R语言基础

理学院–周世祥

2015年10月24日

目录

1 初识R语言

1.1 什么是R语言

你现在最常用的统计软件是什么？SAS？MATLAB？还是Eviews？[1]

如果你是一个经常和数据分析打交道，需要运用或编写各种统计方法的人。如果还没用上R，那么你已经脱离现在的主流了。 KDnuggets网站：http://www.kdnuggets.com. 每年都会发布一个数据分析与挖掘软件排行，比起matlab来R语言遥遥领先。

R是由新西兰奥克兰大学的Ross Ihaka与Robert Gentleman一起开发的一个面向对象的编程语言，因两人的名都是以R开头，所以命名为“R”。

R是一套完整的数据处理、计算和制图软件系统，也是一套开源的数据分析解决方案。其功能包括：

- 数据存储和处理系统
- 数组运算工具（其向量、矩阵运算方面功能尤其强大）
- 完整连贯的统计分析工具
- 优秀的统计制图功能
- 简便而强大的编程语言

1. 一些数据分析和数据挖掘软件使用的调查显示：R语言的使用在众多统计软件中名列榜首。
2. 免费是R流行开来的最大的一个因素。不过，R的最大优点是出色的可视化图形、丰富的统计方法以及高效的更新速度。

R功能强大、内容丰富，各种帮助文档很多。目前R已有近5000个扩展包。但是R的学习曲线比较陡峭，因此一本从入门到精通的参考书，会让您事半功倍!

1.2 R开发环境的安装

R的官方网站是http://www.r-project.org.

[1]这是第一稿，写的还是比较乱，只作为同学们课后学习参考，欢迎帮忙修改!

R安装软件和安装包（package）的下载链接在首页左侧的CRAN。作为中国用户，一般都会选择一个国内的镜像。目前中国大陆的镜像有如下4个.

整个R系统主要由一系列程序包（Package）组成。R包是函数、数据、预编译代码以一种定义完善的格式组成的集合。这些包默认储存在library里。R安装好后，除了默认自带的包之外，CRAN上还有数千附加包，由R核心开发团队之外的用户自行提交，如此多的程序包基本上保证R可以实现目前几乎所有的统计分析。

利用Rstudio安装。

安装完R扩展包，并不能马上使用，需要载入到当前会话中才能使用。R包的载入有两种方法：

1. 在R console输入命令library()载入。如需载入class包，则输入library(class)即可。

2. 利用RStudio载入。

一个包仅需安装一次，但是R的扩展包会经常被更新，要想使用更新后的扩展包，需要对已安装的包进行更新。

1. 在R 的 Console输入命令update.packages()，会出现需要更新的包和相关信息，并可以选择是否需要更新。如选择”y”，则会在线更新。

2. 利用Rstudio更新R包。载入R包后，可以使用该扩展包的函数和数据。比如“class”包，查看该包的函数。一是在R Console里输入命令

   help(package=”class”)，或者在RStudio右侧点击”class”。

载入R包后，可以使用该扩展包的函数和数据。比如我们载入“class”包后，想要查看该包里的函数。一种方法可以在R Console里输入命令

help(package=”class”)，则会返回一个包含了“class”包里所有函数和数据集名称的帮助文档。另一种方法，在RStudio右侧点击”class”,则会返回帮助文档，如需进一步了解该函数的使用方法，继续点击相应的函数，则会返回该函数的详细帮助文档。

1.3 R编辑器

下载安装后，双点R的快捷图标，打开R的操作平台（R Console）。虽然可以直接在此操作平台上输入命令，但是R本身默认的IDE很粗糙。一

般情况下最好不要直接在操作平台上输入命令，而是使用R的编辑软件。R拥有多种优秀的IDE。本书主要使用其中一款优秀的跨平台开源IDE——RStudio。RStudio将常用的窗口都整合在一起，更方便控制。

若你需要编写R语言代码，运行途径有两种：

1. 在左下角的命令窗口输入并回车运行；

2. 使用左上角的脚本编辑窗口编写好代码之后，根据你的需要选择运行代码。若需要使用脚本编辑代码，你需要在RStudio中执行File—>New File—>R Script命令（快捷键为Ctrl+Shift+N）来新建一个脚本编辑窗口。写好代码之后，可以通过点击Run按钮来运行程。（注意：如果直接点击Run按钮，则是运行当前行代码。如果先用鼠标选好要运行的代码，然后再点Run，就运行所选代码。）

在使用脚本编辑窗口的时候，RStudio还提供一些高级功能。在脚本标签下方，右侧有运行当前行或所选行，重新运行上一次代码等。点击左边的发光棒，会显示如图的六个功能，包括：代码补全，注释等，它的左边是查找和替换功能。Source on Save最好把它勾上，可以让你的代码自动保存。

左下角是一个类似Rgui的编辑器。这里可以写代码，也能显示程序运行过程和结果。（注意一般不建议在这里写，一是没法保存，二是不小心一部分写错了，很多都要重来。）

右上角的Data、Value和Function是上一次程序运行后，保存在.RData文件里面的值，通过鼠标点击使它们显示。 Environment下的标签可以让你切换工作空间.

Save可以保存当前工作区。

Import Dataset导入数据作为数据集。

Clear将当前工作区的所有变量和函数清除干净。

History标签运行的历史记录界面，可以保存下来，也可以选择一部分，然后按To Console或者To Source，前者是将选择的代码送到左下角的操作平台运行，后者是将代码送到左上角的光标位置。

右下角的功能较多，Files显示工作区内的文件，New Folder就是新建文件夹，Delete删除，Rename重命名， More则提供了其他功能。

Plot标签下提供图形显示，并可以随着这个工作区的变大而缩放。

Zoom放大图片,Export将图片导出，里面的Image Format处可以选择导出图片的格式，一般选择png格式。

Directory为选择保存的文件夹，File Name为图片的名字，Width和Height可以输入图片的宽高。

Package标签可以显示已经读入内存的包，也显示你安装了的所有的包；可以安装新的包，也可以升级各个包。

当然，除了RStudio之外，还有很多其他优秀的IDE可供选择，如RStudio Server、RKWard、ESS、Vim-R、StatET、Rattle、R AnalyticFlow、Revolution R Enterprise、Red-R等。这些IDE各有优劣。

其他还有类似Notepad++，SciTE等文本编辑器+R插件构成的开发环境，安装文件小巧，操作简单方便，基本语法高亮和代码补全的功能也应有尽有。

与其他计算机程序语言一样，R也有一个记录当前工作环境的工作空间(Workspace)，里面保存了所有用户定义的向量、矩阵、函数、数据框、列表等一系列对象。在一个R的会话结束时，可以选择(自动)保存当前的工作空间并在下次启动R时自动载入。

当前工作路径(Working Directory)是R用来读取和保存文件的默认路径。使用函数getwd()来查看当前路径。还可通过setwd()函数来指定当前的工作路径。如果需要读写这个目录之外的文件则需要使用完整的路径，并且要使用引号闭合路径名。

getwd()	显示当前工作目录
setwd(“mydir”)	修改当前工作目录
ls()	列出当前工作空间的所有对象
rm()	删除一个或多个对象
help(options)	显示可用选项的说明
save.image(“myfile”)	保存工作空间到文件myfile.RData中
load(“myfile”)	读取一个工作空间myfile.RData
q()	退出。并询问是否保存当前工作空间

表 1: 工作路径设置的函数

在RStudio软件中，可以直接在右上角部分直接看到工作空间的各种对象以及历史记录。

2 R的数据结构

R语言的数据类型主要有:包括数值型、逻辑型、字符型、复数型，原型。此外，也可以是缺省值。

- 这种数据的形式是实数。可以写成整数（integers），小数（decimal fractions），或科学记数（scientific notation）的方式。数值型实际上是两种独立模式的混合说法，即整数型（integers）和双精度型（double precision）。该种类型数据默认是双精度型数据（double precision）。— 数值型（numeric）
- 这种数据的形式是夹在双引号“”或单引号‘’之间的字符串，如“sdut”。 —字符型（character）
- 这种数据只能取T（TRUE）或F（FALSE）值。 —逻辑型（logical）
- 这种数据是形如a+bi形式的复数。 — 复数型（complex）
- 这种类型以二进制形式保存数据。 — 原味(原始)型（raw）
- 有些统计资料是不完整的。当一个元素或值在统计的时候是“不可得到”（not available）或“缺失值”（missing value）的时候，相关位置可能会被保留并且赋予一个特定的NA（not available）值。任何NA的运算结果都是NA。is.na()用来检测数据是否缺失，如果数据缺失，则返回TRUE，否则，返回FALSE。 — 缺省值（missing value）

例如：

```
  z<-c(1:5,NA)  #生成向量z
  z  #返回z向量的结果

## [1]  1  2  3  4  5 NA

  is.na(z) #识别z向量的值是否有缺失值

## [1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

上面，我们用c()函数首先生成了向量z。is.na()返回的结果是前面5个元素都FALSE，最后1个是TRUE，这说明前面5个都不是缺失值，最后1个

是缺失值。常见的辨别和转换对象类型的函数如下表： R语言里的数据对象主要有六种结构：

- 向量（vector）
- 数组（array）
- 因子（factor）
- 列表（list）
- 数据框（data frames）

2.1 向量赋值

向量（vector）是由有相同基本类型元素组成的序列，相当于一维数组。

例：

```
x <-c(1,3,5,7,9)  #用c( )构建向量
```

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，而它输出的值是一个把这些参数首尾相连形成的一个向量。

“#”符号后面跟的是注释，在写程序的时候清楚表明程序工作的注释能大大提高程序的可读性。

数据类型	辨别函数	转换函数
numeric	is.numeric()	as.numeric()
character	is.character()	as.character()
complex	is.complex()	as.complex()
double	is.double()	as.double()
integer	is.integer()	as. integer()
logical	is. logical()	as. logical()
NA	is.na()	as.na()

表 2: 辨别和转换数据对象类型的函数

“< −”是赋值符号，表示把< −后面的内容赋值给< −前面的内容，R的赋值符号除了“< −”外,还有“− >”、“=”， − >的用法和< −的用法正好相反。

注意：R也允许“=”赋值，但不是标准语法，有些情况下用“=”有可能会出现问题，因此一般情况不建议使用“=”。

例如：

```
c(1,3,5,7,9) -> y  #将c()生成的数值向量赋值给y
y

## [1] 1 3 5 7 9

z <- c("Male","Female","Female","Male","Male")  #将c()生成的字符向量赋值给z
 z

## [1] "Male"   "Female" "Female" "Male"   "Male"

u=c(TRUE,FALSE,TRUE,FALSE,FALSE) #将c()生成的逻辑向量赋值给u
u

## [1]  TRUE FALSE  TRUE FALSE FALSE
```

此处，y是数值向量，z是字符型向量，u是逻辑型向量。

注意：单个向量中的数据要求是相同类型，同一向量中无法混杂不同类型的数据。

对于字符向量，一个很重要的函数paste()可以把自变量对应元素连成一个字符串，长度不相同时，较短的向量被重复使用。例如：

```
  v<-paste("x",1:5,sep="")
  v

## [1] "x1" "x2" "x3" "x4" "x5"
```

此外，也可以用assign（）函数对向量进行赋值。例如：

```
  assign("w",c(1,3,5,7,9))
 w

## [1] 1 3 5 7 9
```

2.2 向量运算

对于向量的乘法，除法，乘方运算，其方法是对应向量的每个分量做乘法、除法和乘方运算。例如：

```
x <-c(1,3,5,7,9)
  c(1,3,5,7,9) -> y
  x * y  #对应元素相乘

## [1]  1  9 25 49 81

  x / y

## [1] 1 1 1 1 1

  x^2

## [1]  1  9 25 49 81

  y^x

## [1]         1        27      3125    823543 387420489
```

此外，”%/%”表示整数除法(5%/%3为1)，”%%”表示求余数(5%%3为2)。”

向量运算会对该向量的每一个元素都进行同样的运算。出现在同一个表达式的向量最好同一长度。如果长度不一，表达式中短的向量将会被循环使用，表达式的值将是一个和最长的向量等长的向量。例如：

```
  c(1,3,5)+c(2,4,6,8,10)

## Warning in c(1, 3, 5) + c(2, 4, 6, 8, 10):  长的对象长度不是短的对象长度的整倍数
```

```
## [1]  3  7 11  9 13
```

第一个向量的长度小于第二个向量，循环补齐第一向量的长度，即为c(1,3,5,1,3)

2.3 生成有规则序列

R可以产生正则序列，最简单的是用“:”符号，就可以产生有规律的正则序列。例如：

```
 (t <- 1:10)

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

  (r <- 5:1)

## [1] 5 4 3 2 1

  2*1:5

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

其中，5:1表示逆向序列, 并且在表达式运算中，“:”的运算级别最高，即上面的2*1:5，R是先生成1-5的向量，然后再乘上2。这里在表达式外面套()的意思把结果直接打印出来，比如 t < − 1:10而不套括号，则R将运算结果保存在t对象里，但是不会把t的结果打印出来。

可以用函数seq()产生有规律的各种序列，其句法是：seq(from, to, by)，from表示序列的起始值，to表示序列的终止值，by表示步长。其中，by参数（参数by）省略时，默认步长为1。并且函数seq（）也可以产生降序数列。例如：

```
seq(1,10,2) #生成从1开始，10结束，步长为2的序列

## [1] 1 3 5 7 9

 seq(1,10) #默认步长为1

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

```
 seq(10,1,-1) #步长为-1

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

 seq(1,by=2,length=10)# 有时候我们需要关注的是数列的长度

##  [1]  1  3  5  7  9 11 13 15 17 19
```

rep()函数可以用各种复杂的方式重复一个对象。其命令是：rep(x, times, ...)，其中x表示要重复的对象，times表示重复的次数。

```
 rep(c(1,3),4)  #将c(1,3)向量重复4次

## [1] 1 3 1 3 1 3 1 3

rep(c(1,3),each=4)

## [1] 1 1 1 1 3 3 3 3

rep(1:3,rep(2,3))

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

第一个是向量c(1,3)重复4次的情况，也可以用第二个对每个元素进行重复，第三个是函数rep( )的嵌套使用，里层的rep(2,3)实际就等价于向量c(2,2,2)。

向量里元素的个数称为向量的长度（length）。长度为1的向量就是常数（或标量）。函数length()可以返回向量的长度，mode()可以返回向量的数据类型，min()返回向量的最小值，max()返回向量的最大值，range()返回向量的范围，which.min()、which.max()返回在第几个分量求到最小、最大值。例如：

```
x <-c(1,3,5,7,9)
 length(x)

## [1] 5

min(x)
```

```
## [1] 1

range(x)

## [1] 1 9
```

R提供了很多的函数可以对向量进行运算，这里不一一列举，下面列出几个常用函数表。

函数	用途	函数	用途
sum（）	求和	rev（）	反排序
max（）	求最大值	rank（）	求秩
min（）	求最小值	append（）	添加
range（）	求极差（全矩）	replace（）	替换
mean（）	求均值	match（）	匹配
median （）	求中位数	pmatch（）	部分匹配
var（）	求方差	all（）	判断所有
sd（）	求标准差	any（）	判断部分
sort（）	排序	prod（）	积

表 3: 对向量运算常见函数表

2.4 向量的索引

在R中提供了灵活的向量下标运算。取出向量的某一个元素可以用x[i]。也可以通过赋值语句来改变一个或多个元素的值。例如：

```
x <- c(1,3,5)
 x[2]   #返回x向量的第2元素

## [1] 3

 (c(1,2,3)+4)[2]

## [1] 6

#先进行向量运算c(1,2,3)+4，再返回该向量的第2个元素
```

```
 x[2] <- 10 #将10赋值给x向量的第2个元素，,即替换掉原来的值
 x
```

```
## [1]  1 10  5
```

```
  x[c(1,3)] <- c(9,11) #将9和11赋值给x向量的第1和第3个元素
  x
```

```
## [1]  9 10 11
```

```
 x <- c(1,3,5)
          x < 4   #返回逻辑结果，即x向量的元素是否小于4
```

```
## [1]  TRUE  TRUE FALSE
```

```
          x[x<4] #返回x向量里小于4的元素
```

```
## [1] 1 3
```

```
           z <- c(-1,1:3,NA)
           z
```

```
## [1] -1  1  2  3 NA
```

```
           z[is.na(z)] <- 0 #将0赋值给z向量里的NA值
           z
```

```
## [1] -1  1  2  3  0
```

```
            z <- c(-1,1:3,NA)
            y <- z[!is.na(z)]  #将z里的非缺失值赋值给y
 y
```

```
## [1] -1  1  2  3
```

```
           x<-c(-3,-2,-1,0,5,7)
         y <- numeric(length(x))
         #生成于x向量长度相同的数值型向量
   y
```

```
## [1] 0 0 0 0 0 0
```

```
 y[x<0] <- 1-x[x<0]
#求出x中小于0元素对应位置，y对应位置的值用1-x[x<0]替代
 y
```

```
## [1] 4 3 2 0 0 0
```

```
 y[x>=0] <- 1+x[x>=0]
#求出x中大等于0元素对应位置，y对应位置的元素用1-x[x<0]赋值
 y
```

```
## [1] 4 3 2 1 6 8
```

如果x[i]，i取值是小于0的负整数，则表示删除相应位置的元素。

3 矩阵

矩阵（matrix）是将数据用行和列排列的长方形表格，它是二维的数组，其单元必须是相同的数据类型。通常用列来表示不同的变量，用行表示各个对象。R语言生成矩阵的函数是matrix ( )，其句法是：

matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)

其中，data 项为必要的矩阵元素,nrow 为行数,ncol 为列数,注意 nrow 与 ncol 的乘积应为矩阵元素个数, dimnames 给定行和列的名称，byrow 项控制排列元素时是否按行进行,默认byrow=FALSE，即按列顺序排列。

例如：

```
     matrix(1:12,nrow=4,ncol=3) #默认按列填充
```

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

```
  matrix(1:12,nrow=4,ncol=3,byrow=T) #按行填充
```

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

矩阵的运算种类多，方法多，此处作简要介绍。

3.1 矩阵的运算

矩阵的转置可以用函数t()来计算，类似的，若将函数 t()作用于一个向量x,则当做x为列向量,返回结果为一个行向量。若想得到一个列向量,可用t(t(x))。

```
(A <- matrix(1:12,nrow=3,ncol=4))
```

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

```
t(A)
```

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

矩阵的加减法

```
A <- B <- matrix(1:12,nrow=3)
#将生成的矩阵赋给B，同时又赋给A
 A+B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    8   14   20
## [2,]    4   10   16   22
## [3,]    6   12   18   24
```

```
 #矩阵的数乘
 3*A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3   12   21   30
## [2,]    6   15   24   33
## [3,]    9   18   27   36
```

矩阵的乘法，除了矩阵需要满足可以相乘的要求之外，在R中需要用运算符“

```
  B <- t(A)
  A%*%B
```

```
##      [,1] [,2] [,3]
## [1,]  166  188  210
## [2,]  188  214  240
## [3,]  210  240  270
```

R中还可以对矩阵的对角元素进行计算，例如要取一个方阵的对角元素:

```
  (A <- matrix(1:16,nrow=4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

```
  diag(A)
```

```
## [1]  1  6 11 16
```

另外，对一个向量应用diag()函数可以产生以这样向量的元素为对角元的对角矩阵

```
  diag(diag(A))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    6    0    0
## [3,]    0    0   11    0
## [4,]    0    0    0   16
```

如果输入只有一个正整数的话，diag()函数将会生成一个对应维数的单位阵。

```
  diag(3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

求逆，在R中使用solve()函数可以计算，如果是solve(a,b)的话是解线性方程组ax=b，b默认为单位矩阵。

```
 (A <- matrix(rnorm(16),4,4)) #rnorm()是生成16个标准正态分布随机数
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,] -0.4329678  0.3858554 -1.5941060  0.5065469
## [2,] -0.3340897  0.6228424 -0.3819946  0.3148296
## [3,]  0.2520218  0.3800884 -0.6719775 -0.8369154
## [4,] -0.8564510 -1.4129397 -1.4123534 -0.4172030
```

```
solve(A)
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,]  1.2039277 -2.5288843  0.1491816 -0.7458579
```

```
## [2,] -0.2389271  1.0866850  0.3621540 -0.1965457
## [3,] -0.7420817  0.6889521 -0.2119312  0.0440357
## [4,]  0.8498647 -0.8211798 -0.8153028 -0.3492208
```

3.2 矩阵的特征值和特征向量

求特征值和特征向量的运算，在R中可以通过函数eigen()来得到。

```
 (A <- diag(4)+1)

##      [,1] [,2] [,3] [,4]
## [1,]    2    1    1    1
## [2,]    1    2    1    1
## [3,]    1    1    2    1
## [4,]    1    1    1    2

(A.eigen <- eigen(A,symmetric=T))

## $values
## [1] 5 1 1 1
##
## $vectors
##      [,1]       [,2]       [,3]       [,4]
## [1,] -0.5  0.8660254  0.0000000  0.0000000
## [2,] -0.5 -0.2886751 -0.5773503 -0.5773503
## [3,] -0.5 -0.2886751 -0.2113249  0.7886751
## [4,] -0.5 -0.2886751  0.7886751 -0.2113249

   #此处返回的结果是列表格式，
```

对于正定矩阵A，可以对其进行Choleskey分解，即A=PTP，其中P为上三角矩阵，在R中可以用函数chol()进行Choleskey分解。

```
 chol(A)

##          [,1]      [,2]      [,3]      [,4]
```

```
## [1,] 1.414214 0.7071068 0.7071068 0.7071068
## [2,] 0.000000 1.2247449 0.4082483 0.4082483
## [3,] 0.000000 0.0000000 1.1547005 0.2886751
## [4,] 0.000000 0.0000000 0.0000000 1.1180340
```

若矩阵为对称正定阵，则可以利用Choleskey分解来求行列式的值以及矩阵的逆，并且这种用法更有效。

```
 prod(diag(chol(A))^2)#求出A矩阵的对角元素，然后求平方，再求连乘积

## [1] 5

 det(A) #求A矩阵行列式

## [1] 5
```

3.3 奇异值分解

若A为m行n列矩阵，秩为r，则矩阵可以进行奇异值分解得到A=UDVT。在R中可以通过svd()函数进行计算。

```
  (A <- matrix(1:18,3,6))

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    4    7   10   13   16
## [2,]    2    5    8   11   14   17
## [3,]    3    6    9   12   15   18

  (A.svd <- svd(A))

## $d
## [1] 4.589453e+01 1.640705e+00 1.366522e-15
##
## $u
##            [,1]        [,2]       [,3]
```

```
## [1,] -0.5290354  0.74394551  0.4082483
## [2,] -0.5760715  0.03840487 -0.8164966
## [3,] -0.6231077 -0.66713577  0.4082483
##
## $v
##             [,1]        [,2]       [,3]
## [1,] -0.07736219 -0.71960032 -0.4076688
## [2,] -0.19033085 -0.50893247  0.5745647
## [3,] -0.30329950 -0.29826463 -0.0280114
## [4,] -0.41626816 -0.08759679  0.2226621
## [5,] -0.52923682  0.12307105 -0.6212052
## [6,] -0.64220548  0.33373889  0.2596585

#求矩阵A的svd分解，并将结果赋值给A.svd
```

3.4 QR分解

实数矩阵A的QR分解是把A分为A=QR，这里的Q是正交矩阵，而R是上三角矩阵。QR分解在R中可以用函数qr()计算.其中，rank项返回矩阵的秩，qr项包含了矩阵Q和R的信息，要得到对应矩阵，可以用函数qr.R()，qr.Q()计算。 QR分解。例如：

```
  B <- matrix(1:16,4,4)
  qr(B)

## $qr
##            [,1]        [,2]          [,3]          [,4]
## [1,] -5.4772256 -12.7801930 -2.008316e+01 -2.738613e+01
## [2,]  0.3651484  -3.2659863 -6.531973e+00 -9.797959e+00
## [3,]  0.5477226  -0.3781696  1.601186e-15  2.217027e-15
## [4,]  0.7302967  -0.9124744 -5.547002e-01 -1.478018e-15
##
## $rank
## [1] 2
```

```
##
## $qraux
## [1] 1.182574e+00 1.156135e+00 1.832050e+00 1.478018e-15
##
## $pivot
## [1] 1 2 3 4
##
## attr(,"class")
## [1] "qr"
```

```
  #矩阵的秩
    #Q中的额外信息
   #分解过程中的旋转信息
   #返回属性
  qr.R(qr(B)) #提取R矩阵
```

```
##           [,1]       [,2]          [,3]          [,4]
## [1,] -5.477226 -12.780193 -2.008316e+01 -2.738613e+01
## [2,]  0.000000  -3.265986 -6.531973e+00 -9.797959e+00
## [3,]  0.000000   0.000000  1.601186e-15  2.217027e-15
## [4,]  0.000000   0.000000  0.000000e+00 -1.478018e-15
```

```
  qr.Q(qr(B)) #提取Q矩阵
```

```
##            [,1]          [,2]       [,3]        [,4]
## [1,] -0.1825742 -8.164966e-01 -0.4000874 -0.37407225
## [2,] -0.3651484 -4.082483e-01  0.2546329  0.79697056
## [3,] -0.5477226 -1.665335e-16  0.6909965 -0.47172438
## [4,] -0.7302967  4.082483e-01 -0.5455419  0.04882607
```

在R中可使用dim()得到矩阵的维数，nrow()求行数，ncol()求列数；rowSums()求各行和，rowMeans()求行均值，colSums()求各列和，colMeans()求列均值。

3.5 apply()运算

对于矩阵的运算，我们还可以使用apply()函数来进行各种计算，其用法为apply(X, MARGIN, FUN, ...) 。其中，X表示需要处理的数据，MARGIN表示函数的作用范围，1为对行运算，2为对列运算。FUN为需要运用的函数。例如：

```
A <- matrix(1:12,3,4)
  apply(A,2,sum) #矩阵的列求和
```

```
## [1]  6 15 24 33
```

```
  apply(A,2,mean) #矩阵的列求均值
```

```
## [1]  2  5  8 11
```

```
  apply(A,2,var) #矩阵的列求方差
```

```
## [1] 1 1 1 1
```

```
  apply(A,2,sd) #矩阵的列求标准差
```

```
## [1] 1 1 1 1
```

矩阵合并可以使用rbind()和cbind()函数来对矩阵按照行和列进行合并。例如：

```
  B=matrix(c(1,1,1,1),2,2) #生成2×2的矩阵
  rbind(B,B) #将B和B矩阵按行合并
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1
## [3,]    1    1
## [4,]    1    1
```

```
   cbind(B,B) #将B和B矩阵按列合并
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    1    1    1
## [2,]    1    1    1    1
```

对于得到的矩阵，可以使用A[i,j]得到A矩阵第i行第j列的元素，A[i,]和A[,j]分别表示返回第i行和第j列的所有元素。也可以使用A[i:k,j:l]的形式来获得多行多列的子矩阵。

例如：

```
  A=matrix(1:12,3,4)
  A[2,3] #返回矩阵第2行第3列元素

## [1] 8

  A[2,] #返回矩阵第2行所有元素

## [1]  2  5  8 11

  A[,3] #返回矩阵第3列所有元素

## [1] 7 8 9

  A[1:3,2] #返回矩阵第1到3行，且是第2列的元素

## [1] 4 5 6
```

使用as.matrix()把非矩阵格式的转换成矩阵格式，函数is.matrix()可以辨别是否矩阵。

4 数组

数组（array）可以看作是带有多个下标的类型相同的元素的集合。也可以看作是向量和矩阵的推广，一维数组就是向量，二维数组就是矩阵。数组的生成函数是array()，其句法是：

array(data = NA, dim = length(data), dimnames = NULL)

其中data表示数据，可以为空，dim表示维数，dimnames可以更改数组的维度的名称。

例如:

```
(xx <- array(1:24,c(3,4,2)))    # 产生维数为(3,4,2)的3维数组
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

4.1 索引数组

索引数组类似于索引矩阵，索引向量可以利用下标位置来定义；其中dim()函数可以返回数组的维数，dim()还可以用来将向量转化成数组或矩阵(见下例)。数组也可以用“+”、“-”、“*”、“/”以及函数等进行运算，其方法和矩阵相类似，这里不再详述。

```
xx[2,3,2]
```

```
## [1] 20
```

```
xx[2,1:3,2]
```

```
## [1] 14 17 20
```

```
xx[,2,]
```

```
##      [,1] [,2]
## [1,]    4   16
## [2,]    5   17
## [3,]    6   18
```

```
dim(xx)

## [1] 3 4 2

zz=c(2,5,6,8,1,4,6,9,10,7,3,5)
dim(zz)=c(2,2,3)
#将向量转成维度为(2,2,3)的数组
 zz

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

5 因子类型

分类型数据（category data）经常要把数据分成不同的水平或因子（factor）。比如，学生的性别包含男和女两个因子。因子代表变量的不同可能的水平（即使在数据中不出现）。在统计模型统计分析中十分有用，例如将0，1转换为’yes’,’no’就很方便，在R里可以使用factor函数来创建因子，函数形式如下：

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x))

5.1 因子水平

其中，levels用来指定因子的水平；labels用来指定水平的名字；exclude表示在x中需要排除的水平；ordered用来决定因子的水平是否有次序。

例如一组学生数据：

```
 y <- c("女","男","男","女","女","女","男")
 (f <- factor(y)) #生成因子

## [1] 女 男 男 女 女 女 男
## Levels: 男 女

  levels(f) #提取因子的水平

## [1] "男" "女"
```

倘若要表示因子之间有大小顺序(考虑因子之间的顺序)，则可以利用ordered()函数产生。

```
  score <- c("B","C","D","B","A","D","A")
  (score_o <- ordered(score,levels=c("D","C","B","A"))) #生成有序因子

## [1] B C D B A D A
## Levels: D < C < B < A
```

6 列表

向量、矩阵和数组的元素都单元必须是同一类型的数据。如果一个数据对象需要含有不同的数据类型，可以采用列表（list）这种数据对象的形式（数据对象形式）。列表是一个对象的有序集合构成的对象，列表中包含的对象又称为它的分量（components），分量可以是不同的模式或类型，如

一个列表可以包括数值向量、逻辑向量、矩阵、字符、数组等。创建列表的函数是list()，其句法是：list（变量1=分量1，变量2=分量2，...）。

例如：下面是某校部分学生的情况，其中，x、y、z分别表示班级、性别和成绩。

```
 x <- c(1,1,2,2,3,3,3)
  y <- c("女","男","男","女","女","女","男")
  z <- c(80,85,92,76,61,95,83)
  (LST <- list(class=x,sex=y,score=z))

## $class
## [1] 1 1 2 2 3 3 3
##
## $sex
## [1] "女" "男" "男" "女" "女" "女" "男"
##
## $score
## [1] 80 85 92 76 61 95 83
```

6.1 列表访问

若要访问列表的某一成分，可以用LST[[1]],LST[[2]]的形式访问，要访问第二个分量的前三个元素可以用LST[[2]][1:3]：

```
 LST[[3]] #返回列表的第三个成分的值

## [1] 80 85 92 76 61 95 83

  LST[[2]][1:3] #返回列表第二个成分的第1到3元素

## [1] "女" "男" "男"
```

由于分量可以被命名，这时，我们可以在列表名称后加$符号，再写上成分名称来访问列表分量。其中成分名可以简写到可以与其它成分能够区分的最短程度，如LST$sc与LST$score表示同样的分量。例如：

```
LST$score #返回socre值

## [1] 80 85 92 76 61 95 83

  LST$sc #返回socre值

## [1] 80 85 92 76 61 95 83
```

函数length()、mode()、names()可以分别返回列表的长度(分量的数目)、数据类型、列表里成分的名字。在这里要注意LST[[1]]和LST[1]的差别，[[…]]是选择单个元素的操作符，而[…]是一个一般通用的下标操作符。因此前者得到的是LST中的第一个对象，并且包含分量名字的命名列表中的分量名字会被排除在外的1；而后者得到的则是LST中仅仅由第一个元素构成的子列表。

7 数据框

7.1 数据框的生成

数据框（data frame）是一种矩阵形式的数据，但数据框中各列可以是不同类型的数据。数据框每列是一个变量，每行是一个观测。数据框可以看成是矩阵（matrix）的推广，也可以看作是一种特殊的列表对象（list）。数据框是R语言特有的数据类型，也是进行统计分析最为有用的数据类型。不过对于可能列入数据框中的列表有如下一些限制：

- 分量必须是向量(数值，字符，逻辑)，因子，数值矩阵，列表或者其他数据框。
- 矩阵，列表和数据框为新的数据框提供了尽可能多的变量，因为它们各自拥有列、元素或者变量。
- 数值向量、逻辑值、因子保持原有格式，而字符向量会被强制转换成因子并且它的水平就是向量中出现的独立值。
- 在数据框中以变量形式出现的向量结构必须长度一致，矩阵结构必须有一样的行数。

R语言中用函数data.frame ( )生成数据框，其句法是： data.frame(..., row.names = NULL, check.rows = FALSE, ...)

数据框的列名默认为变量名，也可以对列名、行名进行重新命名。例如：

```
(student <- data.frame(x,y,z))

##   x  y  z
## 1 1 女 80
## 2 1 男 85
## 3 2 男 92
## 4 2 女 76
## 5 3 女 61
## 6 3 女 95
## 7 3 男 83

(student <- data.frame(class=x,sex=y,score=z))

##   class sex score
## 1     1  女    80
## 2     1  男    85
## 3     2  男    92
## 4     2  女    76
## 5     3  女    61
## 6     3  女    95
## 7     3  男    83
```

当然，我们也可以对数据框的行名进行修改，例如：

```
row.names(student) <- c("王x","张x","赵x","刘x","黄x","孙x","李x")
  student

##     class sex score
## 王x     1  女    80
## 张x     1  男    85
## 赵x     2  男    92
```

```
## 刘x     2  女    76
## 黄x     3  女    61
## 孙x     3  女    95
## 李x     3  男    83
```

7.2 数据框的引用

以数组形式访问。数据框可以看作是特殊的数组。数组是储存数据的有效方法，可以按行或列访问，就像电子表格一样，但输入的数据必须是同一类型。数据框可以看作数组是因为数据框的列表示变量、行表示样本观察数，因此我们可以访问指定的行或列。例如：

```
 student[,"score"]      #返回y变量的所有样本观察数

## [1] 80 85 92 76 61 95 83

 student[,3]

## [1] 80 85 92 76 61 95 83

 student[1:5,1:3]   #返回第1至第5行，第1至第3列的观察数

##     class sex score
## 王x     1  女    80
## 张x     1  男    85
## 赵x     2  男    92
## 刘x     2  女    76
## 黄x     3  女    61

student[,]              #返回所有行所有列数据

##     class sex score
## 王x     1  女    80
## 张x     1  男    85
## 赵x     2  男    92
## 刘x     2  女    76
```

```
## 黄x     3  女    61
## 孙x     3  女    95
## 李x     3  男    83
```

以列表形式访问数据框。列表比数据框更为一般、更为广泛。列表是对象的集合，而且这些对象可以是不同类型的。数据框是特殊的列表，数据框的列看作向量，而且要求是同一类型对象。以列表形式访问数据框，只要在列表名称后面加$符号，再写上变量名即可。例如：

```
 student$score

## [1] 80 85 92 76 61 95 83
```

除了用$形式访问外，还可以用列表名[[变量名(号)]]形式访问：

```
student[["score"]]

## [1] 80 85 92 76 61 95 83

 student[[3]]

## [1] 80 85 92 76 61 95 83
```

还可以筛选出符合我们条件的数据，比如对上面的数据要得到成绩大于80分的学生，可以按如下的方法得到。例如：

```
student[student$score>80,]

##     class sex score
## 张x     1  男    85
## 赵x     2  男    92
## 孙x     3  女    95
## 李x     3  男    83
```

7.3 数据框绑定Attach()函数

数据框的主要用途是保存统计建模的数据。R软件的统计建模功能都需要以数据框为输入数据。可把数据框当成一种矩阵来处理。在使用数据

框的变量时可以用“数据框名$变量名”的记法。但这样使用较麻烦，R软件提供了attach()函数可以把数据框中的变量“链接”到内存中，将数据框“连接(绑定)”入当前的名字空间，从而可以直接用数据框中的变量名访问而不必用“数据框名$变量名”这种格式。要取消连接，用函数detach()即可。

例如上面student数据框有三个变量，对于数据框里的变量我们不能直接引用，可以用“数据框名$变量名”的格式，或是利用attach( )把数据框“连接（绑定）”入当前的名字空间。

```
  score
```

```
## [1] "B" "C" "D" "B" "A" "D" "A"
```

```
  student$score
```

```
## [1] 80 85 92 76 61 95 83
```

```
  attach(student)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##    score
```

```
  score
```

```
## [1] "B" "C" "D" "B" "A" "D" "A"
```

要取消连接，用函数detach()即可。

参考文献

[1] 方匡南,朱建平,姜叶飞.R数据分析——方法与案例详解（双色）.电子工业出版社,201502.

[2] 王斌会.多元统计分析及R语言建模.暨南大学出版社.201405.