Lecture 1: R基础以及散点图和回归

张伟平

Thursday 4$^{\text{th}}$ March, 2010

Contents

Chapter 1

Start with R

R 是一个免费开源的用于统计计算和作图的语言和软件环境. 支持的操作系统包括各种UNIX平台, Windows 和MacOS 等. R 提供了广泛的统计工具(线性和非线性建模, 经典的统计检验, 时间序列, 生存分析, 分类, 聚类... 等等)和灵活高质量的图形工具. R 的功能可以通过添加 **package** 来扩充.

R 是一种解释性程序语言, 因此不必像 C 或者 Fortran 之类的编译语言首先要构成一个完整的程序形式. 当 R 运行时, 所有变量, 数据, 函数及结果都以对象 (objects) 的形式存在计算机的活动内存中, 并冠有相应的名字代号. 我们可以通过用一些运算符(如算术，逻辑，比较等) 和一些函数(其本身也是对象) 来对这些对象进行操作.

在R中,一个统计分析一般分几步完成, 中间的结果都是存储在对象里. 因此在回归分析时, SAS和SPSS会给出非常多的输出结果, 但R 将给出最少的输出, 而将结果存储在一个拟合对象中, 可以使用R函数进行后续的分析.

R 语言中最简单的命令莫过于通过输入一个对象的名字来显示其内容了.

例如, 一个名为 n 的对象, 其内容是数值10:

↑Example

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

↓Example

方括号中的数字1表示从 n 的第一个元素开始显示.

↑Example

```
> n<-1:30
> n
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[21] 21 22 23 24 25 26 27 28 29 30
```

↓Example

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

R有一个内建的帮助工具以获得指定名称函数的更多信息. 例如, 函数solve, 使用

```
> help(solve)
```

↑Code ↓Code

或者

```
> ?solve
```

↑Code ↓Code

对一些特殊的字符或者有着语法意义的词汇(如if,for,function等), 则必须将其放在双引号或者单引号中:

```
> help("[[") ; help("if")
```

↑Code ↓Code

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

↑Example

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

↓Example

也可以启动R的HTML格式帮助文档

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

↑Code

↓Code

help.search()(等价的??)命令给出更多查找帮助的方式.另外, 也可以使用example()命令查看某指定的主题. 例如

```
> help.search("tree") 等价地, ??mean
```

↑Code

↓Code

会列出所有在帮助页面含有"tree"的函数. 注意如果有一些包是最近才安装的, 应该首先使用函数help.search中的rebuild选项来刷新数据库(e.g., help.search("tree", rebuild = TRUE)). 而

```
> example(mean); example(InsectSprays)等
```

↑Code ↓Code

会启动相应主题的范例.

使用函数 apropos 能在所有被载入内存的包中找出所有名字含有指定字符串的函数, 例如

↑Example

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

↓Example

如果R命令是放在一个文件(例如commands.R)里,则可以使用

↑Code

 脆

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

↓Code

在Windows下也可以使用文件菜单运行此文件.

命令可以使用分号(‘;’)来隔开, 或者使用一个新行. 基本命令可以成组的放在花括号(‘{’ 和 ‘}’)之间. 注释符号(‘#’)可以从任何地方开始, 表示其后一直到该行结束部分被注释掉. 如果一个命令没有完成, R将会在下续的第二行开始处给出一个提示符

```
+
```

表示继续读入命令, 直到该命令语法完整为止.

命令 objects 与 ls 列出当前 R 进程(内存)中的所有对象名称例如

↑Example

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

↓Example

如果只要显示出在名称中带有某个指定字符的对象，则通过设定选项 pattern 来实现(可简写为 pat):

↑Example

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

↓Example

如果进一步限为显示在名称中以某个字母开头的对象，则可:

↑Example

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

↓Example

rm 命令可以将指定的对象从内存中删除. rm(x)删除名为x的对象, rm(x,y)删除名为x和y的对象,而rm(list=ls(all=TRUE))将删除当前内存中的所有对象.

Chapter 2
Data with R

2.1 Objects

R中常用的对象(objects)包括向量(vector), 因子(factor), 数组(array), 矩阵(matrix), 数据框(data frame), 时间序列(ts), 列表(list)等等. 所有的对象都有两个内在属性: 类型(mode)和长度(length). 类型是对象元素的基本种类, 常用的有四种: 数值型(numeric), 字符型(character), 复数型(complex)和逻辑型(logical)(FALSE或TRUE). 虽然也存在其它的类型(raw), 但是并不能用来表示数据, 例如函数或表达式; 长度是对象中元素的数目. 对象的类型和长度可以分别通过函数mode和length得到. 例如

↑Example

```
> x <- 1
> mode(x)
[1] "numeric"
```

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

↓Example

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

↑Example

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

↓Example

当前工作空间中的所有对象名称可以用函数objects()来查看.

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

↑Example

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

↓Example

下表给出了表示数据的对象的类别概览

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

向量是一个变量, 其意思也即人们通常认为的那样; 因子是一个分类变量; 数组是一个k维的数据表; 矩阵是数组的一个特例, 其维数k = 2. 注意, 数组或者矩阵中的所有元素都必须是同一种类型的; 数据框是由一个或几个向量和(或)因子构成, 它们必须是等长的, 但可以是不同的数据类型; “ts”表示时间序列数据, 它包含一些额外的属性, 例如频率和时间; 列表可以包含任何类型的对象, 包括列表! 对于一个向量, 用它的类型和长度足够描述数据; 而对其它的对象则另需一些额外信息, 这些信息由外在的属性(attribute)给出. 例如我们可以引用这些属性中的*dim*属性, 其是用来表示对象的维数, 比如一个2行2列的的矩阵, 它的dim属性是一对数值[2,2], 但是其长度是4.例如:

↑Example

```
> x
   C1 C2
R1  1  3
R2  2  4
> attributes(x)
$dim
[1] 2 2
$dimnames
$dimnames[[1]]
[1] "R1" "R2"
$dimnames[[2]]
[1] "C1" "C2"
> attributes(x)$dim
[1] 2 2
```

↓Example

类型之间可以通过as.*something*()形式的命令来转换. 例如:

↑Example

```
> z<-0:9
> digits<-as.character(z)
> digits
 [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
> as.numeric(digits)->x
> x
 [1] 0 1 2 3 4 5 6 7 8 9
> mode(x)
[1] "numeric"
```

↓Example

所有对象都有一个属性: 类*class*. 可以通过函数class来得到其类型. 这个特殊的属性被用来在R中进行面向对象的程序设计.

2.2 Reading data in a file

R使用工作目录(working directory)来完成读写文件. 可以使用命令 getwd()和setwd()来获得和设定工作目录, 例如

```
setwd("C:/data") 或者setwd("/home/paradis/R")
```

↑Code ↓Code

R 可以使用函数read.table(有好几种变体形式), scan 和read.fwf 来进行读取txt (ASCII)文件中的数据. 命令

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

↑Code ↓Code

从当前工作目录中的文件data.dat中读取数据, 创建一个名为mydata的类型为数据框的对象. 函数read.table有很多参数:

[使用?read.table来了解各个参数的含义]

函数scan比read.table更加灵活. 它们之间的一个区别在于scan可以指定读入变量的类型,例如

```

```

↑Code

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

↓Code

读取了文件data.dat中三个变量, 第一个是字符型变量, 后两个是数值型变量. 另一个重要的区别在于scan()可以用来创建不同的对象, 向量, 矩阵, 数据框, 列表... 缺省情况下, scan创建一个向量类型的对象.

函数read.fwf可以用来读取文件中一些固定宽度格式的数据

[使用?scan来了解各个参数]

[使用?read.fwf来了解各个参数]

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

↑Code

↓Code

除选项widths外,其余选项基本和read.table相同. 例如

↑Example

数据

```
A1.501.2
A1.551.3
B1.601.4
B1.651.5
```

```
> mydata <- read.fwf("data.txt", widths=c(1, 4, 3))
> mydata
  V1 V2 V3
1  1  A 1.50 1.2
2  2  A 1.55 1.3
```

```
C1.701.6    3 B 1.60 1.4
C1.751.7    4 B 1.65 1.5
            5 C 1.70 1.6
            6 C 1.75 1.7
```

↓Example

2.3 Saving data

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

↑Code

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

↓Code

若想用更简单的方法将一个对象(例如x, 可以是向量, 矩阵,或者数组)写入文件, 可以使用命令`write(x,file ="data.txt")`. 这里有两个选项:

- nc(或者ncol), 用来定义文件中的列数(在缺省情况下, 如果x是字符型数据, 则nc=1; 对于其它数据类型nc=5);

- append,用来决定是附加到文件data.txt内容后(TRUE), 还是替换掉原来已有的内容(FALSE).

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

2.4 generating data

生成一个向量(函数`c()`,算子`":"`,函数`seq()`等等:

↑Example

```
> x<-c(1,2,3,4,5)    > ch<-c("sa","ba")      > 1:5-1
> x                  > ch                      [1] 0 1 2 3 4
```

```
[1] 1 2 3 4 5
> y<-1:5
> y
[1] 1 2 3 4 5
```

```
[1] "sa" "ba"
> z<-seq(1,5)
> z
[1] 1 2 3 4 5
```

```
> 1:(5-1)
 [1] 1 2 3 4
```

↓Example

函数seq()功能强大:

[seq(length.out=0)的结果是什么?]

```
seq(from, to)                    #生成从from到to间隔为1的序列
seq(from, to, by= )              #生成从from到to间隔为by的序列
seq(from, to, length.out= )      #生成从from到to的长度为length.out的序列
seq(along.with= )                #生成序列1:length(along.with)
seq(from)                        #生成序列1:length(from)
seq(length.out= )                #生成序列1:length(length.out)
```

↑Code

↓Code

如果想用键盘输入一些数据也是可以的，只需要直接使用默认选项的scan函数, 以Ctrl+Z结束.

↑Example

```
> z <- scan()
1: 1.0 1.5 2.0 2.5
5:
Read 4 items
```

```
> z
[1] 1.0 1.5 2.0 2.5
```

↓Example

重复某个(些)值可以使用函数rep():

↑Example

```
> rep(1,3)
[1] 1 1 1
> rep(c(1,2),2)
[1] 1 2 1 2
> rep(1:3,1:3)
[1] 1 2 2 3 3 3
```

```
> rep(1:2,each=2)
 [1] 1 1 2 2
```

↓Example

生成规则的因子可以使用函数gl():

↑Example

```
> gl(2,3)
[1] 1 1 1 2 2 2
Levels: 1 2
> gl(2,2,label=c("F","M"))
[1] F F M M
Levels: F M
```

```
> gl(2,3,length=4)
[1] 1 1 1 2
Levels: 1 2
```

↓Example

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

↑Example

```
> expand.grid(h= c(60, 80), w=c(100),s= c("M","F"))
   h     w  s
1 60   100 M
2 80   100 M
3 60   100 F
4 80   100 F
```

产生随机序列可以使用R内建的各种函数:

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

大多数这种统计函数都有相似的形式, 只需用d、p或者q去替代r, 比如密度

函数(dfunc(x, ...)), 累计概率分布函数(也即分布函数)(pfunc(x, ...))和分位数函数(qfunc(p, ...), $0 < p < 1$).

↑Example

```
> pnorm(0)                    > qnorm(0.05)
[1] 0.5                       [1] -1.644854
> dnorm(0)                    > 1/sqrt(2*pi)
[1] 0.3989423                 [1] 0.3989423
```

↓Example

2.5 Manipulating objects

2.5.1 Creating objects

▷ *vector* 向量是R中的基本对象,前面我们已经介绍了使用函数c(),seq()等等创建向量. 向量可以是数值型,字符型和逻辑性.

```
vector(mode = "logical", length = 0) as.vector(x, mode = "any")
```

↑Code

```
is.vector(x, mode = "any")
```
↓Code

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

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

于此函数相关的函数包括ordered(同样功能, 为和S语言兼容), is.factor, is.ordered, as.factor 和as.ordered等. 函数levels可以查看一个因子对象的水平.

▷ *matrix* 一个矩阵实际上是有一个附加属性（维数dim）的向量

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

例如

↑Example

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

```
> x <- 1:6
> x
[1] 1 2 3 4 5 6
> dim(x)
NULL
> dim(x) <- c(2, 3)
> x
     [,1] [,2] [,3]
[1,] 1    3    5
[2,] 2    4    6
```

↓Example

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

↑Example

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

↓Example

▷ *array* 数据(array)是矩阵的多维推广.

↑Code

```
array(data = NA, dim = length(data), dimnames = NULL)
as.array(x)
is.array(x)
```

↓Code

例如

↑Example

```
> x<-array(1:2,dim=c(2,2,2))
> x
, , 1
     [,1] [,2]
[1,]    1    1
[2,]    2    2
, , 2
     [,1] [,2]
[1,]    1    1
[2,]    2    2
```

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

↓Example

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

↑Example

```
> L2<-list(A=x,B=as.character(y))
> L2
$A
[1] 1 2

$B
[1] "2" "3" "4"
```

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

> L2$A
[1] 1 2
```

↓Example

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

↑Example

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

↓Example

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

↑Example

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

↓Example

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

↑Code

↓Code

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

try ?ts for details.

2.5.2 Operators

R 中的赋值运算符号有“<-”,“<<-”和“=”. 例如赋值(value)到对象 x:

[三种赋值有什么区别?]

↑Code

```
x <- value
x <<- value
value -> x
value ->> x
```

```
x = value
```

↓Code

R 中的运算符如下:

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

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

逻辑型运算符适用于一个(对“!”运算符)或两个逻辑型对象(其它运算符), 并且返回一个(或几个)逻辑性变量. 运算符“逻辑与”和“逻辑或”存在两种

形式: “&”和“|”作用在对象中的每一个元素上并且返回和比较次数相等长度的逻辑值: “&&”和“||”只作用在对象的第一个元素上.

R 中的数学运算中“^”和“<-,-,=,<<-”为从右向左, 而其他运算都是从左到右. 例如

[R Language Definition]

$2\hat{}2\hat{}3 = 2^8 \neq 4^3$; $1 - 1 - 1 = -1 \neq 1$

↑Code

↓Code

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

↑Example

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

```
> x=1:2;y=1:3
> x==y
[1]  TRUE  TRUE FALSE
Warning message:
In x == y : longer object length is not
a multiple of shorter object length
```

```
[1] TRUE
```

↓Example

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

↑Example

```
> 0.9 == (1 - 0.1)
[1] TRUE
> identical(0.9, 1 - 0.1)
[1] TRUE
> all.equal(0.9, 1 - 0.1)
[1] TRUE
```

```
> 0.9 == (1.1 - 0.2)
[1] FALSE
> identical(0.9, 1.1 - 0.2)
[1] FALSE
> all.equal(0.9, 1.1 - 0.2)
[1] TRUE
```

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

↓Example

对字符型对象的运算常用的几个函数有substr()和paste()等. 例如

↑Example

```
> colors <- c("red", "yellow", "blue")
> more.colors <- c(colors, "green", "magenta", "cyan")
> substr(colors, 1, 2)
[1] "re" "ye" "bl"
> paste(colors, "flowers")
[1] "red flowers"    "yellow flowers" "blue flowers"
> paste("several ", colors, "s", sep="")
[1] "several reds"    "several yellows" "several blues"
> paste("I like", colors, collapse = ", ")
[1] "I like red, I like yellow, I like blue"
```

↓Example

2.5.3 Accessing the values of an object: the indexing system

下标系统可以用来有效、灵活且有选择性地访问一个对象中的元素; 下标可以是数值型的或逻辑型的. 举例来说, 对向量

↑Example

```
> x <- 1:5
> x[3]
[1] 3
> x[c(F,T)]
[1] 2 4
```

```
> x[c(1,3)]
[1] 1 3
> x[x>2]
[1] 3 4 5
> x[3] <- 20
```

```
> x[-2]
 [1] 1 3 4 5
> x[-c(1,3)]
 [1] 3 5
```

↓Example

当用一个非整数的数字访问向量某个元素是,小数部分就会被略去. 例如

↑Example

```
> x
[1] 1 2
> x[0.5]
```

```
numeric(0)
> x[1.5]
[1] 1
```

↓Example

对矩阵

↑Example

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

```
> x[, 2:3]
> x
> x[,2:3]
      [,1] [,2]
[1,]    3    5
[2,]    4    6
> x[, 3, drop = FALSE]
     [,1]
 [1,] 21
 [2,] 22
```

```
> x[, 3]
[1] 21 22
```

```
> x[-1,]
 [1] 2 4 22
```

↓Example

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

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

2.5.4 Accessing the values of an object with names

对象的元素也可以通过名字访问. 例如对向量

↑Example

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

```
> x
 a b c
 1 2 3
> x["b"]
 b
 2
```

↓Example

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

↑Example

```
> x<-matrix(1:4,2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> dimnames(x)
```

```
> rownames(x)<-c("r1","r2")
> colnames(x)<-c("c1","c2")
> x
   c1 c2
r1  1  3
r2  2  4
```

```
NULL
> dimnames(x)<-list(c("R1","R2"),c("C1","C2"))
> x                        > x["R2",]
   C1 C2                    [1] 2 4
R1  1  3
R2  2  4
```

↓Example

对于list对象, 我们可以使用*mylist$name*的形式访问其名为name的列表. 例如

↑Example

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

↓Example

2.5.5 Arithmetics and simple functions

向量是可以进行算术运算的,例如

↑Example

```
> x <- 1:4
> y <- rep(1, 4)
> z <- x + y
> z
[1] 2 3 4 5
> a <- 10
> z <- a * x
> z
[1] 10 20 30 40
```

```
> y <- 1:2
> z <- x + y
> z
[1] 2 4 4 6
> y <- 1:3
> z <- x + y
Warning message: In x + y : longer object length
 is not a multiple of shorter object length
> z
[1] 2 4 6 5
```

↓Example

R中用来处理数据的函数太多了而不能全部列在这里。读者可以找到所有的基本数学函数(log, exp, log10, log2, sin, cos, tan, asin, acos, atan, abs,

sqrt, . . .), 专业函数(gamma, digamma, beta, besselI, . . .), 同样包括各种统计学中有用的函数. 下表中列出了一部分函数:

函数	意义
`sum(x)`	对`x`中的元素求和
`prod(x)`	对`x`中的元素求连乘积
`max(x)`	`x`中元素的最大值
`min(x)`	`x`中元素的最小值
`which.max(x)`	返回`x`中最大元素的下标
`which.min(x)`	返回`x`中最小元素的下标
`range(x)`	与`c(min(x), max(x))`作用相同
`length(x)`	`x`中元素的数目
`mean(x)`	`x`中元素的均值
`median(x)`	`x`中元素的中位数
`var(x) or cov(x)`	`x`中元素的的样本方差; 如果`x`是一个矩阵或者一个数据框, 将计算协方差阵
`cor(x)`	如果`x`是一个矩阵或者一个数据框则计算相关系数矩阵(如果`x`是一个向量则结果是`1`)

var(x, y) or cov(x, y)	x和y的协方差，如果是矩阵或数据框则计算x和y对应列的协方差
cor(x, y)	x和y的线性相关系数，如果是矩阵或者数据框则计算相关系数矩阵

下面的函数返回更复杂的结果

round(x, n)	将x中的元素四舍五入到小数点后n位
rev(x)	对x中的元素取逆序
sort(x)	将x中的元素按升序排列；要按降序排列可以用命令rev(sort(x))
rank(x)	返回x中元素的秩
log(x, base)	计算以base为底的x的对数值
scale(x)	如果x是一个矩阵， 则中心化和标准化数据；若只进行中心化则使用选项scale=FALSE，只进行标准化则center=FALSE(缺省值是center=TRUE, scale=TRUE)
pmin(x,y,...)	逐对比较x和y的分量，返回一个向量，它的第i个元素是x[i], y[i],...中最小值

pmax(x,y,...)	同上，取最大值
cumsum(x)	返回一个向量，它的第i个元素是从x[1]到x[i]的和
cumprod(x)	同上，取乘积
cummin(x)	同上，取最小值
cummax(x)	同上，取最大值
match(x, y)	返回一个和x的长度相同的向量，表示x中与y中元素相同的元素在y中的位置(没有则返回NA)
which(x == a)	返回一个包含x符合条件(当比较运算结果为真的下标的向量，在这个结果向量中数值i说明x[i] == a
choose(n, k)	计算从n个样本中选取k个的组合数
na.omit(x)	忽略有缺失值(NA)的观察数据(如果x是矩阵或数据框则忽略相应的行)
na.fail(x)	如果x包含至少一个NA则返回一个错误消息
unique(x)	如果x是一个向量或者数据框，则返回一个类似的对象但是去掉所有重复的元素(对于重复的元素只取一个)
table(x)	返回一个表格,给出x中重复元素的个数列表(尤其对于整数型或者因子型变量)

`table(x, y)`	x与y的列联表
`subset(x, ...)`	返回x中的一个满足特定条件...的子集，该条件通常是进行比较运算：`x$V1 < 10`；如果x是数据框，选项`select`给出要保留的变量（或者用负号表示去掉）
`sample(x, size)`	从x中无放回抽取`size`个样本，选项`replace = TRUE`表示有放回的抽样

2.5.6 Matrix Computation

在矩阵运算中, 有如下几个常用的运算(函数):

`%*%` 矩阵相乘 `t(A)` 矩阵转置 `eigen(A)` 矩阵A的特征值(向量)
`solve(A)`矩阵求逆(`solve(A,b)`用于解线性方程组`AX=b`)

Definition

例如

[A*B,A/B表示什么意思呢?]

个Example

```
> A<-matrix(1:4,2,2)
```

```
> eigen(A)
```

```
> B<-diag(2)
 >A%*%B
      [,1] [,2]
 [1,]    1    3
 [2,]    2    4
> solve(A)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> dim(A)
[1] 2 2
```

```
$values
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

↓Example

在执行形如`t(A)%*%B`的运算时,更有效率的方式是使用函数crossprod(A,B) [为什么?]
其他的矩阵运算函数包括%x% Kronecker乘积, cbind(), rbind(), dim(), diag(), nrow(), ncol(), lower.tri(), upper.tri(), qr(), svd().... 例如

↑Example

```
> A<-cbind(1:2,3:4)
```

```
> A<-rbind(c(1,3),c(2,4))
```

```
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

↓Example

函数lower.tri和upper.tri用来访问矩阵的下三角元素以及上三角元素,例如

↑Example

```
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> B<-A
> B[lower.tri(A,diag=T)]<-0
> B
     [,1] [,2] [,3]
```

```
> lower.tri(A,diag=T)
      [,1]  [,2]  [,3]
[1,] TRUE FALSE FALSE
[2,] TRUE  TRUE FALSE
[3,] TRUE  TRUE  TRUE
> B[lower.tri(A)]<-0
> B
     [,1] [,2] [,3]
[1,]    0    4    7
```

```
[1,]    0    4    7                    [2,]    0    0    8
[2,]    0    0    8                    [3,]    0    0    0
[3,]    0    0    0
```

↓Example

奇异值分解: $A = UDV^T$, 其中矩阵U和T为正交矩阵, D为对角阵且D^2的对角元为矩阵A^TA的特征根. R 中使用函数svd()进行奇异值分解:

↑Example

```
> svd(A)->A.svd
> A.svd
$d
[1] 1.684810e+01 1.068370e+00 3.069525e-16

$u
           [,1]       [,2]       [,3]
[1,] -0.4796712  0.77669099  0.4082483
[2,] -0.5723678  0.07568647 -0.8164966
```

```
[3,] -0.6650644 -0.62531805  0.4082483

$v
           [,1]       [,2]       [,3]
[1,] -0.2148372 -0.8872307 -0.4082483
[2,] -0.5205874 -0.2496440  0.8164966
[3,] -0.8263375  0.3879428 -0.4082483

> A.svd$u%*%diag(A.svd$d)%*%t(A.svd$v)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

↓Example

由奇异值分解的性质, 当矩阵A为可逆方阵时, 很容易计算它的逆矩阵: $A^{-1} = VD^{-1}U^T$.

Choleski 分解: 若方阵$A > 0$, 则存在一个上三角阵U,使得$A = U^TU$. R中使用函数chol() 对方阵A作Choleski分解:

↑Example

```
> A                    > chol(A)
     [,1] [,2] [,3]          [,1] [,2] [,3]
[1,]    1    2    3    [1,]    1    2    3
[2,]    2    5    6    [2,]    0    1    0
[3,]    3    6   10    [3,]    0    0    1
```

↓Example

有了Choleski分解后, 我们就可以很方便的计算正定阵A的逆: $A^{-1} = U^{-1}(U^{-1})^T$. 这种计算比直接使用Gauss消元法计算$A^{-1}$要稳定的多. 也可以用于线性方程组求解.

QR分解: $A = QR$, 其中Q为一正交阵, R为一上三角阵. 例如

↑Example

```
> qr.A<-qr(A)
> qr.A
$qr
          [,1]       [,2]       [,3]
```

```
[1,] -3.7416574 -8.0178373 -12.0267559
[2,]  0.5345225 -0.8451543   0.5070926
[3,]  0.8017837 -0.4001484   0.3162278
$rank
[1] 3
$qraux
[1] 1.2672612 1.9164504 0.3162278
$pivot
[1] 1 2 3
attr(,"class")
[1] "qr"
```

↓Example

此输出为一个qr类. 使用函数qr.Q() 和qr.R() 对此qr类以得到矩阵Q和R:

↑Example

```
> Q<-qr.Q(qr.A)
> R<-qr.R(qr.A)
```

```
> Q%*%R
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    5    6
[3,]    3    6   10
```

↓Example

矩阵的条件数: 函数kappa() 可以用来计算矩阵的条件数.

↑Example

```
> kappa(A)
[1] 213.1021
```

↓Example

外积: 函数outer()在计算两个向量的外积时很有用.

↑Example

```
> x<-1:5
```

```
> outer(x,x,"*")
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    2    4    6    8   10
[3,]    3    6    9   12   15
[4,]    4    8   12   16   20
[5,]    5   10   15   20   25
```

↓Example

在对矩阵进行行或者列操作中, 函数apply()也是很常用. 比如计算对矩阵A按列计算均值:

↑Example

```
> apply(A,2,mean)
[1] 2.000000 4.333333 6.333333
```

↓Example

Chapter 3
Graphics with R

绘图函数的工作方式与前面描述的工作方式大为不同, 不能把绘图函数的结果赋给一个对象, 其结果直接输出到一个“绘图设备”上. 绘图设备是一个绘图的窗口或是一个文件. 有两种绘图函数: 高级绘图函数(high-level plotting functions)创建一个新的图形, 低级绘图函数(low-level plotting functions)在现存的图形上添加元素. 绘图参数(graphical parameters)控制绘图选项, 可以使用缺省值或者用函数par修改.

3.1 Managing graphics

当绘图函数开始执行, 如果没有打开绘图设备, 那么R 将打开一个绘图窗口来展示这个图形. 可用的绘图设备种类取决于操作系统. 在Unix/Linux 下, 绘图窗口称为x11, 而在Windows 下称为windows. 但是绘图窗口都可以用x11()命令打开, windows 下还可以用windows()命令打开.

可以用函数打开一个文件作为绘图设备, 这包括：postscript(), pdf(), png(), jpeg(),..., 可用的绘图设备列表可以用?device来察看. 最后打开的设备将成为当前的绘图设备, 随后的所有图形都在这上面显示. 函数dev.list() 显示打开的列表.

↑Example

```
> x11(); x11(); pdf()
> dev.list()
X11 X11 pdf
2 3 4
> dev.cur()#查看当前设备
pdf
4
> dev.set(3)#设置设备3为当前设备
X11
3
> dev.off(2) #关闭设备2
X11
3
> dev.off()  #关闭当前设备
pdf
4
> graphics.off() #
```

↓Example

当要把多个绘图绘制在同一个绘图设备上时, 可以使用如下方法:

▽ 使用函数**split.screen** 方法, 可以单独控制每个分割出的screen.

↑Code

```
split.screen(figs, screen, erase = TRUE)
screen(n = , new = TRUE)
erase.screen(n = )
close.screen(n, all.screens = FALSE)
```

↓Code

例如

↑Example

```
 split.screen(c(2,1)) #分割显示屏幕为2个,格式为2行1列
[1] 1 2               #分割出的屏幕编号
> split.screen(c(1,3), screen = 2) # 分割第2个显示屏为3个, 1行3列
[1] 3 4 5
> screen(1)           #使用屏幕1
> plot(10:1)          #绘制图形
> screen(4)           #使用屏幕4
```

```
> plot(10:1)            #绘制图形
> close.screen(all = TRUE) #关闭如上屏幕分割定义
```

↓Example

▽ 使用函数**layout** 方法, 把当前的图形窗口分割为多个部份, 图形将一次显示在各部分中.

```
layout(mat, widths = rep(1, ncol(mat)),
        heights = rep(1, nrow(mat)), respect = FALSE)
layout.show(n = 1)
```

↑Code ↓Code

例如

↑Example

```
def.par <- par(no.readonly = TRUE) # 存储当前图形设备设置
layout(matrix(c(1,1,0,2), 2, 2, byrow = TRUE))
#fig1占第一行,fig2占第二行的第二列
```

```
layout.show(2) # 显示每个fig的区域
plot(1:10)
plot(10:1)
par(def.par) #恢复默认设置
```

↓Example

▽ 使用函数**par** 中的参数**mfrow/mfcol** 方法函数par() 中的参数mfrow 或者mfcol 可以用来指定将当前图形窗口分割为几行几列.

```
par(mfrow=c(nr,nc)) # 将图形窗口分为nr行nc列
```

↑Code

↓Code

这种方法都是平均分割,不能像layout那样可以控制每个区域的大小.

3.1.1 Graphical Functions

下面是R中高级绘图函数的概括:

↑Code

`plot(x)`	以x的元素值为纵坐标、以序号为横坐标绘图
`plot(x, y)`	x(在x-轴上)与y(在y-轴上)的二元作图
`pie(x)`	饼图
`boxplot(x)`	盒形图(box-and-whiskers)
`pairs(x)`	如果x是矩阵或是数据框，作x的各列之间的二元图
`hist(x)`	x的频率直方图
`barplot(x)`	x的值的条形图
`qqnorm(x)`	正态分位数－分位数图
`qqplot(x, y)`	y对x的分位数－分位数图

`contour(x, y, z)`	等高线图(画曲线时用内插补充空白的值)，x和y必须为向量，z必须为矩阵，使得 `dim(z)=c(length(x),length(y))` (x和y可以省略)
`filled.contour(x,y, z)`	同上，等高线之间的区域是彩色的，并且绘制彩色对应的值的图例
`image(x, y, z)`	同上，但是实际数据大小用不同色彩表示
`persp(x, y, z)`	同上，但为透视图

↓Code

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

↑Code

`axes=TRUE`	如果是`FALSE`，不绘制轴与边框
`type="p"`	指定图形的类型，`"p"`：点，`"l"`：线，`"b"`：点连线， `"o"`：同上，但是线在点上，`"h"`：垂直线， `"s"`：阶梯式，先水平再垂直， `"S"`：同上，先垂直再水平
`xlim=, ylim=`	指定轴的上下限， 例如`xlim=c(1, 10)`或者`xlim=range(x)`
`xlab=, ylab=`	坐标轴的标签，必须是字符型值
`main=`	主标题，必须是字符型值
`sub=`	副标题（用小字体）

↓Code

3.1.2 Low-level plotting commands

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

↑Code

`points(x, y)`	添加点图（可以使用选项`type=`）

`lines(x, y)`	同上，但是添加线
`text(x, y, labels, ...)`	在(x,y)处添加用labels指定的文字；典型的用法是：`plot(x, y, type="n"); text(x, y, names)`
`mtext(text, side=3, line=0, ...)`	在边空添加用text指定的文字，用side指定添加到哪一边(参数axis())；line指定添加的文字距离绘图区域的行数
`segments(x0, y0, x1, y1)`	从(x0,y0)各点到(x1,y1)各点画线段
`arrows(x0, y0, x1, y1, angle= 30, code=2)`	同上但加画箭头， 如果code=2则在各(x0,y0)处画箭头， 如果code=1则在各(x1,y1)处画箭头，如果code=3则在两端都画箭头；angle控制箭头轴到箭头边的角度

abline(a,b)	绘制斜率为b和截距为a的直线
abline(h=y)	在纵坐标y处画水平线
abline(v=x)	在横坐标x处画垂直线
abline(lm.obj)	画由lm.obj确定的回归线
rect(x1, y1, x2, y2)	绘制长方形，(x1, y1)为左下角，(x2,y2)为右上角
polygon(x, y)	绘制连接各x,y坐标确定的点的多边形
legend(x, y,legend)	在点(x,y)处添加图例，说明内容由legend给定
title()	添加标题，也可添加一个副标题
axis(side, vect)	画坐标轴，side=1时画在下边，side=2时画在左边，side=3时画在上边，side=4时画在右边.

可选参数at指定画刻度线的位置坐标

box()	在当前的图上加上边框
rug(x)	在x-轴上用短线画出x数据的位置
locator(n, type="n", ...)	在用户用鼠标在图上点击n次后返回n次点击的坐标(x; y)；并可以在点击处绘制符号(type="p"时)或连线(type="l"时)，缺省情况下不画符号或连线

↓Code

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

[用?demo (plotmath) 查看更多数学符号和公式命令]

↑Code

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

↓Code

在图中相应坐标点(x; y)处显示下面的方程:

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

3.1.3 Graphical Parameters

除了低级作图命令之外, 图形的显示也可以用绘图参数来改良. 绘图参数可以作为图形函数的选项(但不是所有参数都可以这样用), 也可以用函数par 来永久地改变绘图参数, 也就是说后来的图形都将按照par 指定的参数来绘制. 比如命令par(bg="yellow") 将使得致后来的图形都以黄色的背景来绘制. 总共有73个绘图参数, 其中一些有非常相似的功能. 这些参数详细的列表可以参阅?par; 下面的我们只列举了最常用的参数.

↑Code

`adj`	控制 `text`, `mtext` 和 `title` 中文字的对齐方式, `0`是左对齐, `0.5`是居中对齐, `1`是右对齐, 值`>` `1`时对齐位置在文本右边的地方, 取负值时对齐位置在文本左边的地方; 如果给出两个值(例如`c(0, 0)`), 第二个只控制关于文字基线的垂直调整

bg	指定背景色(例如bg="red", bg="blue"; 用colors()可以显示657种可用的颜色名)
bty	控制图形边框形状，可用的值为："o", "l", "7", "c", "u" 和"]" (边框和字符的外表相像)；如果bty="n"则不绘制边框
cex	控制缺省状态下符号和文字大小的值；另外，cex.axis 控制坐标轴刻度数字大小，cex.lab 控制坐标轴标签文字大小，cex.main 控制标题文字大小，cex.sub控制副标题文字大小
col	控制符号的颜色；和cex类似，还可用：col.axis, col.lab, col.main, col.sub
font	控制文字字体的整数(1: 正常，2: 斜体，3: 粗体，4: 粗斜体)；和cex类似，还可用：font.axis, font.lab,

font.main, font.sub

las 控制坐标轴刻度数字标记方向的整数(0：平行于轴，1：横排，2：垂直于轴，3：竖排）

lty 控制连线的线型，可以是整数(1：实线，2：虚线，3：点线，4：点虚线，5：长虚线，6：双虚线)，或者是不超过8个字符的字符串(字符为从"0"到"9"之间的数字)交替地指定线和空白的长度，单位为磅(points)或象素，例如lty="44"和lty=2效果相同

lwd 控制连线宽度的数字

mar 控制图形边空的有4个值的向量 c(bottom, left, top, right)，缺省值为c(5.1, 4.1, 4.1, 2.1)

mfcol c(nr,nc)的向量，分割绘图窗口为nr行nc列的矩阵布局，按列次序使用各子窗口

mfrow	同上，但是按行次序使用各子窗口
pch	控制符号的类型，可以是1到25的整数，也可以是""里的单个字符
ps	控制文字大小的整数，单位为磅(points)
pty	指定绘图区域类型的字符，"s"：正方形，"m":最大利用
tck	指定轴上刻度长度的值，单位是百分比，以图形宽、高中最小一个作为基数；如果tck=1则绘制grid
tcl	同上，但以文本行高度为基数（缺省下tcl=-0.5）
xaxt	如果xaxt="n"则设置x-轴但不显示（有助于和axis(side=1, ...)联合使用）
yaxt	如果yaxt="n"则设置y-轴但不显示（有助于和axis(side=2, ...)

联合使用）

↓Code

我们可以是如下代码查看参数pch 的各个形状:

↑Code

```
plot(rep(1,10),ylim=c(-2,1.2),pch=1:10,cex=3,axes=F,xlab="",ylab="")
text(rep(0.6,10),as.character(1:10))
points(rep(0,10),pch=11:20,cex=3)
text(rep(-0.4,10),as.character(11:20))
points(rep(-0.8,5),pch=21:25,cex=3)
text(rep(-1.2,5),as.character(21:25))
points(6:10, rep(-0.8,5),pch=c("*","?","X","x","&"),cex=3)
text(6:10,rep(-1.2,5),c("*","?","X","x","&"))
```

↓Code

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

↑Code

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

↓Code

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

色这样的安排。命令如下

↑Code

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

和以前一样, 先保存缺省的绘图参数, 然后修改背景颜色和边空. 画图时用type=”n”不画出点, 用xlab=””, ylab=””不画坐标轴标签, 和用xaxt=”n”, yaxt=”n”不画坐标轴. 这样只画了绘图区域的边框, 并用xlim和ylim规定了坐标轴范围. 注意, 我们可以用选项axes=FALSE, 但这样的话不仅不画坐标轴, 而且也不画边框. 然后, 用低级图形函数在上面确定的坐标区域内加入各种图形元素. 在添加点以前, 用rect()修改绘图区域的颜色：长方形大小选得比绘图区域大得多.

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

用title()添加标题, 但是字体稍微改变了. 开始的两个边空文字函数mtext()调用画坐标轴的标签。这个函数的第一自变量是要画的文本. 选项line指出到绘图区域的距离行数（缺省时line=0）, at给出坐标. 第二次调用mtext(), 调用利用了side (3)的缺省值. 另外两个mtext()用数值型向量作第一自变量, 会自动转换为字符型.

Chapter 4
Statistical Analysis with R

本章我们的目的是对R的统计分析功能进行一个粗略而又系统的介绍.

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

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

An Example of Linear Regression

在stats 包里面的线性回归分析函数是lm. 为了示范这个函数, 我们采用R 分发的数据集: women

↑Code

```
> data(women)
> lm.wm<-lm(weight~height,data=women)
```

↓Code

函数lm 的主要参数(必要的)是一个公式. 公式的左边是响应变量, 右边是预测变量, 二者通过"~" 连接. 可选项data = women 表明这些变量是在数据框women 中.

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

↑Example

```
> print(lm.wm) # 和直接 lm.wm 等价
```

```
Call:
lm(formula = weight ~ height, data = women)

Coefficients:
(Intercept)       height
     -87.52         3.45

> summary(lm.wm)

Call:
lm(formula = weight ~ height, data = women)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7333 -1.1333 -0.3833  0.7417  3.1167

Coefficients:
```

```
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -87.51667    5.93694  -14.74 1.71e-09 ***
height        3.45000    0.09114   37.85 1.09e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.525 on 13 degrees of freedom
Multiple R-Squared: 0.991,      Adjusted R-squared: 0.9903
F-statistic:  1433 on 1 and 13 DF,  p-value: 1.091e-14
```

↓Example

可以通过函数plot() 展示分析结果的统计图.

4.1 Formulae

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

a+b	a 和 b 的相加效应
X	如果 X 是一个矩阵，这将反映各列的相加效应，即 X[,1]+X[,2]+...+X[,ncol(X)]；还可以通过索引向量选择特定列进行分析(如，X[,2:4])
a:b	a 和 b 的交互效应
a*b	相加和交互效应(等价于a+b+a:b)
poly(a, n)	a的n价(正交)多项式
^n	包含所有的直到n阶的交互作用，即(a+b+c)^2 等价于a+b+c+a:b+a:c+b:c
b %in% a	b 和 a 的嵌套分类设计(等价于a+a:b，或者a/b)
-b	去掉因子b的影响，如：(a+b+c)^2-a:b 等价

于a+b+c+a:c+b:c

-1 y~x-1 表示通过原点的线性回归(等价于y~x+0 或者0+y~x)

1 y~1 拟合一个没有因子影响的模型(仅仅是截距)

offset(...) 向模型中增加一个影响因子但不估计任何参数(如, offset(3*x))

显然, 公式里面的运算符和表达式里面使用的运算符有着不同的含义. 例如, 公式y~x1+x2 表示模型$y = x_1\beta_1 + x_2\beta_2 + e$. 为了在公式里面使用常规的运算符, 我们可以使用函数I(), 例如想要表示模型$y = x^2 + e$, 则表达式为y~I(x^2) (这里和Splus里设定不同).

4.2 Generic Functions

和许多统计编程语言不同的是, R 函数将输入对象的属性作为输入参数. 类是最应该关注的的一个属性. R 统计函数常常返回一个类名与函数名相同的对

象(如lm 返回类"lm"的对象, aov 返回类"aov"的对象). 我们用来解析结果的函数对特定的类对象有特定的行为. 这些函数被称为泛型(generic).

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

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

↑Code

`print`	返回简单的汇总信息
`summary`	返回较为详细的汇总信息
`df.residual`	返回残差的自由度
`coef`	返回被估计的系数(有时还包括他们的标准差)
`residuals`	返回残差
`deviance`	返回一个模型的残差平方和

`fitted`	返回拟合值
`logLik`	计算对数似然值和返回参数数目
`AIC`	计算`Akaike` 信息准则(`Akaike information criterion, AIC`)(依赖于`logLik()`)

↓Code

像lm 或者aov 之类的函数返回一个保存各分析结果的列表, 我们可以使用函数str 来查看该对象的结构. 也可以使用函数names 来查看该列表对象各个元素的名字. 例如

↑Example

```
> names(lm.wm)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

↓Example

这些元素可以通过例如`lm.wm$coef` 的形式提取.

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

↑Code

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

↓Code

4.3 Packages

在R 启动后, 可以使用search() 来显示当前载入的package名称. 例如

↑Example

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

↓Example

其他包需要在载入后才能使用. 比如载入"grid"包:

↑Example

```
> library(grid)
```

↓Example

还可以用函数data()来载入制定的数据.

R的基本包包括

↑Example

包	描述
base	基本R函数
datasets	基本R数据集
grDevices	基本的或grid图形的设备函数
graphics	基本图形函数
grid	grid图形
methods	用于R对象和编程工具的方法和类的定义
splines	样条回归函数和类

stats	统计函数
stats4	基于S4标准定义的统计函数
tcltk	R 和Tcl/Tk 图形接口元素的交互函数
tools	包开发和管理的工具
utils	R 工具函数

↓Example

除了R 环境的基本包外, 还有一些和R捆绑在一起发行的推荐包, 比如

↑Example

包	描述
boot	抽样和bootstraping 方法
class	分类方法
cluster	聚类方法
foreign	读取各种格式(S3, Stata, SAS, Minitab, SPSS, Epi Info)的外部数据
KernSmooth	核密度拟合方法(包括双变量核)
lattice	grid 图
MASS	Venables & Ripley 著的 "Modern Applied

	Statistics with S"中的配套库，包含很多有用的函数,工具和数据集
mgcv	广义的可加模型
nlme	线性和非线性混合效应模型
nnet	神经网络和多项对数线性模型
rpart	递归分割
spatial	空间分析("kriging", 空间协方差, ...)
survival	生存分析

↓Example

等等.

查看一个包内可用的函数除了可以使用在线帮助文档外, 还可以使用library(help=*package*)的形式. 比如

↑Example

```
>libraray(help=boot)
```

↓Example

和包管理和安装有关的几个常用函数如installed.packages, CRAN.packages, 或者download.packages. 而可以使用update.packages() 对已经安装的包进行更新.

Chapter 5

Programming with R

5.1 Flow Control

- **For 循环**

  函数For() 在R 程序设计中很常见, 用以实现命令组的循环.

```
for (变量名in 取值向量) {命令组}
```

↑Code
↓Code

比如我们要创建一个Fibonacci序列前20个值的向量:

$$F_0 = 0 \qquad F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, n \geq 2$$

则可以利用如上性质, 使用for 循环创建:

↑Example

```
> Fib<-rep(0,20)
> Fib[2]<-1
> for(i in 3:20) Fib[i]<-Fib[i-1]+Fib[i-2]
> Fib
 [1]    0    1    1    2    3    5    8   13
 [9]   21   34   55   89  144  233  377  610
[17]  987 1597 2584 4181
```

↓Example

- **While 循环**

`while` (条件) 命令组

↑Code
↓Code

使用while 循环重写上述例子

↑Example

```
> Fib<-rep(0,20)
> Fib[2]<-1
```

```
> i<-3
> while(i <=20) { Fib[i]<-Fib[i-1]+Fib[i-2]; i<-i+1}
```

↓Example

- **if 语句**

↑Code

```
if (条件) {命令组(若条件为真)}
if (条件) {命令组(若条件为真)} else {命令组(若条件为假)}
```

↓Code

一个简单的例子

↑Example

```
> x <- 3
> if (x > 2) y <- 2 * x else y <- 3 * x
```

↓Example

if... else 还可以再深入下去if...else if...else...

- **repeat循环, 以及break 和next 语句**

↑Code

```
repeat {
    命令组
   if (条件) break
   }
```

↓Code

repeat循环使用的不太频繁, 使用repeat 重写while 循环中的例子:

↑Example

```
> Fib<-rep(0,20)
> Fib[2]<-1
> i<-3
>repeat { Fib[i]<-Fib[i-1]+Fib[i-2]; i<-i+1; if(i>20) break}
```

↓Example

5.2 Functions

我们可以是用function命令编写自己的函数,格式如下

↑Code

```
函数名<-function(变量1,变量2,...) {
        函数体
        return(结果变量)
         }
```

↓Code

比如我们要编写一个函数来计算自然数的阶乘:

↑Example

```
ft<-function(m){
    if(m==1) rlt<-1
    else rlt<-m*ft(m-1)
    return(rlt)
}
```

↓Example

一个函数的变量仅在该函数内部有效, 比如

↑Example

```
> f <- function() {
+ x <- 1
+ g() # g will have no effect on our local x
+ return(x)
+ }
> g <- function() {
+ x <- 2 # this changes g's local x, not the one in f
+ }
> f()
[1] 1
```

↓Example

Chapter 6

Scatterplots and Regression

回归是用于研究一个变量和其他一些变量(因变量/自变量,响应变量/解释变量)相依关系的一种统计方法。比如班级学生数目对和学生学业成绩有关联吗？等等。

6.1 Inheritance of Height

E.S. Pearson 在1893 - 1898 年间在英国收集了$n = 1375$位65岁以下母亲和18岁以上女儿的身高数据, Pearson and Lee (1903) 发表了此数据，我们以此数据来研究母亲身高和女儿身高之间的遗传关系。

↑Code

```
heights<-read.table("heights.txt",header=T)
attach(heights)
apply(heights,2,summary)
```

```
dim(heights)
rp<-apply(heights,2,table)
names(rp)
length(rp$Mheight)
length(rp[[2]])
a<-as.numeric(names(rp$Mheight))
b<-rp$Mheight
plot(a,b,type="h")
points(a,b)
plot(Mheight,Dheight,xlim=c(55,75),ylim=c(55,75),pch=20,cex=.3)
```

↓Code

可以看出

- 两组数据的范围基本上是相同的
- 每组数据都有大量的相同值
- 两个变量之间存在相依关系，可以更进一步由母亲身高为58，64，68英寸左右的女儿身高均值看出。

- 散点图有椭圆形状, 这意味着两者之间可以用简单线性回归模型来刻画.
- 散点图对发现离群点(separated points)也是有帮助的. 在本例中, 离群点指那些母亲身高很矮,很高,以及女儿身高很矮,很高的点. 这两种点在回归中有着不同的名称. $x-$轴上的极值点(母亲身高)常称为杠杆点(leverage points), 这种点对回归模型的拟合有着重要的作用; $y-$轴上的极值点(女儿身高)常称为异常点(outlier), 它们表示了其相依关系和其他点不同,因此这种点一般要在预处理中剔除掉.

↑Code

```
heights.mod<-heights+matrix(runif(1375*2,-0.5,0.5),1375,2)
attach(heights.mod)
plot(Mheight,Dheight,xlim=c(55,75),ylim=c(55,75),pch=20,cex=.3)

sel <- (57.5 < Mheight) & (Mheight <= 58.5) |
       (62.5 < Mheight) & (Mheight <= 63.5) |
       (67.5 < Mheight) & (Mheight <= 68.5)
plot(Mheight[sel],Dheight[sel],xlim=c(55,75),ylim=c(55,75),pch=20,
```

```
    cex=.3, xlab="Mheight",ylab="Dheight")
```

↓Code

6.2 Forbes Data

在1857年的一篇文章中,苏格兰物理学家James D. Forbes讨论了气压和水的沸点之间的关系. 他知道海拔高度可以由气压决定, 低气压意味着高海拔, 而气压可以由气压计来测定. 由于在19世纪中期,气压计是易碎的设备, Forbes关心的是否可以通过简单的测量水的沸点来确定气压. 他在阿尔卑斯山做了系列实验, 在不同位置测量了气压和沸点(华氏)数值. 沸点的测量值根据测量位置的空气温度和标准温度之间的差异进行了调整.

↑Code

```
data(forbes);attach(forbes)
plot(Temp,Pressure,xlab="Temperature",ylab="Pressure")
abline(lm(Pressure~Temp,data=forbes))
```

↓Code

从此散点图可以看出

- 观测点数据只有$n = 17$个, 散点图显示出非常强的线性性. 几乎所有的点都非常靠近拟合的直线. 这意味着在给定温度的条件下,压力的波动范围非常的小.

- 仔细观测还可以看出, 存在系统性误差. 除了没有拟合的点外,中间部分的点落在拟合线的下方, 两头的点落在拟合线的上方. 此可以由残差-温度图更清楚的看出.

- Forbes知道log(压力)和温度之间是线性关系的,因此对原数据压力做log_{10} 变换(无特定意义,Forbes采用此变换),再做散点图, 此时残差图不再有系统性趋势,拟合效果好很多.

↑Code

```
par(mfrow=c(1,2))
plot(Temp,Pressure,xlab="Temperature",ylab="Pressure")
m0 <- lm(Pressure~Temp)
```

```
abline(m0)
plot(Temp,residuals(m0), xlab="Temperature", ylab="Residuals")
abline(h=0,lty=2)

# after log transformation
plot(Temp,logb(Pressure,10),xlab="Temperature",ylab="log(Pressure)")
m0 <- lm(logb(Pressure,10)~Temp)
abline(m0)
plot(Temp,residuals(m0), xlab="Temperature", ylab="Residuals")
abline(h=0,lty=2)
```

↓Code

6.3 Length at Age for Smallmouth Bass

Smallmouth Bass(一种黑色的鲈鱼)是内陆湖常见的一种鱼. 在对鱼的不同群体生长情况研究时, 鱼的长度和年龄之间的相依关系可以用来比较不同喂养管理方式下鱼群的生长情况. wblake.txt 数据是在1991年收集了Minnesota 东

北部West Bearskin Lake $n = 439$ 条smallmouth bass 鱼的长度(mm), 年龄(年)和年轮圈的半径(mm)信息. 鱼的年龄可以通过数其身上的年轮圈来决定. 这种数据称为切面数据(cross-sectional data), 即所有数据是在同一时间测量获得.

↑Code

```
data(wblake);attach(wblake);
plot(Age,Length)
abline(lm(Length~Age))
lines(1:8,tapply(Length,Age,mean),lty=2)
```

↓Code

可以看出

- 年龄只取整数值,因此散点图表示了8种不同鱼群的长度分布情况.
- 最长的1岁鱼要比最短的4岁鱼长度长,因此知道鱼的年龄是不能准确预测其长度的.

6.4 Turkey Growth

Noll, Weibel, Cook,and Witmer, 1984年发表了一项关于火鸡生长的研究. turkey.txt记录了70栏火鸡的生长数据. 这些火鸡的饲料除添加剂蛋氨酸外完全相同, 其中有10栏没有添加剂, 蛋氨酸的来源有三种, 添加剂的剂量有4种, 按此对其他60栏分为12组,每组使用一种蛋氨酸和一种添加剂剂量, 响应变量是所有火鸡的平均体重增加量(Gain,克), 记录的变量有剂量(A,在饲料中的比例), 添加剂的来源(S), A-S下的栏数(m)和平均体重增加量在改组的标准差.

↑Code

```
plot(turkey$A[turkey$S==1],turkey$Gain[turkey$S==1],pch="O",
     xlab="Amount (percent of diet)", ylab="Weight gain (g)")
points(turkey$A[turkey$S==2],turkey$Gain[turkey$S==2],pch="+")
points(turkey$A[turkey$S==3],turkey$Gain[turkey$S==3],pch="*")
legend(.02,780,legend=c("1","2","3"),pch="o+*")} else {
plot(turkey$A,turkey$Gain,pch=turkey$ S, xlab="Amount (percent of diet)",
       ylab="Weight gain (g)")
legend(.02,780,legend=c("1","2","3"),pch=1:3,cex=.6)
```

↓Code

可以看出

- 火鸡的平均体重增加量随着添加剂剂量的增加而增加;
- 平均体重增加量和添加剂剂量之间呈非线性关系. 事实上,

$$E(Y|Dose = x) = \beta_0 + \beta_1[1 - exp(\beta_2 x)]$$

可以用来刻画解释二者之间的关系.

6.5 Scatterplot matrices

当解释变量有多个时, 一个散点图就不足以呈现多个变量直接的关系. 此时使用变量两两之间的散点图是一种发现关系的方法.

6.5.1 Fuel Consumption

这个数据fuel2001.txt的目的是研究美国50个州以及哥伦比亚特区在汽油消费差异, 特别是研究州汽油税对汽油消费的影响效果. 解释变量包括Drivers, FuelC, Income, Miles, Pop, Tax, State (意义如下). 由于州人口多,自然拥有

驾照的人数就越多, 消费的燃料也就越多, 为了具有可比性, 我们使用其相对于人口的比例.

Drivers	该州有驾照的人数
FuelC	道路使用的汽油(1000s of gal.)
Income	2000年的人均收入(1000$)
Miles	该州联邦资助的高速路里程数
Pop	2001年16岁以上的人口数2001
Tax	州汽油税(cents per gallon)
State	州名
Fuel	1000 ×Fuelc/Pop
Dlic	1000 ×Drivers/Pop
log(Miles)	Base-two logarithm of Miles

使用R pairs可以得到解释变量两两之间的散点图:

```
data(fuel2001)
fuel2001$Dlic <- 1000*fuel2001$Drivers/fuel2001$Pop
fuel2001$Fuel <- 1000*fuel2001$FuelC/fuel2001$Pop
fuel2001$Income <- fuel2001$Income/1000
fuel2001$logMiles <- logb(fuel2001$Miles,2)
names(fuel2001)
pairs(Fuel~Tax+Dlic+Income+logMiles,data=fuel2001,gap=0.4,cex.labels=1.5)
```

直观上的印象是汽油消费和汽油税是负相关的. 总的印象是汽油消费量和其他变量之间至多是弱相关的.

6.6 Mean function and variance function

在一个一般的$X-Y$散点图中, 我们感兴趣的是Y的分布如何随着X变动. 此分布的一个重要方面就是其均值函数

$$E(Y|X=x)=f(x)$$

前面几个例子中我们已经看到这种函数可以是线性的, 也可以是非线性的. 在身高遗传研究中, 我们可以使用

$$E(Dheight|Mheight=x)=\beta_0+\beta_1 x$$

来刻画这种关系. 使用最小二乘方法估计β_0和β_1. 图示如下(虚线为$\beta_0=0,\beta_1=1$):

↑Code

```
plot(Mheight,Dheight,bty="l",cex=.3,pch=20)
```

```
abline(0,1,lty=2)
abline(lm(Dheight~Mheight),lty=1)
```

↓Code

响应变量分布的另一重要特征是方差函数. 在前面这些例子中, 我们都假设$Var(Y|X=x)=\sigma^2$, 其中σ^2未知.

Anscombe (1973, anscombe.txt) 提供了一组人造数据以说明散点图在回归中的重要性. 在这种数据下,简单的回归模型

$$E(Y|X=x)=\beta_0+\beta_1 x$$

被用来拟合数据. 在这四组数据下, β_0, β_1的估计值是相同的. 但是在散点图中对各自的直观印象完全不同. **因此散点图(或者数据的可视化)是回归分析(数据分析)的第一步**!

↑Code

```
library(lattice)
attach(anscombe)
x <- c(anscombe$x1, anscombe$x4, anscombe$x1, anscombe$x1)
```

```
y <- c(anscombe$y3, anscombe$y4, anscombe$y1, anscombe$y2)
n <- c("(c)","(d)","(a)","(b)")
g <- if(is.null(version$language)){
    ordered(rep(n, rep(11,4)),levels=n)} else {
    factor(rep(n, rep(11,4)),levels=n,ordered=TRUE)}
xyplot(y~x|g, between=list(x=.32,y=.32), data=anscombe,
       panel = function(x,y) {panel.xyplot(x,y)
                              panel.lmline(x,y)})
```

↓Code