R 语言基础

shixiangzhou

2015 年 11 月 1 日

R 软件的安装

R 综合档案网络 CRAN, 国内有镜像。

启动

双击：`Rgui.exe`，退出 q().

工作目录

```
getwd()
setwd()
```

帮助

```
help(x) #x 为函数名
help.search("x") #x 为函数名
help.start() # 得 HTML 帮助界面
```

包的安装

Rstudio 提供的图形用户界面 GUI 来安装，或键盘按快捷键 Ctr+7 找到安装包界面。另一种安装包的方法是

```
install.packges("coefplot")
```

最近出现直接在存储库 GitHub 或 BitBucket 上安装软件包的方法，这种方法能够得到包的开发版本。但需要通过 devtools 包来完成。

```
require(devtools)
install_github(repo="coefplot",username="jaredlander")
```

有时候需要在本地文件安装软件包，可能是 zip 格式或 tar.gz 格式。也可以通过命令

`install.packages("coefplot_1.1.7.zip")` 安装。

包的卸载

极少情况下需要卸载，最简单是在 Rstudio 中软件包界面的右边单击灰色圆圈内的白色 X 按钮即可，或使用 `remove.packges` 命令，第一个参数为需要卸载的包名字符向量。

包的加载

library 或者 require 都可以。只是 require 会在加载成功时返回 TRUE，加载失败时返回带 FALSE 的字符警告。在一个函数体内部加载包时这个返回值非常有用，可以作为选择或拒绝某一选择命令的依据。

`require(coefplot)` 会依次显示同样加载的相关包，可以隐藏。

`require(coefplot, quietly=TRUE)` 一个软件包仅在启动一个新的 R 会话时被加载。一旦被加载，它将会一直存在 R 工作空间中直至 R 被重启或其被分离。有时候已经加载的包可能需要被卸载。

`detach("package:coefplot")`

不同包内名称相同并不少见。例如在 arm 包和 coefplot 包内都有函数 coefplot 函数，如果这两个包被加载，当使用此函数时，则被认为调用的是后被加载包内函数。解决办法是在调用函数时在两个冒号前指明其所在的包名。

`arm::coefplot(object)`

`coefplot::coefplot(object)`

CRAN 任务视图：cran.r-project.org/web/views 实现不同功能的软件包列表。

基本计算环境

可以作为计算器。

+, -, *, /, ^, %%(取模), %/%(整除运算符)

```
1+1
```

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

```
1+2+3
```

```
## [1] 6
```

```
3*7*2
```

```
## [1] 42
```

```
4/2
```

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

```
4/3
```

```
## [1] 1.333333
```

```
4 * 6+5
```

```
## [1] 29
```

```
(4 * 6)+5
```

```
## [1] 29
```

```
4 * (6+5)
```

```
## [1] 44
```

建议乘除号每个运算符之间加一个白色空格，尽管不是必须的。

```
(1+1/100)^100
```

```
## [1] 2.704814
```

```
17 %% 5
```

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

```
17 %/% 5
```

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

默认7位有效数字，可用 options(digits=x) 修改精度。

```
exp(1)
```

```
## [1] 2.718282
```

```
options(digits=16)
exp(1)
```

```
## [1] 2.718281828459045
```

```
pi
```

```
## [1] 3.141592653589793
```

```
sin(pi/6)
```

```
## [1] 0.4999999999999999
```

变量

变量是任何编程语言不可缺少的一部分。R 具有很大的灵活性，不需要事先定义变量的类型。R 中变量可以存储任何数据类型，也可以存放任何 R 的对象，例如函数，分析的结果，以及一个图形。

变量赋值

```
x <- 2
x
```

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

```
y =5 # 倾向于前者
y
```

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

```
3 ->z
z
```

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

```
a<-b<-7
a
```

```
## [1] 7
```

```
b
```

```
## [1] 7
```

```
assign("j",4)
j
```

```
## [1] 4
```

```
# 研究极限
x<- 100
(1+1/x)^x
```

```
## [1] 2.704813829421529
```

```
x<- 200
(1+1/x)^x
```

```
## [1] 2.711517122929293
```

```
(y<-(1+1/x)^x)
```

```
## [1] 2.711517122929293
```

```
n<- 1
n<- n+1 # 右边表达式首先被计算
n
```

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

删除变量

```
j
```

```
## [1] 4
```

```
rm(j)

# 找不到对象: j, Error: object 'j' not found
```

虽然对于操作系统来说，释放存储空间并不必要，但是释放存储空间可以使得 R 存储更多的对象。

R 中的变量名是区分大小写的。

```
theVariable <- 17

theVariable
```

```
## [1] 17
```

```
# 找不到对象: TheVariable
```

内置的简单函数

```
seq(from=1,to=9,by=2)
```

函数是具有一个或多个自变量 (或输入)，而且可以生成一个或多个输出的功能体.

seq() 是内置函数，参数间用逗号，参数赋值用等号，返回一个等差数列。

```
floor(x),celing(x) # 向下和向上取整
```

```
 seq(from=1,to=9) # 默认步长为 1, 与 matlab 类似
```

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

```
 seq(1,9,2) # 参数按照默认顺序，
```

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

```
 seq(to=9,from=1) # 写了参数名的好处，不需要按顺序排列
```

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

```
 seq(by=-2,9,1) # 有写名字的，有不写名字的，比较灵活，看 by 的值可以为负的
```

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

```
x<- 9
 seq(1,x,x/3) # 参数可以为变量的
```

```
## [1] 1 4 7
```

R 中函数参数个数是变化的，甚至没有参数也是可以的，但必须有圆括弧，即使无参数！

```
 mean(x) # 内置函数
```

```
## [1] 9
```

```
  ? mean # 查看函数帮助信息
```

```
## starting httpd help server ... done
```

```
   apropos("mea")# 模糊查找
```

```
##  [1] ".colMeans"          ".rowMeans"          "colMeans"
##  [4] "influence.measures" "kmeans"             "mean"
##  [7] "mean.Date"          "mean.default"       "mean.difftime"
## [10] "mean.POSIXct"       "mean.POSIXlt"       "rowMeans"
## [13] "weighted.mean"
```

试试：demo,demo(),demo(graphics)

工作空间

列出当前工作空间所有已定义的对象：ls(),objects()

移除对象：rm(x), rm(list=ls()), 移除全部

把当前工作目录中的已存在的对象保存到 fname 文件中，`save.image(file:"fname")`.

若要将指定的对象存储：`save(x,y,file:"fname")`.加载某些存储的对象：`load(file="fname")`

退出时，R 将会询问是否需要将工作空间保存起来，若选择是，则对象被保存在当前工作目录的.Rhistory 文件中。

数据类型

最主要四种类型是数值型 (numeric), 字符型 (character), 日期型 (Date) 或 POSIXct(基于日期的)，逻辑型 (logical).

```
class(x)
```

```
## [1] "numeric"
```

```
is.numeric(x)# 类似于其他语言的 float 或 double
```

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

```
i<- 5L # 整数型

i
```

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

```
is.integer(i)
```

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

```
is.numeric(i)
```

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

```
# 当然也是数值型

class(4L)
```

```
## [1] "integer"
```

```
class(2.8)
```

```
## [1] "numeric"
```

```
class(4L*2.8)
```

```
## [1] "numeric"
```

```
class(5L)
```

```
## [1] "integer"
```

```
5L/2L
```

```
## [1] 2.5
```

```r
class(5L/2L)
```

```
## [1] "numeric"
```

字符数据

字符和因子虽然表面上类似，但处理起来完全不同。

```r
  x <- "data"
  x
```

```
## [1] "data"
```

```r
   y <- factor("data")
   y
```

```
## [1] data
## Levels: data
```

```r
    nchar(x)
```

```
## [1] 4
```

```r
    nchar("hello")
```

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

```r
  nchar(3)
```

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

```r
  nchar(452)
```

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

```r
 # nchar(y)
#Error: 'nchar()' requires a character vector
```

日期

任何一种语言中处理日期和时间都比较困难，R 具有很多不同的日期类型。Date 和 POSIXct，Date 仅存储日期，而 POSIXct 存储日期与时间，这两个对象实际上是代表自 1970 年 1 月 1 日以来的天数和秒数 (POSIXct)。

```
date1 <- as.Date("2012-06-28")
date1
```

```
## [1] "2012-06-28"
```

```
 class(date1)
```

```
## [1] "Date"
```

```
  as.numeric(date1)
```

```
## [1] 15519
```

```
   date2 <- as.POSIXct("2012-06-28 17:42")
  date2
```

```
## [1] "2012-06-28 17:42:00 CST"
```

```
   class(date2)
```

```
## [1] "POSIXct" "POSIXt"
```

```
    as.numeric(date2)
```

```
## [1] 1340876520
```

用 lubridate and chron packages包可以很容易对日期和时间进行操作处理，使用‘ as.numeric or 这样的函数不仅仅改变对象的格式，而且会改变其潜在的类型。

```
     class(date1)
```

```
## [1] "Date"
```

```
     class(as.numeric(date1))
```

```
## [1] "numeric"
```

逻辑型

```
 TRUE * 5
```

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

```
  FALSE * 5
```

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

```
   k <- TRUE
 class(k)
```

```
## [1] "logical"
```

```
  is.logical(k)
```

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

```
 TRUE
```

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

```
  T
```

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

```
 class(T)
```

```
## [1] "logical"
```

```
 T <- 7
 T
```

```
## [1] 7
```

```
  class(T)
```

```
## [1] "numeric"
```

简写 T 可代替 TRUE，但 T 是变量，可以被改写。

逻辑值的产生：

```
 # does 2 equal 3?
  2 == 3
```

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

```
   # does 2 not equal three?
  2 != 3
```

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

```
   2 < 3
```

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

```
    2 <= 3
```

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

```
     2 > 3
```

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

```
      2 >= 3
```

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

```
       # is 'data' equal to 'stats'?
 "data" == "stats"
```

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

```
  # is 'data' less than 'stats'?
  "data" < "stats"
```

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

矢量或向量

一些元素的集合，所有的元素都是同一类型。如：c(1, 3, 2, 1, 5)，c("R", "Excel","SAS", "Excel")。向量不仅是简单的容器，矢量化的语言！操作自动地应用于向量的每一个分量，不需要遍历向量的每个分量。

向量没有维数，意味着没有像列向量和行向量这样的东西，不像数学中的向量有行和列的区别。创建向量的方式是用 c，combine 合并的意思。

```
    x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
  x
```

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

```
   x * 3# 不需要循环
```

```
##  [1]  3  6  9 12 15 18 21 24 27 30
```

```
    x + 2
```

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

```
    x - 3
```

```
##  [1] -2 -1  0  1  2  3  4  5  6  7
```

```
     x/4
```

```
##  [1] 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50
```

```
      x^2
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
       sqrt(x)
```

```
##  [1] 1.000000000000000 1.414213562373095 1.732050807568877
##  [4] 2.000000000000000 2.236067977499790 2.449489742783178
##  [7] 2.645751311064591 2.828427124746190 3.000000000000000
## [10] 3.162277660168380
```

向量运算符“: ”。

```
       1:10
```

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

```
       10:1
```

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

```
       -2:3
```

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

```
       5:7
```

```
## [1] 5 6 7
```

处理等长向量。

```
   x <- 1:10
y <- -5:4
 # add them
 x + y
```

```
##  [1] -4 -2  0  2  4  6  8 10 12 14
```

```
 # subtract them
x - y
```

```
##  [1] 6 6 6 6 6 6 6 6 6 6
```

```
 # multiply them
 x * y
```

```
##  [1] -5 -8 -9 -8 -5  0  7 16 27 40
```

```
  # divide them--notice division by 0 results in Inf
x/y
```

```
##  [1] -0.2 -0.5 -1.0 -2.0 -5.0  Inf  7.0  4.0  3.0  2.5
```

```
 # raise one to the power of the other
x^y
```

```
##  [1] 1.000000000000000e+00 6.250000000000000e-02 3.703703703703703e-02
##  [4] 6.250000000000000e-02 2.000000000000000e-01 1.000000000000000e+00
##  [7] 7.000000000000000e+00 6.400000000000000e+01 7.290000000000000e+02
## [10] 1.000000000000000e+04
```

```
  # check the length of each
 length(x)
```

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

```
 length(y)
```

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

```
 # the length of them added together should be the same
 length(x + y)
```

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

处理不等长向量，较短的向量会自动补齐，。

```
  x + c(1, 2)
```

```
##  [1]  2  4  4  6  6  8  8 10 10 12
```

```
  x + c(1, 2, 3)
```

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

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

```
   # 若长向量长度不是短向量长度的整数倍时会警告
#Warning: longer object length is not a multiple of shorter object length
```

向量间比较 (逻辑表达式)

```
x <= 5
```

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

```
x > y
```

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

```
 x < y
```

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

检查是否所有元素都是 TRUE，用 all 函数；检查是否所有元素中是否存在 TRUE，用 any 函数；

```
  x <- 10:1
y <- -4:5
any(x < y)
```

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

```
all(x < y)
```

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

nchar 函数可以对向量的每一个元素进行操作

```
q <- c("Hockey", "Football", "Baseball", "Curling", "Rugby",
       "Lacrosse", "Basketball", "Tennis", "Cricket", "Soccer")
nchar(q)
```

```
##  [1]  6  8  8  7  5  8 10  6  7  6
```

```
  nchar(y)
```

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

向量元素获取

```
x[1]
```

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

```
    x[1:2]
```

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

```
     x[c(1, 4)]# 获取非连续元素
```

```
## [1] 10  7
```

小练习：1 到 20 间所有能被 4 整除的整数。

```
x<-1:20
x %% 4 == 0
```

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

```
y<-x[x %% 4 == 0] # 逻辑表达式的妙用，作为索引
y
```

```
## [1]  4  8 12 16 20
```

R 还提供 subset 函数，选取 x 的子向量，与用索引操作的一个区别是他们对缺失值 NA 的处理方式不同，subset 函数忽略缺失值，而 x[subset] 命令保持缺失值不动。

```
 x<-c(1,NA,3,4)
 x>2
```

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

```
 x[x>2]
```

```
## [1] NA  3  4
```

```
 subset(x,subset=x>2)
```

```
## [1] 3 4
```

想知道向量 x 进行逻辑运算后 TRUE 元素对应的索引位置，可使用指令 which(x)

```
x<-c(1,1,2,3,5,8,13)

which(x %% 2 == 0)
```

```
## [1] 3 6
```

可以给向量命名

```
      c(One = "a", Two = "y", Last = "r")
```

```
##  One  Two Last
##  "a"  "y"  "r"
```

```
      # create a vector
  w <- 1:3
  # name the elements
  names(w) <- c("a", "b", "c")
  w
```

```
## a b c
## 1 2 3
```

因子向量

```
   q2 <- c(q, "Hockey", "Lacrosse", "Hockey", "Water Polo","Hockey", "Lacrosse")
  #q 在前面定义过
   q2Factor <- as.factor(q2)
  q2Factor
```

```
##  [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
##  [7] Basketball Tennis     Cricket    Soccer     Hockey     Lacrosse
## [13] Hockey     Water Polo Hockey     Lacrosse
## 11 Levels: Baseball Basketball Cricket Curling Football ... Water Polo
```

```
   as.numeric(q2Factor)
```

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

打印出 q2Factor 的每个元素之后，R 也打印出 q2Factor 的水平数，一个因子的水平数是那个因子变量中不重复的元素个数，从技术上讲，R 给每个因子的唯一值为一个整数。

普通因子中水平的顺序无关紧要，一个水平和另外一个是相同的，有时，理解一个因子顺序很重要，如在编码教育水平时，设置顺序选项为 TRUE 时，会建立一个顺序因子。

```
 factor(x=c("High School", "College", "Masters", "Doctorate"),
      levels=c("High School", "College", "Masters", "Doctorate"),
        ordered=TRUE)
```

```
## [1] High School College     Masters     Doctorate
## Levels: High School < College < Masters < Doctorate
```

因子可以大幅度减少变量的大小，因为他们只存储唯一值，但是如果使用不恰当，也会让人头疼。

以向量为参数的函数

sum(),prod(),max(),min(),sqrt(),sort(),mean(),var()

里面有些函数对于向量的处理是基于元素的，而有些则是以整个向量作为输入，返回一个结果的。

```r
sqrt(1:6) # 想想看，是基于什么的？
```

```
## [1] 1.000000000000000 1.414213562373095 1.732050807568877 2.000000000000000
## [5] 2.236067977499790 2.449489742783178
```

```r
mean(1:6)
```

```
## [1] 3.5
```

```r
sort(c(5,1,3,4,2))
```

```
## [1] 1 2 3 4 5
```

做两个例题休息一下

第一个题关于统计的：设 x 为向量 (1.2,0.9,0.8,1,1.2), 写一个表达式计算 x 的均值 x.mean(为变量)；计算完毕与内置函数 mean() 比较。

第二题是关于数值积分的：用定积分定义计算 $\int_0^{\pi/6} cos(t)dt$, 用 seq 函数等分积分区间。

参考：

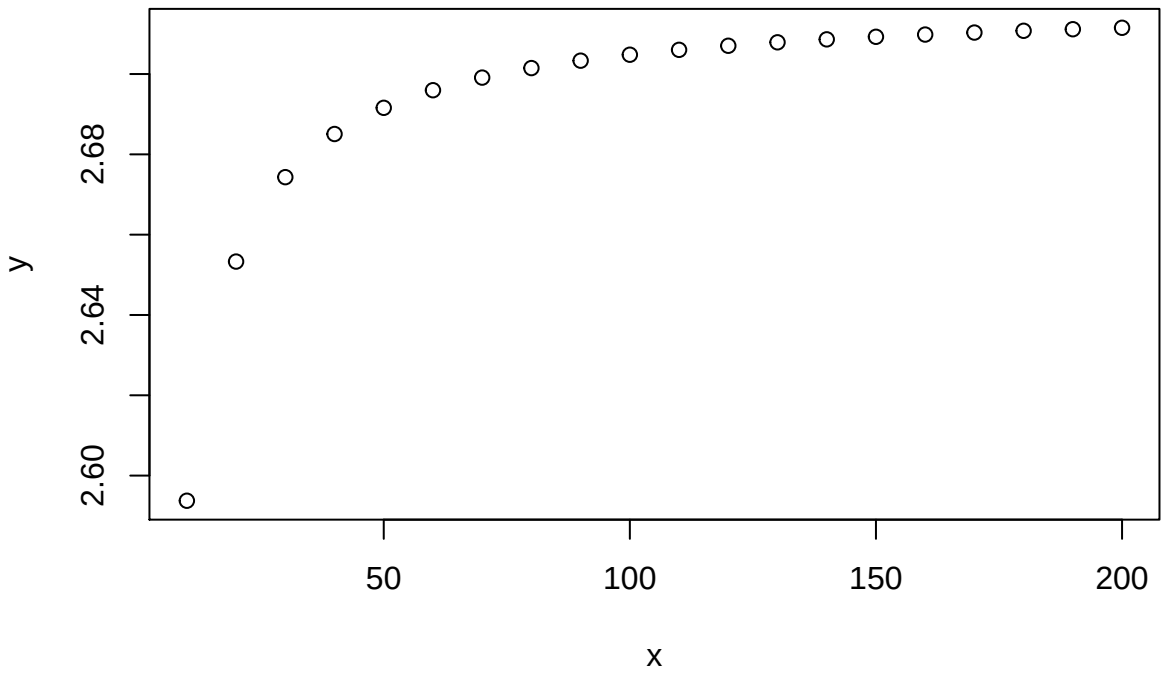
```r
dt<- 0.005 #区间长度
 t<-seq(0,pi/6,by=dt)
 ft<- cos(t) # 为向量
 (I<- sum((ft)*dt)) #sum函数直接对向量运算

 I-sin(pi/6) # 验证
```

第三题：指数形式的极限

```r
x<-seq(10,200,by=10)
y<-(1+1/x)^x
exp(1)-y
```

```
##  [1] 0.124539368359043223 0.064984123314624220 0.043963052588742890
##  [4] 0.033217990069077885 0.026693799385440364 0.022311689128826195
##  [7] 0.019165457482870796 0.016796887705710528 0.014949367400859170
## [10] 0.013467999037516165 0.012253746954282274 0.011240337596804206
## [13] 0.010381746740938169 0.009645014537912555 0.009005917124189189
## [16] 0.008446252151244504 0.007952077235182209 0.007512532619640133
## [19] 0.007119033847880374 0.006764705529752391
```

```r
plot(x,y) # 两向量的关系图，type="p"，点，"l" 线，"o" 点在线上
```



缺失数据：**NA** 和 **NULL**

```r
z <- c(1, 2, NA, 8, 3, NA, 3)
z
```

```
## [1]  1  2 NA  8  3 NA  3
```

```r
mean(z)
```

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

```r
mean(z,na.rm=TRUE)
```

```
## [1] 3.4
```

```r
is.na(z)
```

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

```r
zChar <- c("Hockey", NA, "Lacrosse")
zChar
```

```
## [1] "Hockey"    NA          "Lacrosse"
```

```r
 is.na(zChar)
```

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

```r
   z <- c(1, NULL, 3)
  z
```

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

```r
  d <- NULL
is.null(d)
```

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

```r
 is.null(7)
```

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

处理缺失数据的技术有多重添补法，参考 Andrew Gelman and Jennifer Hill 的书《Data Analysis Using Regressionand Multilevel/Hierarchical Models》，主要是用 `mi, mice and Amelia` 包实现.

NULL 是没有任何东西的意思，不算准确的缺失，而是空白，函数有时会返回 NULL，其参数也可以是 NULL，NA 和 NULL 的区别是，NULL 是最小的“原子”，其不能存在于向量中，如果在向量中使用，自然就消失了。

```r
  z <- c(1, NULL, 3)
  z# 只要两个元素，NULL 没有被存储
```

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

```r
   d <- NULL
 is.null(d)
```

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

```r
 is.null(7)
```

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

高级数据结构

有时需要存储比向量更复杂的数据结构，R 有数据框，矩阵，列表，数组。搞数学的人对矩阵比较熟悉，而程序员对列表较熟悉。

数据框

像 excel 电子表格一样有列也有行，每一列代表一个变量，每一行代表一个观测值，每一列实际上是一个向量，各列都有相同的长度，可以保存不同类型的数据，每一列里，每一个元素必须类型相同。

```
  x <- 10:1
  y <- -4:5
  q <- c("Hockey", "Football", "Baseball", "Curling", "Rugby",
         "Lacrosse", "Basketball", "Tennis", "Cricket", "Soccer")
  theDF <- data.frame(x, y, q)
  theDF
```

```
##      x  y          q
## 1   10 -4     Hockey
## 2    9 -3   Football
## 3    8 -2   Baseball
## 4    7 -1    Curling
## 5    6  0      Rugby
## 6    5  1   Lacrosse
## 7    4  2 Basketball
## 8    3  3     Tennis
## 9    2  4    Cricket
## 10   1  5     Soccer
```

可以在创建数据框的过程中设定变量名字

```
   theDF <- data.frame(First = x, Second = y, Sport = q)
 theDF
```

```
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
```

```
  nrow(theDF)# 数据框具有很多属性
```

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

```
   ncol(theDF)
```

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

```
    dim(theDF)
```

```
## [1] 10  3
```

```
     names(theDF)# 检查数据框的列名字
```

```
## [1] "First"  "Second" "Sport"
```

```
 names(theDF)[3]
```

```
## [1] "Sport"
```

```
  rownames(theDF)# 检查数据框的行名字
```

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

```
   rownames(theDF) <- c("One", "Two", "Three", "Four", "Five", "Six",
"Seven", "Eight", "Nine", "Ten")# 重新起名
 rownames(theDF)
```

```
##  [1] "One"   "Two"   "Three" "Four"  "Five"  "Six"   "Seven" "Eight"
##  [9] "Nine"  "Ten"
```

```
  rownames(theDF) <- NULL    # 去掉名字
 rownames(theDF)
```

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

```
 head(theDF)
```

```
##   First Second    Sport
## 1    10     -4   Hockey
## 2     9     -3 Football
## 3     8     -2 Baseball
## 4     7     -1  Curling
## 5     6      0    Rugby
## 6     5      1 Lacrosse
```

```
  head(theDF, n = 7)
```

```
##   First Second      Sport
## 1    10     -4     Hockey
## 2     9     -3   Football
## 3     8     -2   Baseball
## 4     7     -1    Curling
## 5     6      0      Rugby
## 6     5      1   Lacrosse
## 7     4      2 Basketball
```

```
   tail(theDF)
```

```
##    First Second      Sport
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
```

```
    class(theDF)
```

```
## [1] "data.frame"
```

```
     theDF$Sport# 第三列元素
```

```
##  [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
##  [7] Basketball Tennis     Cricket    Soccer
## 10 Levels: Baseball Basketball Cricket Curling Football ... Tennis
```

```
      theDF[3, 2]
```

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

```
      # row 3, columns 2 through 3
  theDF[3, 2:3]# 使用指标向量访问更多的行或列
```

```
##   Second    Sport
## 3     -2 Baseball
```

```
   # rows 3 and 5, column 2
  # since only one column was selected it was returned as a vector
 # hence the column names will not be printed
  theDF[c(3, 5), 2]
```

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

```
   # rows 3 and 5, columns 2 through 3
 theDF[c(3, 5), 2:3]
```

```
##   Second    Sport
## 3     -2 Baseball
## 5      0    Rugby
```

```
  # all of column 3
 # since it is only one column a vector is returned
  theDF[, 3]# 指定列而不指定行，访问整列
```

```
##  [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
##  [7] Basketball Tennis     Cricket    Soccer
## 10 Levels: Baseball Basketball Cricket Curling Football ... Tennis
```

```
  # all of columns 2 through 3
theDF[, 2:3]
```

```
##    Second      Sport
## 1      -4     Hockey
## 2      -3   Football
## 3      -2   Baseball
## 4      -1    Curling
## 5       0      Rugby
## 6       1   Lacrosse
## 7       2 Basketball
## 8       3     Tennis
## 9       4    Cricket
## 10      5     Soccer
```

```
  # all of row 2
theDF[2, ]# 指定行而不指定列，访问整行
```

```
##   First Second    Sport
## 2     9     -3 Football
```

```
  # all of rows 2 through 4
theDF[2:4, ]
```

```
##   First Second    Sport
## 2     9     -3 Football
## 3     8     -2 Baseball
## 4     7     -1  Curling
```

```
  theDF[, "Sport"]# 访问特定的列
```

```
##  [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
##  [7] Basketball Tennis     Cricket    Soccer
## 10 Levels: Baseball Basketball Cricket Curling Football ... Tennis
```

```
  theDF[, c("First", "Sport")]# 由名字访问列
```

```
##    First      Sport
## 1     10     Hockey
## 2      9   Football
## 3      8   Baseball
## 4      7    Curling
## 5      6      Rugby
## 6      5   Lacrosse
## 7      4 Basketball
## 8      3     Tennis
## 9      2    Cricket
## 10     1     Soccer
```

```
  class(theDF[, "Sport"])# 返回向量
```

```
## [1] "factor"
```

```
   class(theDF["Sport"])# 单列数据框
```

```
## [1] "data.frame"
```

```
    theDF[["Sport"]]
```

```
##  [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
##  [7] Basketball Tennis     Cricket    Soccer
## 10 Levels: Baseball Basketball Cricket Curling Football ... Tennis
```

```
     class(theDF[["Sport"]])
```

```
## [1] "factor"
```

```
      theDF[, "Sport", drop = FALSE]
```

```
##         Sport
## 1      Hockey
## 2    Football
## 3    Baseball
## 4     Curling
## 5       Rugby
## 6    Lacrosse
## 7  Basketball
## 8      Tennis
## 9     Cricket
## 10     Soccer
```

```
      # 使用单方括号时，为确保返回一个单列数据框，输入一个 drop 参数

       class(theDF[, "Sport", drop = FALSE])
```

```
## [1] "data.frame"
```

```
        theDF[, 3, drop = FALSE]
```

```
##         Sport
## 1      Hockey
## 2    Football
## 3    Baseball
## 4     Curling
## 5       Rugby
## 6    Lacrosse
## 7  Basketball
## 8      Tennis
## 9     Cricket
## 10     Soccer
```

```
class(theDF[, 3, drop = FALSE])# 数据框
```

```
## [1] "data.frame"
```

```
newFactor <- factor(c("Pennsylvania", "New York", "New Jersey", "New York", "Tennessee"
model.matrix(~newFactor - 1)
```

```
##   newFactorMassachusetts newFactorNew Jersey newFactorNew York
## 1                      0                   0                  0
## 2                      0                   0                  1
## 3                      0                   1                  0
## 4                      0                   0                  1
## 5                      0                   0                  0
## 6                      1                   0                  0
## 7                      0                   0                  0
## 8                      0                   0                  1
##   newFactorPennsylvania newFactorTennessee
## 1                     1                  0
## 2                     0                  0
## 3                     0                  0
## 4                     0                  0
## 5                     0                  1
## 6                     0                  0
## 7                     1                  0
## 8                     0                  0
## attr(,"assign")
## [1] 1 1 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$newFactor
## [1] "contr.treatment"
```

因素被存储为矩阵的特殊形式，也可以被表示成数据框的形式。用 model.matrix 创建一个指示性的变量 (哑变量) 的集合，列代表一个因素的水平，每行里，对应哪个水平是 1，其他都是 0. model.matrix 用法请参考帮助信息。

列表

该容器可以存储任意相同类型或不同类型的对象

```
 # creates a three element list
list(1, 2, 3)
```

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

```
  # creates a single element list where the only element is a vector
  # that has three elements
  list(c(1, 2, 3))
```

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

```
  # creates a two element list
  # the first element is a three element vector
  # the second element is a five element vector
  (list3 <- list(c(1, 2, 3), 3:7))
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 3 4 5 6 7
```

```
  # two element list
  # first element is a data.frame
  # second element is a 10 element vector
  list(theDF, 1:10)
```

```
## [[1]]
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
  # three element list
 # first is a data.frame
 # second is a vector
 # third is list3, which holds two vectors
 list5 <- list(theDF, 1:10, list3)
 list5
```

```
## [[1]]
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
```

```
## 4       7     -1    Curling
## 5       6      0      Rugby
## 6       5      1   Lacrosse
## 7       4      2 Basketball
## 8       3      3     Tennis
## 9       2      4    Cricket
## 10      1      5     Soccer
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[3]]
## [[3]][[1]]
## [1] 1 2 3
##
## [[3]][[2]]
## [1] 3 4 5 6 7
```

与数据框一样，列表也有名字

```
   names(list5)
```

```
## NULL
```

```
 names(list5) <- c("data.frame", "vector", "list")
  names(list5)
```

```
## [1] "data.frame" "vector"     "list"
```

```
 list5
```

```
## $data.frame
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
##
## $vector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $list
## $list[[1]]
## [1] 1 2 3
##
## $list[[2]]
## [1] 3 4 5 6 7
```

利用姓名—值创建列表的时候，名字也可以分配给列表元素。

```
  list6 <- list(TheDataFrame = theDF, TheVector = 1:10, TheList = list3)
names(list6)
```

```
## [1] "TheDataFrame" "TheVector"    "TheList"
```

```
  list6
```

```
## $TheDataFrame
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
##
## $TheVector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $TheList
## $TheList[[1]]
## [1] 1 2 3
##
## $TheList[[2]]
## [1] 3 4 5 6 7
```

令人惊奇的是，可以创建一个具有一定长度的空列表。

```
   (emptyList <- vector(mode = "list", length = 4))
```

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

通过设定元素编号或元素名字，利用双方括号，可以访问列表的单个元素。一次只能访问一个元素。

```
   list5[[1]]
```

```
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
```

```
   list5[["data.frame"]]
```

```
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
```

实际上如果使用的元素在一个被访问的元素中，可以通过元素嵌套索引来访问。

```
    list5[[1]]$Sport
```

```
##  [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
##  [7] Basketball Tennis     Cricket    Soccer
## 10 Levels: Baseball Basketball Cricket Curling Football ... Tennis
```

```
    list5[[1]][, "Second"]
```

```
##  [1] -4 -3 -2 -1  0  1  2  3  4  5
```

```
     list5[[1]][, "Second", drop = FALSE]
```

```
##    Second
## 1      -4
## 2      -3
## 3      -2
## 4      -1
## 5       0
## 6       1
## 7       2
## 8       3
## 9       4
## 10      5
```

利用一个并不存在的指标(数字或名字)，可以简单地对列表增加元素。

```
    length(list5)
```

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

```
     # add a fourth element, unnamed
list5[[4]] <- 2
 length(list5)
```

```
## [1] 4
```

```
  # add a fifth element, named
list5[["NewElement"]] <- 3:6
 length(list5)
```

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

```
  names(list5)
```

```
## [1] "data.frame" "vector"     "list"       ""           "NewElement"
```

```
  list5
```

```
## $data.frame
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
##
## $vector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $list
## $list[[1]]
## [1] 1 2 3
##
## $list[[2]]
## [1] 3 4 5 6 7
##
##
## [[4]]
## [1] 2
##
## $NewElement
## [1] 3 4 5 6
```

偶尔对列表或向量或数据框增加元素都还好，但如果反复这样做计算代价就太高，最好是创建一个具有期望长度的列表，然后用合适指标填充。

矩阵

与数据框有点象，是一个矩形结构，具有行和列，每一个单一元素必须类型相同，一般都为数值型，跟向量相似，也可以进行元素对元素的加法，减法，乘法，除法等式等运算，函数 nrow，ncol,dim 也跟数据框一样使用。

乘法 * 对于矩阵来说是按元素进行的，若要进行矩阵乘法，运算符为 %*%,除此之外，还有很多专门的函数是针对矩阵的，nrow(),ncol(),det(x),t(x).

```
  # create a 5x2 matrix
A <- matrix(1:10, nrow = 5)
# create another 5x2 matrix
B <- matrix(21:30, nrow = 5)
# create another 5x2 matrix
C <- matrix(21:40, nrow = 2)
A
```

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

```
B
```

```
##      [,1] [,2]
## [1,]   21   26
## [2,]   22   27
## [3,]   23   28
## [4,]   24   29
## [5,]   25   30
```

```
C
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   21   23   25   27   29   31   33   35   37    39
## [2,]   22   24   26   28   30   32   34   36   38    40
```

```
nrow(A)
```

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

```
ncol(A)
```

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

```
dim(A)
```

```
## [1] 5 2
```

```
A + B
```

```
##      [,1] [,2]
## [1,]   22   32
## [2,]   24   34
## [3,]   26   36
## [4,]   28   38
## [5,]   30   40
```

```
  # multiply them
A * B
```

```
##      [,1] [,2]
## [1,]   21  156
## [2,]   44  189
## [3,]   69  224
## [4,]   96  261
## [5,]  125  300
```

```
 # see if the elements are equal
A == B
```

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

矩阵乘法

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

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  177  184  191  198  205
## [2,]  224  233  242  251  260
## [3,]  271  282  293  304  315
## [4,]  318  331  344  357  370
## [5,]  365  380  395  410  425
```

另一个跟数据框相似的是，矩阵也可以由行列的名字

```
 colnames(A)
```

```
## NULL
```

```
   rownames(A)
```

```
## NULL
```

```
    colnames(A) <- c("Left", "Right")
  rownames(A) <- c("1st", "2nd", "3rd", "4th", "5th")

  colnames(B)
```

```
## NULL
```

```
  rownames(B)
```

```
## NULL
```

```
  colnames(B) <- c("First", "Second")
   rownames(B) <- c("One", "Two", "Three", "Four", "Five")

 colnames(C)
```

```
## NULL
```

```
  rownames(C)
```

```
## NULL
```

```
 colnames(C) <- LETTERS[1:10]
  rownames(C) <- c("Top", "Bottom")
```

两个特殊的向量 letters and LETTERS 分别表示小写字母和大写字母。

```
   t(A)
```

```
##       1st 2nd 3rd 4th 5th
## Left    1   2   3   4   5
## Right   6   7   8   9  10
```

```
   A %*% C
```

```
##       A   B   C   D   E   F   G   H   I   J
## 1st 153 167 181 195 209 223 237 251 265 279
## 2nd 196 214 232 250 268 286 304 322 340 358
## 3rd 239 261 283 305 327 349 371 393 415 437
## 4th 282 308 334 360 386 412 438 464 490 516
## 5th 325 355 385 415 445 475 505 535 565 595
```

转置或置换行和列的名字。矩阵乘法，行名字保留左边矩阵的行名字，列名字保留右边矩阵的列名字。

问题研究：解矩阵方程 $AX = B$.

solve(A,B) 返回满足方程上述方程的 x 值，若 A 为可逆矩阵，则 solve(A) 返回 A 的逆矩阵。

```r
A<-matrix(c(3,5,2,3),nrow=2,ncol=2)
B<-matrix(c(1,1,0,1),nrow=2,ncol=2)
A %*% B
```

```
##      [,1] [,2]
## [1,]    5    2
## [2,]    8    3
```

```r
A * B
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    5    3
```

```r
A[2] # 访问矩阵元素，以列存储
```

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

```r
A.inv<- solve(A)

A %*% A.inv
```

```
##      [,1]                    [,2]
## [1,]    1 -8.881784197001252e-16
## [2,]    0  1.000000000000000e+00
```

```r
A^(-1)
```

```
##                      [,1]                [,2]
## [1,] 0.3333333333333333 0.5000000000000000
## [2,] 0.2000000000000000 0.3333333333333333
```

可见，A %*% A.inv 值有些小误差.

is.matrix(),is.vector(),在数学上，向量也是一个行数或列数为 1 的矩阵，但 R 将矩阵和向量作为两个不同的类型的对象来处理，如果要用向量 x 来生成一个列数为 1 的矩阵 A，可用命令 A<- as.matrix(x)，注意此变换并不改变 x 的值。

若将矩阵按列生成一个向量，可以 as.vector(A)，此操作仅仅是将存储在 A 中的维度属性删除掉而已，其元素的存储保持不变 (矩阵按列存储), 这个转换对象的过程称为类型强制转换。

数组

本质上是一个多维向量，所有元素必须是相同类型，利用方括号进行单个元素的访问，方括号的第一个参数是行指标，第二个参数是列指标，剩下的参数代表其他维数。

```
  theArray <- array(1:12, dim = c(2, 3, 2))
theArray
```

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

```
  theArray[1, , ]
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    3    9
## [3,]    5   11
```

```
   theArray[1, , 1]
```

```
## [1] 1 3 5
```

```
    theArray[, , 1]
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

数组和矩阵最主要的不同是，矩阵限制为两维，而数组可以有任意维数。

最常见的数据结构是一维的向量，它是构成 **R** 中一切的基础。最强大的结构是数据框，大多数其他语言没有，以类似于电子表格的方式处理混合类型的数据。列表在存储数据集合方面比较有用，类似于 **perl** 中的 **hash** 关联数组。

如有错误请联系 [e-mail]:shixiangbupt@qq.com.

参考文献

- 吴喜之. 应用时间序列分析–R 软件陪同. 人民大学出版社
- 王亮等译. R 语言的科学编程与仿真. 西北工业大学出版社
- 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.
- 王斌会. 多元统计分析及 R 语言建模. 暨南大学出版社.201405.
- 胡伟. LaTeX 2_ε 完全学习手册. 清华大学出版社，201101.