,y)	添加 (x,y) 组成的折线
text(x,y,labels)	将 labels 的每个字符串添加到相应的 (x,y) 位置处
mtext(text,side=3,line=0,...)	在 side 指定的边空处添加 text 制定的文字
segments(x0,y0,x1,y1)	用线段从 (x0,y0) 的各点连接到 (x1,y1) 各点
rug(x)	在 x 坐标轴上的每个数据点处添加短线
arrows(x0,y0,x1,y1)	用带箭头的线段从 (x0,y0) 的各点连接到 (x1,y1)
abline(a,b)	绘制斜率 b,截距 a 的直线
abline(h=y);abline(v=x)	在横(纵)坐标 y(x)处画水平(垂直)直线
rect(x1,y1,x2,y2)	画一个矩形,左下角和右上角分别为 (x1,y1) 和 (x
polygon(x,y)	绘制连接 (x,y) 各点组成的多边形
legend(x,y,legend)	(x,y) 处添加图例,内容为 legend
title()	为图形添加标题或副标题
axis()	为图形添加坐标轴
box()	为图形添加边框
locator()	获取鼠标位置

为了使得画出的图形美观、包括更多更明显的信息,可以在绘图函数中设置各种参数

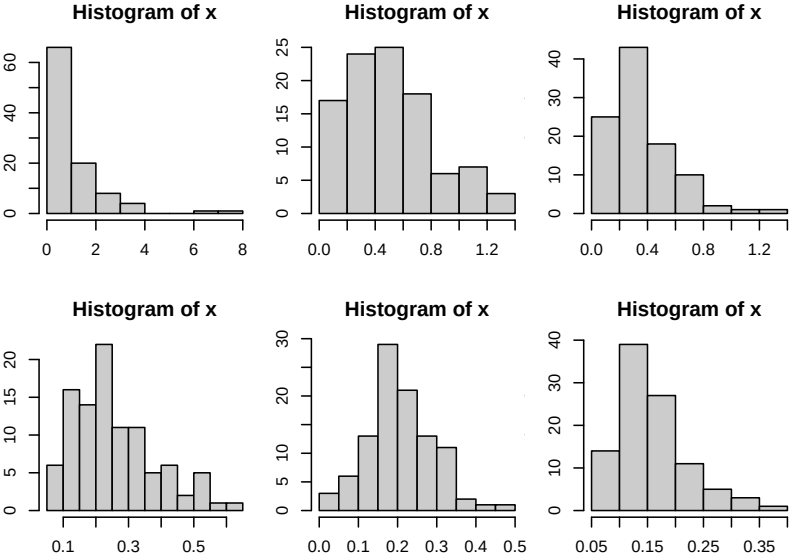
绘图命令的参数

参数	功能
add=TRUE	若 TRUE 则新图像添加到前一个图形上,若 FALSE 则绘新图
axes=TRUE	是否画坐标轴
type="p"	图形类型。p:点;l:线;b/o:点连线;h:垂直线;s/S:阶梯式
xlim,ylim	图像显示范围
xlab,ylab	x,y 坐标的 label
main	主标题
sub	副标题
mar	图片边框外留白
col	颜色
cex	点、字的大小
lwd	线的粗细
lty	虚线、点线、实线等类型

3.2 画图面板分割

可以用 par(mfrow=c(m,n)) 等命令将画图面板分为 m 行 n 列的阵列。然后将 m*n 个图依次画在各个部分中。注意，画图完毕后，一般要注意将面板用 par(mfrow=c(1,1)) 恢复为一整块。

```
par(mfrow=c(2,3))
par(mar=c(3,3,3,.1))
for(i in 1:6){
  x=rgamma(100,i,i^2)
  hist(x,col="grey80")
}
par(mfrow=c(1,1))
```


Histogram of x
0
20
40
60
0
2
4
6
8
Histogram of x
0
5
10
15
20
25
0.0
0.4
0.8
1.2
Histogram of x
0
10
20
30
40
0.0
0.4
0.8
1.2
Histogram of x
0
5
10
15
20
0.1
0.3
0.5
Histogram of x
0
5
10
20
30
0.0
0.1
0.2
0.3
0.4
0.5
Histogram of x
0
10
20
30
40
0.05
0.15
0.25
0.35

类似的命令还包括 par(mfcol=c(...)),split.screen(),layout() 等函数,具体用法请参考帮助文档。

3.3 图片保存

R 生成的图片可以通过右击图片或其它方法直接保存为适当的格式,但更加方便可靠地方法是通过运行命令而保存。

R 支持多种常用的图片格式,包括 pdf、postscript(ps 或 eps)、gif、svg、tiff、png、jpeg、bmp、xfig、bitmap、pictex 等。对于不同的图片格式,请使用命令 help(png) 等命令查看帮助文档。例如以保存为 pdf 图片为例:

```
pdf(file="D:/picture/mypicture.pdf",width=5,height=3.5) ## 创建空白的图片并设置
plot(x,y)                ## 开始画图
...
dev.off()                ## 绘图完毕关闭图形设备
```

值得指出的是,pdf、ps、eps、svg、pictex 格式的图片是向量形式,因此可以随

意放大或缩小而不会损失像素。所以如果想在 microsoft office 中插入图片，建议使用 svg 格式的图片，而不要使用 png 等格式。当然了，若图片包含的数据非常大，建议使用点阵格式的图片格式。

3.4 例子

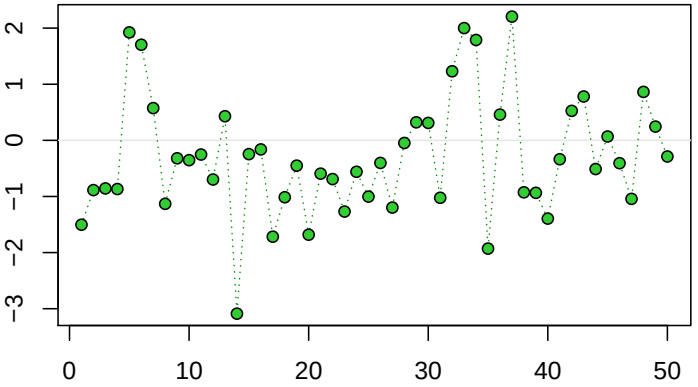
R 中有很多绘图的例子，可以通过键入如下命令得到：

```
require(datasets)
require(grDevices)
require(graphics)
demo(graphics}
demo(persp)
```

(i) 散点图

```
## Note that colors are generally specified
## by a character string name (taken from the X11 rgb.txt file) and that line
## textures are given similarly.  The parameter "bg" sets the background
## parameter for the plot and there is also an "fg" parameter which sets
## the foreground color.
x <- stats::rnorm(50)
opar <- par(bg = "white")
plot(x, ann = FALSE, type = "n")
abline(h = 0, col = gray(.90))
lines(x, col = "green4", lty = "dotted")
points(x, bg = "limegreen", pch = 21)
title(main = "Simple Use of Color In a Plot",
      xlab = "Just a Whisper of a Label",
      col.main = "blue", col.lab = gray(.8),
      cex.main = 1.2, cex.lab = 1.0, font.main = 4, font.lab = 3)
```

Simple Use of Color In a Plot



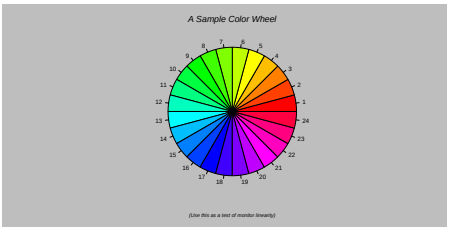
Just a Whisper of a Label

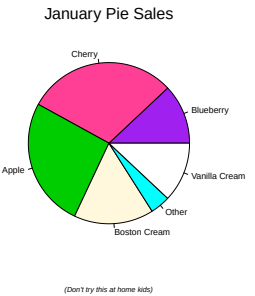
(ii) 饼图

```
## A little color wheel.  This code just plots equally spaced hues in
## a pie chart. If you have a cheap SVGA monitor (like me) you will
## probably find that numerically equispaced does not mean visually
## equispaced.  On my display at home, these colors tend to cluster at
## the RGB primaries.  On the other hand on the SGI Indy at work the
## effect is near perfect.
par(bg = "gray")
pie(rep(1,24), col = rainbow(24), radius = 0.9)
title(main = "A Sample Color Wheel", cex.main = 1.4, font.main = 3)
title(xlab = "(Use this as a test of monitor linearity)",
      cex.lab = 0.8, font.lab = 3)
## We have already confessed to having these.  This is just showing off X11
## color names (and the example (from the postscript manual) is pretty "cute".

pie(pie.sales,
    col = c("purple","violetred1","green3","cornsilk","cyan","white"))
title(main = "January Pie Sales", cex.main = 1.8, font.main = 1)
```

```
title(xlab = "(Don't try this at home kids)", cex.lab = 0.8, font.lab = 3)
```

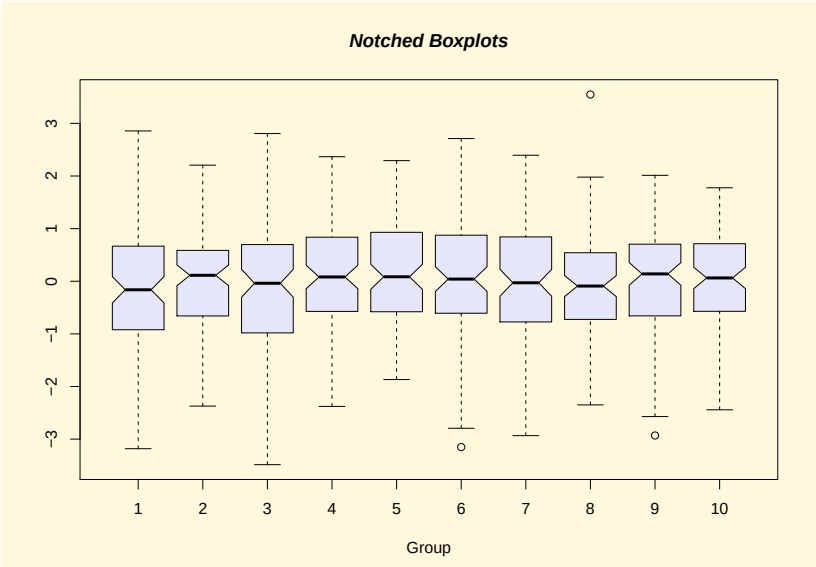




(iii) 箱线图

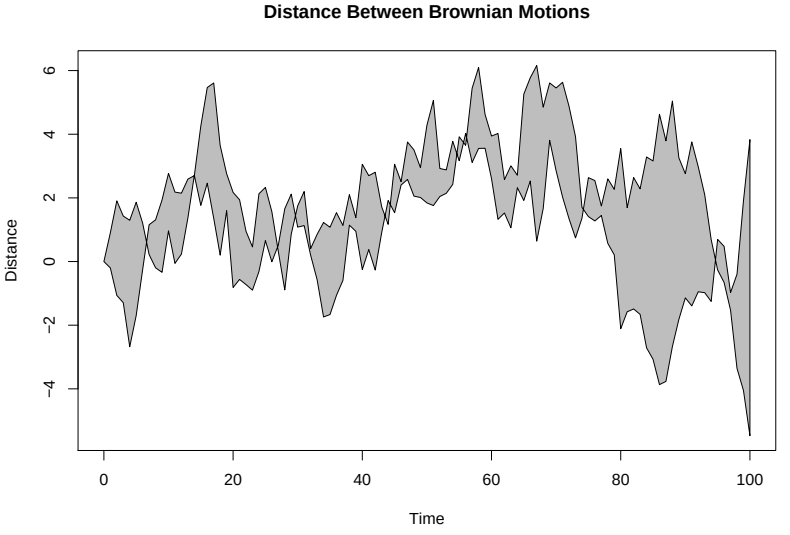
```
## Boxplots:  I couldn't resist the capability for filling the "box".
## The use of color seems like a useful addition, it focuses attention
## on the central bulk of the data.
par(bg="cornsilk")
```

```
n <- 10
g <- gl(n, 100, n*100)
x <- rnorm(n*100)
boxplot(split(x,g), col="lavender", notch=TRUE)
title(main="Notched Boxplots", xlab="Group", font.main=4, font.lab=1)
```


Notched Boxplots
3
2
1
0
−1
−2
−3
1
2
3
4
5
6
7
8
9
10
Group

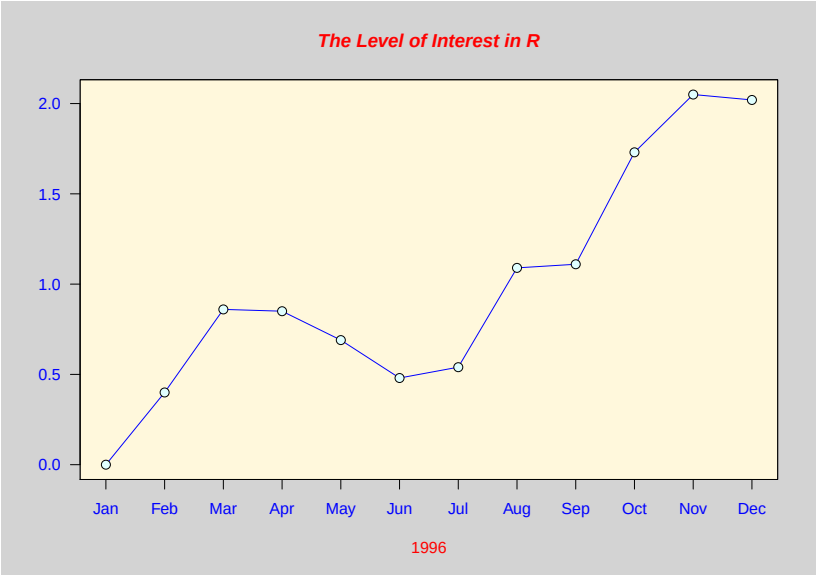
(iv) 多边形

```
## An example showing how to fill between curves.
par(bg="white")
n <- 100
x <- c(0,cumsum(rnorm(n)))
y <- c(0,cumsum(rnorm(n)))
xx <- c(0:n, n:0)
yy <- c(x, rev(y))
plot(xx, yy, type="n", xlab="Time", ylab="Distance")
polygon(xx, yy, col="gray")
title("Distance Between Brownian Motions")
```


Distance Between Brownian Motions
Distance
6
4
2
0
−2
−4
0
20
40
60
80
100
Time

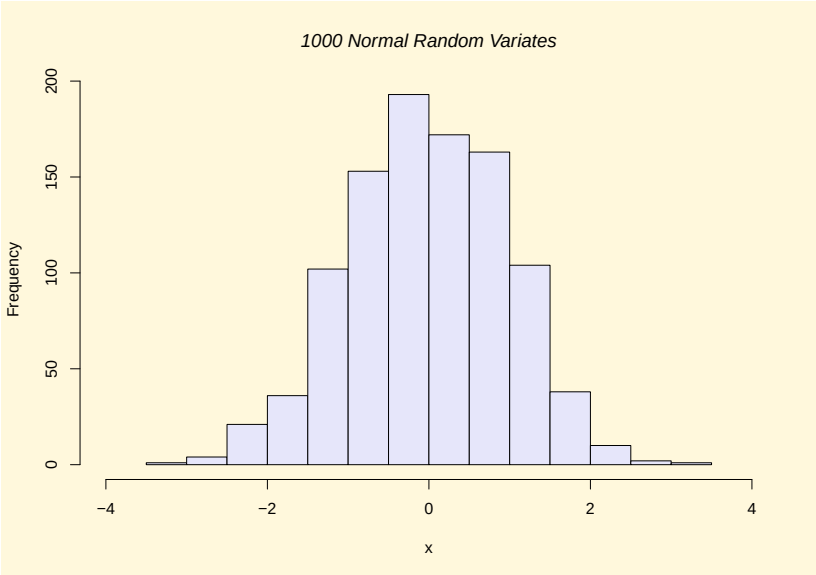
(v) 坐标

```
## Colored plot margins, axis labels and titles.  You do need to be
## careful with these kinds of effects. It's easy to go completely
## over the top and you can end up with your lunch all over the keyboard.
## On the other hand, my market research clients love it.
x <- c(0.00, 0.40, 0.86, 0.85, 0.69, 0.48, 0.54, 1.09, 1.11, 1.73, 2.05, 2.02)
par(bg="lightgray")
plot(x, type="n", axes=FALSE, ann=FALSE)
usr <- par("usr")
rect(usr[1], usr[3], usr[2], usr[4], col="cornsilk", border="black")
lines(x, col="blue")
points(x, pch=21, bg="lightcyan", cex=1.25)
axis(2, col.axis="blue", las=1)
axis(1, at=1:12, lab=month.abb, col.axis="blue")
box()
title(main= "The Level of Interest in R", font.main=4, col.main="red")
title(xlab= "1996", col.lab="red")
```


The Level of Interest in R
2.0
1.5
1.0
0.5
0.0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
1996

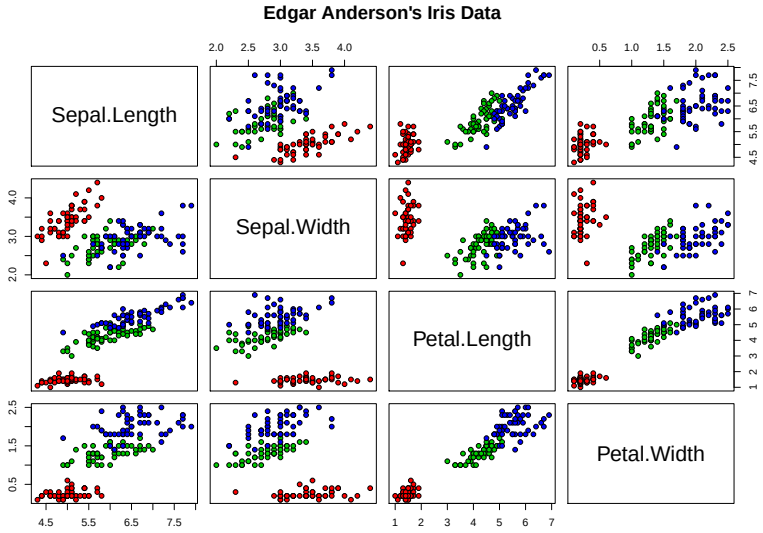
(vi) 直方图

```
## A filled histogram, showing how to change the font used for the
## main title without changing the other annotation.
par(bg="cornsilk")
x <- rnorm(1000)
hist(x, xlim=range(-4, 4, x), col="lavender", main="")
title(main="1000 Normal Random Variates", font.main=3)
```


1000 Normal Random Variates
Frequency
0
50
100
150
200
−4
−2
0
2
4
x

(vii) 散点阵

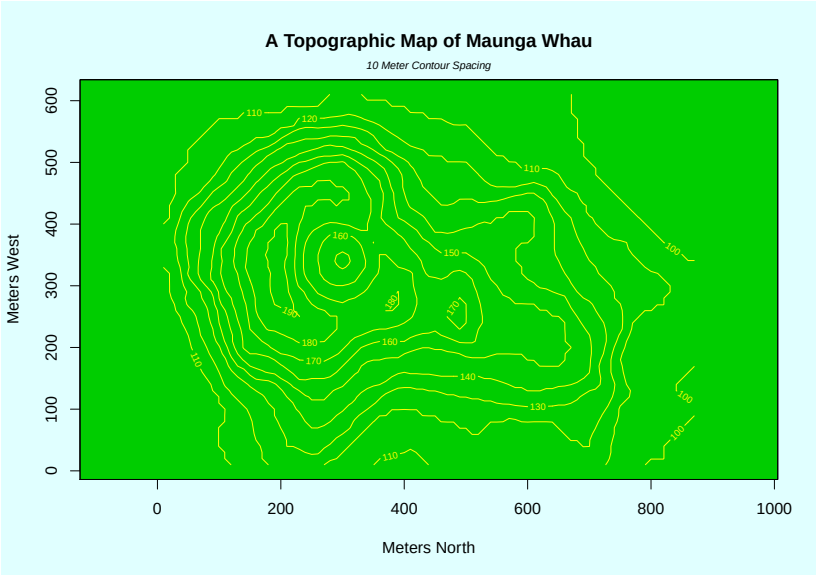
```
## A scatterplot matrix
## The good old Iris data (yet again)
pairs(iris[1:4], main="Edgar Anderson's Iris Data", pch=21,
      bg = c("red", "green3", "blue")[unclass(iris$Species)])
```


Edgar Anderson's Iris Data
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width

(viii) 等高线图

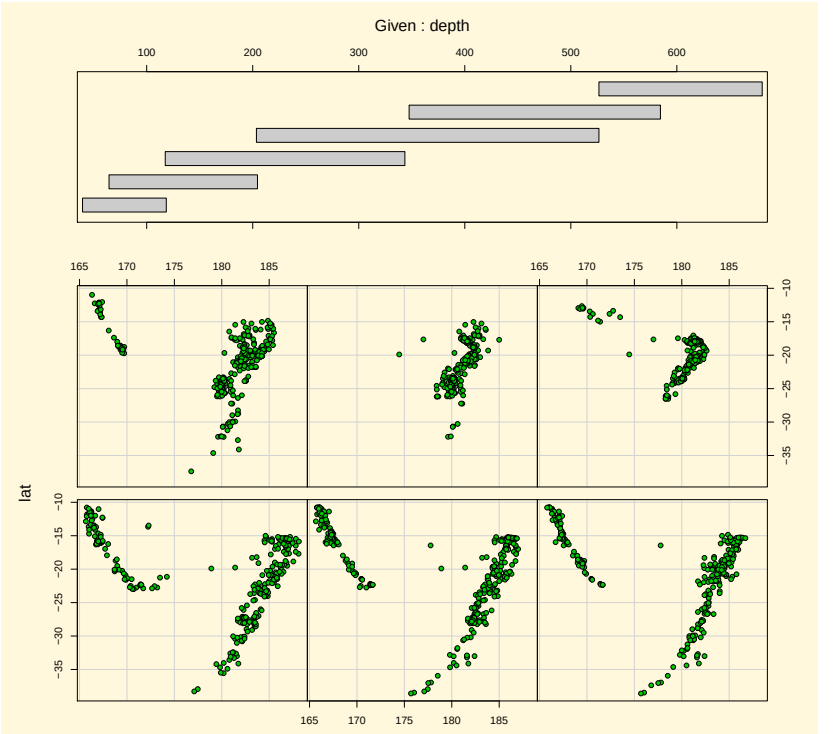
```
## Contour plotting
## This produces a topographic map of one of Auckland's many volcanic "peaks".
x <- 10*1:nrow(volcano)
y <- 10*1:ncol(volcano)
lev <- pretty(range(volcano), 10)
par(bg = "lightcyan")
pin <- par("pin")
xdelta <- diff(range(x))
ydelta <- diff(range(y))
xscale <- pin[1]/xdelta
yscale <- pin[2]/ydelta
scale <- min(xscale, yscale)
xadd <- 0.5*(pin[1]/scale - xdelta)
yadd <- 0.5*(pin[2]/scale - ydelta)
plot(numeric(0), numeric(0),
     xlim = range(x)+c(-1,1)*xadd, ylim = range(y)+c(-1,1)*yadd,
     type = "n", ann = FALSE)
```

```
usr <- par("usr")
rect(usr[1], usr[3], usr[2], usr[4], col="green3")
contour(x, y, volcano, levels = lev, col="yellow", lty="solid", add=TRUE)
box()
title("A Topographic Map of Maunga Whau", font= 4)
title(xlab = "Meters North", ylab = "Meters West", font= 3)
mtext("10 Meter Contour Spacing", side=3, line=0.35, outer=FALSE,
      at = mean(par("usr")[1:2]), cex=0.7, font=3)
```


A Topographic Map of Maunga Whau
10 Meter Contour Spacing
Meters West
Meters North
0
100
200
300
400
500
600
0
200
400
600
800
1000
110
120
110
100
160
150
190
180
170
160
180
170
140
130
110
100
100
110

(ix) 条件图

```
## Conditioning plots
par(bg="cornsilk")
coplot(lat ~ long | depth, data = quakes, pch = 21, bg = "green3")
```


Given : depth
100
200
300
400
500
600
165
170
175
180
185
−10
−15
−20
−25
−30
−35
lat

(x) 3D 图

```
## (1) The Obligatory Mathematical surface.
##     Rotated sinc function.
x <- seq(-10, 10, length.out = 50)
y <- x
rotsinc <- function(x,y){
  sinc <- function(x) { y <- sin(x)/x ; y[is.na(y)] <- 1; y }
  10 * sinc( sqrt(x^2+y^2) )
}
sinc.exp <- expression(z == Sinc(sqrt(x^2+y^2)))
z <- outer(x, y, rotsinc)
oldpar <- par(bg = "white")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
title(sub=".")## work around persp+plotmath bug
title(main = sinc.exp)

persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue",
      ltheta = 120, shade = 0.75, ticktype = "detailed",
```

```
      xlab = "X", ylab = "Y", zlab = "Z")
title(sub=".")## work around persp+plotmath bug
title(main = sinc.exp)
```