Lecture 2: R 基础(二)

张伟平

Monday 7th September, 2009

Contents

Chapter 1
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.

1.1 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() # 关闭所有绘图设备
```

↓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.

1.1.1 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)
`coplot(x~y|z)`	关于z的每个数值（或数值区间）绘制x与y的二元图
`matplot(x,y)`	二元图，其中x的第一列对应y的第一列，x的第二列对应y的第二列，依次类推.
`dotchart(x)`	如果x是数据框，作Cleveland点图（逐行逐列累加图）

`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
`stars(x)`	如果x是矩阵或者数据框，用星形和线段画出
`symbols(x, y, ...)`	在由x和y给定坐标画符号(圆，正方形，长方形，星，温度计式或者盒形图)，符号的类型、大小、颜色等由另外的变量指定
`termplot(mod.obj)`	回归模型（`mod.obj`）的（偏）影响图
`sunflowerplot(x,y)`	同上但是以相似坐标的点作为花朵，其花瓣数目为点的个数
`stripchart(x)`	把x的值画在一条线段上，样本量较小时可作为盒形图的替代

`interaction.plot(f1, f2, y)` 如果f1和f2是因子， 作y的均值图， 以f1的不同值作为x轴，而f2的不同值对应不同曲线；可以用选项fun指定y的其他的统计量(缺省计算均值，fun=mean)

`fourfoldplot(x)` 用四个四分之一圆显示2X2列联表情况(x必须是dim=c(2, 2,k)的数组，或者是dim=c(2, 2)的矩阵，如果k = 1)

`assocplot(x)` Cohen-Friendly图，显示在二维列联表中行、列变量偏离独立性的程度

`mosaicplot(x)` 列联表的对数线性回归残差的马赛克图

`plot.ts(x)` 如果x是类"ts"的对象，作x的时间序列曲线，x可以是多元的,但是序列必须有相同的频率和时间

`ts.plot(x)` 同上，但如果x是多元的，序列可有不同的时间但须有相同的频率

↓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

1.1.2 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,`	从`(x0,y0)`各点到`(x1,y1)`各点画线段

`x1, y1)`	
`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

[用?demo (plotmath) 查看更多数学符号和公式命令]

数expression把自变量转换为数学公式.例如，

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

↑Code ↓Code

在图中相应坐标点(x; y)处显示下面的方程:

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

1.1.3 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)
par(opar)
```

↓Code

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 2
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.

2.1 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).

2.2 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

2.3 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 3

Programming with R

3.1 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

3.2 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

3.3 Miscellaneous programming tips

1. 使用fix()

使用fix(函数)可以更新(正)已经存在的函数. 比如R中计算阶乘的函数factorial 是使用gamma函数计算. 我们可以使用fix(factorial)修改你内容, 然后更新facorial.

2. 使用注释符号#

对任何你的程序做一个简单的描述是非常有用的, 它可以让你在以后重新理解和使用该函数.

3. 复杂程序分块化

对个发杂冗长的程序, 将其细分为若干结构相对对立的子程序, 对于理解整个程序很有帮助.

3.4 Debugging

在R中,我们可以使用函数traceback 来获得程序出错时的额外信息. 例如

↑Example

```
> cv <- function(x) {
```

```
+ sd(x / mean(x))
+ }
> cv(0)
Error in var(x, na.rm = na.rm) : missing observations in cov/cor
#错误信息显示函数var出错,但是我们没有使用此函数呀
> traceback()
3: var(x, na.rm = na.rm)
2: sd(x/mean(x))
1: cv(0)
```

↓Example

告诉我们命令sd调用了函数var, 而在计算var是出错. 标准的sd定义要求输入为正数, 因此解决方法是检查输入值

↑Example

```
> cv <- function(x) {
+ stopifnot(all(x > 0))
+ sd(x / mean(x))
+ }
```

```
> cv(0)
Error: all(x > 0) is not TRUE
```

↓Example

traceback函数显示错误出现在哪一步. 但是没有告诉我们为什么出错.

R中可以使用browser和debug函数对程序进行调试. 它们运行我们在函数计算过程中检查局部的变量, 或者执行任何R命令, 可以使用的调试命令为

↑Code

1. n-"next", 执行程序的下一行命令
2. c-"continue", 让程序执行下去
3. Q-推出调试

↓Code

例如, 我们可以调试前面的ft函数:

↑Example

```
> debug(ft)
> ft(3)
Browse[1]> n
```

```
debugging in: ft(m - 1)
debug: {
    if (m == 1)
        rlt <- 1
    else rlt <- m * ft(m - 1)
    return(rlt)
}
Browse[1]> n
debug: if (m == 1) rlt <- 1 else rlt <- m * ft(m - 1)
Browse[1]> n
debugging in: ft(m - 1)
debug: {
    if (m == 1)
        rlt <- 1
    else rlt <- m * ft(m - 1)
    return(rlt)
}
Browse[1]> n
```

```
debug: if (m == 1) rlt <- 1 else rlt <- m * ft(m - 1)
Browse[1]> n
debug: return(rlt)
Browse[1]> n
exiting from: ft(m - 1)
debug: return(rlt)
Browse[1]> n
exiting from: ft(m - 1)
debug: return(rlt)
Browse[1]> n
exiting from: ft(3)
[1] 6
```

↓Example

使用undebug(ft)来关闭函数ft的调试环境.

3.5 Efficient programming

当你的程序运行的速度和你期望的不一样快时, 就存在加速程序(称为优化)的

必要. 本节我们介绍几种手动优化(重写代码)方式.

1. 了解你的工具

为了使得代码更有效率, 了解你的工具是很有必要的. 比如R是以向量为操作对象的, 因此对整个向量直接操作要比单独操作向量各元素有效率得多.

↑Example

```
> X <- rnorm(100000)
> Y <- rnorm(100000)
> Z <- c()
> for (i in 1:100000) {
+ Z <- c(Z, X[i] + Y[i]) # this takes about 54.09 seconds
+ }
> Z <- rep(NA, 100000)
> for (i in 1:100000) {
+ Z[i] <- X[i] + Y[i] # this takes about 0.54 seconds
+ }
> Z <- X + Y # 0.002 seconds (approx)
```

↓Example

2. 使用有效率的程序设计

在算法设计上, 尽量使用高效率的程序设计方法.

3. 检测你的程序运行时间

在R中, 可以使用system.time()函数来测量程序的执行时间. 例如

↑Example

```
X <- rnorm(100000)
Y <- rnorm(100000)
Z <- rep(NA, 100000)
system.time({
for (i in 1:100000) {
Z[i] <- X[i] + Y[i]
}
})
 user  system elapsed
 0.54    0.00    0.55
```

↓Example

user时间指此任务执行花费的时间, system指系统完成其他任务所花费的时间, elapsed时间指我们可以从时钟上看到的执行此任务花费的时间.

我们也可以测量程序的各个部分执行所需要的时间, 以决定对程序的那一部分需要进行优化. 计算中一个著名的基本准则就是90/10准则: 90%的执行时间花在10%的程序代码上. 在R 中,我们可以使用函数Rprof()测试程序的性能. 此10% 的代码即时程序执行效率的瓶颈, 我们可以使用比如C或者Fortran改写此段代码, 以提升效率.

4. 在R 中调用C 或者Fortran 程序

例如, 我们有如下简单的C函数foo.c

↑Example

```
void foo(int *nin, double *x)
{
int n = nin[0];
int i;
for (i=0; i<n; i++)
x[i] = x[i] * x[i];
}
```

↓Example

为在R中调用此函数, Unix系统下我们先在命令提示符(在R外面)下输入

↑Example

```
R CMD SHLIB foo.c
```

↓Example

会在目录下生成编译后的程序foo.so, 然后在R中载入并使用该函数

↑Example

```
dyn.load("foo.so") # 载入
.C("foo", n=as.integer(5), x=as.double(rnorm(5))) #使用
dyn.unload("foo.so") # 取消载入
```

↓Example

在windows下, 必须先使用C编译器(需要另外安装)生成foo.c的编译dll文件: foo.dll, 然后在R中调用此dll文件.

Fortran程序的调用类似.

3.6 R script editors

我们可以使用各种文档编辑软件来写R的程序代码. 但是, 有一些针对R语法特别设计的软件可以更好的进行R程序开发. 常用的编辑器有

1. R commander [访问其网站]

R commander 是一款为R软件开发的图形用户界面(类似于Splus的图形界面). 安装R commander 可以通过安装“**Rcmdr**” 包完成, 安装此包后, 加载时(library(Rcmdr))会自动安装其所依赖的其他包.

2. Tinn-R 是一款免费的用于R程序开发的编辑器. 拥有丰富的功能, 比如词法检查, 语法高亮显示, 自动补全命令名等等. [访问其网站]

3. RWinEdt 是一款使用文本编辑器WinEdt来编写和执行R程序的插件. 可以通过安装包“**RWinEdt**” 来完安装此插件. 安装后, 在桌面生成RWinEdt的图标.