2021年数理统计上机课

–循环与函数编写

黄启岳

北京师范大学统计学院

2021 年 3 月 31 日

目录

目的

循环和函数是减少代码重复的重要工具。很多场合我们需要重复执行很多相同的命令，一次又一次不必要的重复劳动(duplication)既不快乐也不现实，浪费的是自己宝贵的时间与精力。

幸运的是，循环与函数两种存在直接将我们从繁重的体力劳动中解放出来，让精力得以集中在更应该被关注的领域。就这一点而言，其意义相当于“工业革命”。

Hadley Wickham总结了三个减小不必要重复的原因：

- It's easier to see the **intent** of your code, because your eyes are drawn to what's **different**, not what stays the same.
- It's easier to **respond to changes** in requirements. As your needs change, you only need to make changes in one place, rather than remembering to change every place that you copied-and-pasted the code.
- You're likely to have **fewer bugs** because each line of code is used in more places.

循环

相比于多数“下拉菜单”的软件，R语言可以通过循环的方式将重复的简单操作程序化并批量执行。换句话说，循环操作相当于把“加法”上升为“数乘”。

常见的循环种类有以下三种：

- for、while和repeat循环
- apply函数族
- 泛型函数map

FOR循环

首先从最基本的for循环开始。for循环主要由输出层(output)、迭代器(sequence)和循环体(body)组成。

- 输出层一般用于存储程序的运算结果，即每一次循环的结果需要被输出层“收纳”。
- 迭代器用于确定“怎么循环”。一般在for之后的小括号里，记录符号常用i表示(i指代it)。
- 循环体就是希望被重复执行的代码，这部分会随着迭代器变化(i变化)不断重复运行。

这里使用例子说明。案例需要求100个服从标准正态分布随机数的中位数，并求这些中位数的中位数。图1标注的位置显示了for循环的结构。

```
3  vec <- rep(0,1000)                    #输出层
4  for(i in 1:length(vec)){              #迭代器
5    a = rnorm(100)
6    vec[i] = median(a)                  #循环体
7    cat("  The loop has been carried out",i,"times.\r")
8  }
9  median(vec)
10
```

图 1: for循环结构

注意迭代器本质上是按照“数”的更替相应运行。

提到这一点主要是因为R语言没有直接对字符串包括文本循环的能力。有的语言如python中的for循环是可以直接对字符型数据或文本进行的，例如图2：

```
24 targets = []
25
26 for line in open(r'C:\Users\lenovo\Desktop\UNIVERSI
27     category, review = line.strip().split('\t')
28     corpus.append(review)
29     if category == 'N':
30         targets.append(0)
31     elif category == 'P':
32         targets.append(1)
33     else:
34         print('error', line)
35
```

图 2: python中对字符的循环

针对输出层，一般建议预设好存储空间再进行循环，否则R会反复转存信息，造成循环运行变慢，也比较浪费内存。当然在程序相对不复杂的情形下性能差距并没有那么显著。

```
out3 <- c()                                  #输出层
for(i in seq_along(chart)){                  #迭代器
  out3 = c(out3,median(chart[[i]]))          #循环体
}
out3
```

图 3: 反复转存的写法

条件循环

有的时候并不能准确得知循环过程需要使用的存储空间大小，甚至不知道循环具体需要多少步才能结束，这种情况可以使用条件循环，即循环在一定条件下停止。事实上，这种情况在统计模拟(simulation)中相对于指定循环次数应用更加广泛。

这里主要介绍两种用于条件循环的函数while和repeat。这两种函数在R语言中的功能和在其他函数中的功能相差不大。

条件循环函数

while函数一般在while之后的小括号中加入循环的条件，而repeat则是在循环体中嵌入脱出循环的条件语句。这两个函数相对于for()最主要的区别就是跳出循环的条件需要设定。

以下通过梯度下降法简单描述构造两种循环的方法。

案例：梯度下降法

梯度下降法简单来说就是一种寻找目标函数最小化的方法。从数学上说，一个函数导函数取绝对值值最大的方向为梯度方向，这也是“下降”速度最快的方向，经由这个方向可以最快速到达函数极值点。

由此给出定义：

DEFINITION (梯度下降法)

针对某个$\theta \in \mathbb{K}^n$(通常是$\mathbb{R}^n$)的函数$J(\theta) \in C^m(\Omega)$，$\epsilon_0$为给定常数，求解$J(\theta)$的局部极值可以由以下迭代算法完成：

$$\theta^{(i+1)} = \theta^{(i)} - \alpha \nabla J(\theta^{(i)})$$

其中α称为学习率(步长)，∇表示求梯度的微分算子。若$\theta^{(i+1)}$满足

$$||\theta^{(i+1)} - \theta^{(i)}|| < \epsilon_0$$

迭代停止，找到$J(\theta)$的一个局部极小值。反向过程称为梯度上升法，常用于求局部极大值。

图4给出了梯度下降在一维条件下的几何解释。

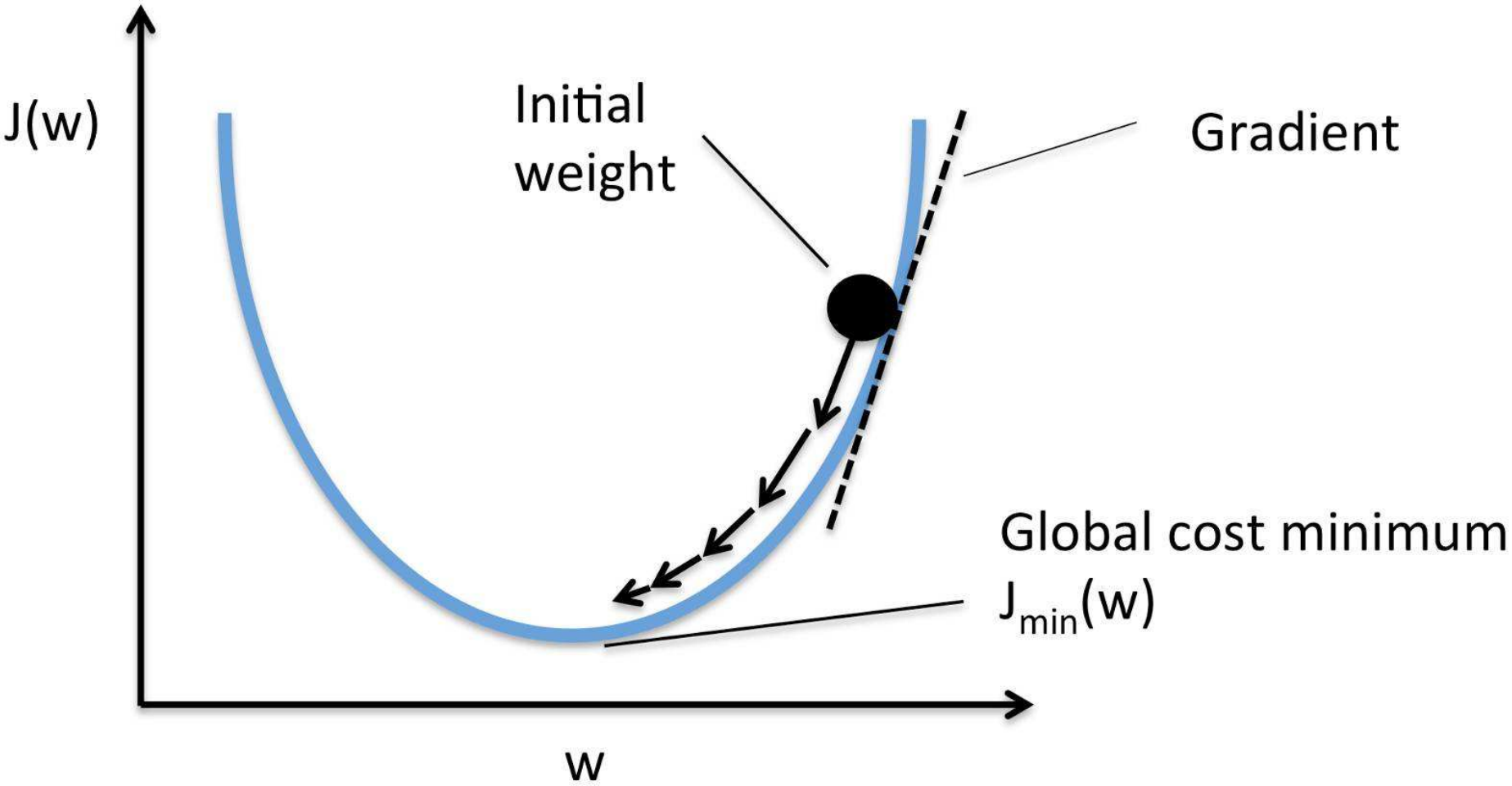

图 4: 梯度下降图解

以下以一维实数空间上的函数$f(x) = 0.4 \ln x + 3 \sin x$为例，其导函数为：

$$\frac{\mathbf{d}}{\mathbf{d}x} f(x) = \frac{2}{5x} + 3 \cos x$$

二阶导数为:

$$\frac{\mathbf{d}^2}{\mathbf{d}x^2} f(x) = -\frac{2}{5x^2} - 3 \sin x$$

这个函数本身定义在$(0, +\infty)$上，根据导函数形式直接求驻点和拐点难度相对较大。这里采用梯度下降法求解极小值，即：

$$x_{j+i} = x_j - \alpha(\frac{2}{5x_j} + 3 \cos x_j)$$

具体编程过程见R程序。

APPLY函数族

apply系列函数主要包括apply()，lapply()，sapply()，vapply()和mapply()，这几个函数与泛型函数map()、reduce()都是函数型编程的主要函数。

这其中vapply(),sapply(),lapply()的用法一致，只是返回的结果分别为向量、矩阵、列表。

格式为：v(s)(l)apply(列表, 函数, 函数的剩余参数)

运行的结果是:按列表的顺序输入函数的第一个位置的参数，得到结果，以向量（矩阵，列表）形式保存。

其中函数参数可以是对象，也可以是函数或者方法，函数的剩余参数如果有，必须指定，同时列表只能传入第一个位置，与管道函数的机制类似。

各个apply()函数族中函数的关系如图5所示，不仅限于我们提到的那几种。

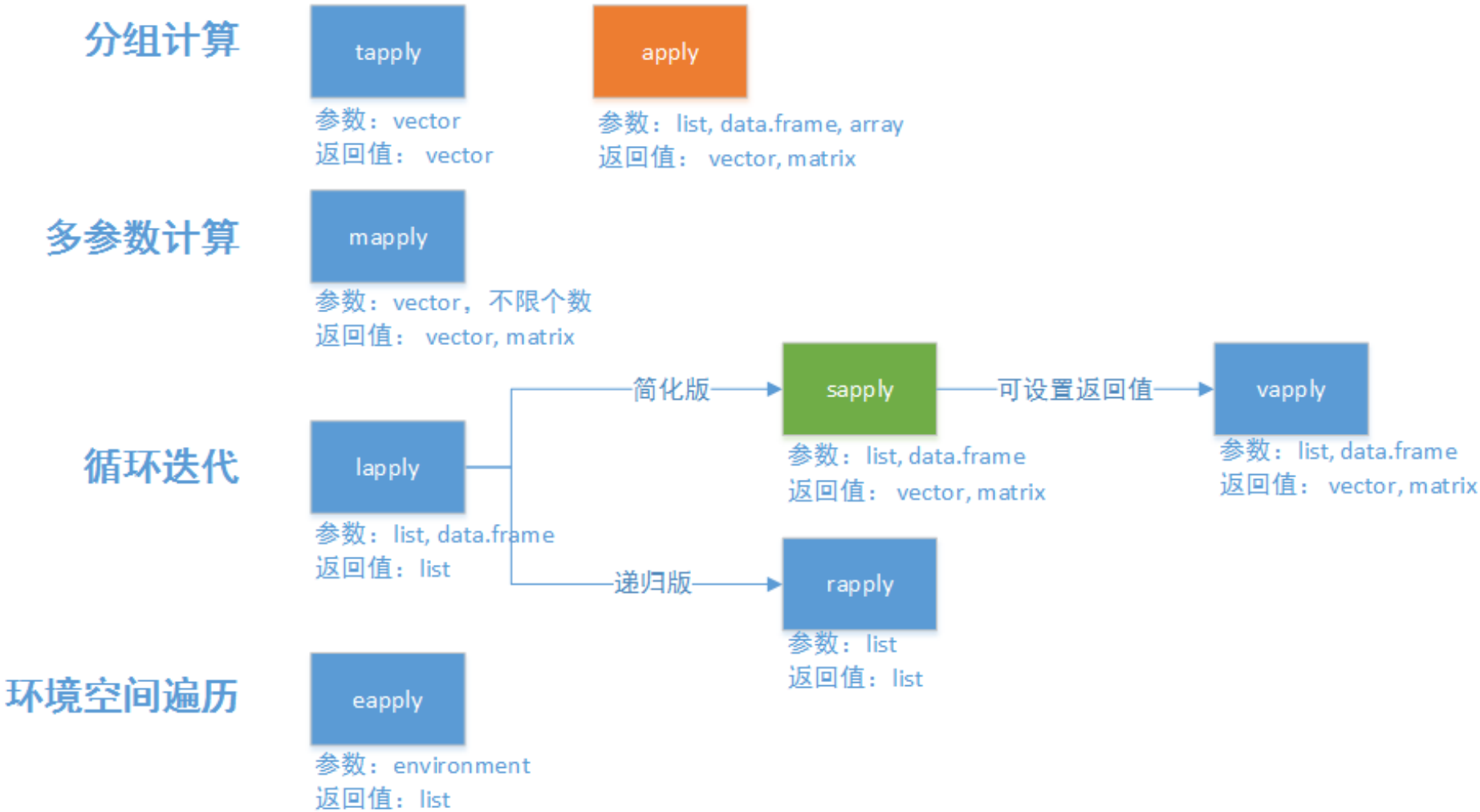

图 5: apply函数族

APPLY

- apply(数组、矩阵、数据框, MARGIN = 1(行) or 2(列)，FUN = 函数)

  效果相当于for循环。apply函数可以对矩阵、数据框、数组(二维、多维)，按行或列进行循环计算，对子元素进行迭代，把子元素以参数传递的形式给自定义的FUN函数中，并以返回计算结果。

LAPPLY

- lapply(list、数据框, FUN = 函数,...)
  lapply函数是一个最基础循环操作函数之一，用来对list、data.frame进行循环，返回和X长度同样的list结构作为结果集，通过lapply的开头的第一个字母"l"可以判断返回结果集的类型。
  注意lapply()针对列表和数据框效果较好，但针对向量和矩阵效果就相对不好了。例如对矩阵，lapply会分别循环矩阵中的每个值，而不是按行或按列进行分组计算(相当于失效)。

SAPPLY

- sapply(X, FUN, ..., simplify=TRUE, USE.NAMES = TRUE)
  其中X为数组、矩阵、数据框；FUN为定义的函数；simplify代表是否数组化，当值为simplify = "array"时，输出结果按数组进行分组；USE.NAMES为T时，如果X为字符串E设置字符串为数据名，FALSE不设置。
  sapply()相当于简化版的lapply()，如果simplify与USE.NAME全部设置为F其相当于lapply()。包括lapply()的特性sapply()也会一并继承。

VAPPLY

- vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)
  vapply()类似于sapply()，相比于sapply()提供了FUN.VALUE参数，用来控制返回值的行名，即定义返回值的行名row.names。

MAPPLY

- mapply(FUN, list1, list2, ..., MoreArgs = NULL, SIMPLIFY = TRUE,USE.NAMES = TRUE)

  mapply()相当于多变量版的sapply()。有时一个函数可能有多个参数，直接使用apply()系列进行嵌套或许可以完成，即：

  v(s)(l)apply(列表1,function(i,j)lapply(列表2, 函数, 参数2=j))

  但循环次数相当于dim(*list*1) $\times$ dim(*list*2)次。

  使用mapply()则只需要将函数与参数列表输入即可，参数列表为函数的参数调用。这样程序相对简单，可读性更高。

REDUCE

- Reduce(f, x, init, right = FALSE, accumulate = FALSE)

  函数或者算符需要为二元函数或者面向两个对象的算符，结果是将列表中的元素，按照顺序不断复合迭代函数或者算符。

  针对选项accumulate，如果赋值为T，则会输出每次复合迭代的结果。

  如对于函数f(x,y)，Reduce(f,1:3)表示f(f(1,2),3).

PURRR包

purrr包中提供了一些更简捷的函数型编程工具，且所有的purrr函数都是用C实现的，因此速度非常快。

简单总结常用函数如下：

- map(list(x), function(x)):这样可以对一列x进行操作。
- map2()与pmap()(invoke_map()):两个或多个参数。
- reduce()与accumulate()：同Reduce()和Reduce(, accumulate = T)

函数编写

封装函数是循环之外另一种减少代码无意义重复的工具。将重复的部分封装命名为一个函数，之后需要时调用函数名称即可，使得程序“模块化”，以及更好地适应参数变化，即不需要因为调整一个参数而大规模修改程序。

封装函数的优势

Hadley Wickham总结了三个封装函数的优势：

- You can give a function an evocative name that makes your code **easier to understand**.
- As requirements change, you only need to **update** code **in one place**, instead of many.
- You **eliminate** the chance of making incidental **mistakes** when you copy and paste (i.e. updating a variable name in one place, but not in another).

封装方法

R中最常见的封装函数的方法是使用function()函数。格式如下：

函数名<- function(变量){函数体}

变量一般指整个程序中发生变动的部分，类似于数学中函数定义域中的元素；函数体则是整个程序，相比于正常的可直接运行的程序仅仅缺少了变量赋值。

一般封装一个函数后，存储区会在Functions栏显示已经封装好的函数，使用时只要输入变量即可，与一般的R函数使用相同。

调用方法

函数调用方式主要有两种：一种是在主程序之前写好函数并存储，这种在现阶段常用；还有一种方法是调用存储在其他R文件中的函数，一般使用source()函数调用，括号里填写R文件所在地址，常见于更大规模的编程。

*泛型函数(选讲)

泛型函数是一种特殊的函数,其根据传入对象的类型决定调用哪个具体的子函数。根据定义可以看出泛型函数实质上是面向对象编程。

面向对象编程(object-oriented programming)是一种编程范式。它将对象作为程序的基本单元，将程序和数据封装(encapsulate)其中，以提高软件的重用性,灵活性和扩展性。

*补充：S3类和S4类

R语言的类型系统相对于一般语言而言要复杂很多，一般来说，官方制定的类型系统有四种：基础类型、S3类型、S4类型和RC类型。

S3类：使用已有对象，是R中最常见的类。S3类方法属于函数而不属于对象，常用"."调用。例如plot.ecdf()。

S4类：创建新的对象，方法之间存在严格的继承关系，并且可由多个参数指定方法。常用setClass()创建。

R中S3对象的方法和S4类的方法是通过泛型函数机制关联到目标，方法通过S3和S4泛型函数机制绑定到S3对象和S4类上。简言之，对于不同的输入类型，泛型函数调用不同的子函数处理。

*泛型函数的查看

一般查看对象所属的函数类使用pryr包中的otype()与ftype()函数查看。

泛型函数中子函数的查找常用methods()函数。例如对plot函数，可以用methods(plot)查看其所有子函数。源代码查看相比于一般函数较复杂，可以使用getAnywhere()查找，如getAnywhere(plot.ecdf)。

*泛型函数的编写

泛型函数编写主要使用函数UseMethod()，之后针对每一个子函数用"."分隔后分别编写。

注意在对一般函数命名时，尽可能不要使用"."，否则容易与泛型函数混淆。

函数命名

应当注意到函数不仅面向电脑，还面向使用者。理想状态下函数命名应该“言简意赅”，尽可能简短。

函数名称多使用动词，变量名称则较多使用名词，但当单个名词可以表达清楚函数内容时也可以使用名词。这需要程序员们合理把握。

性能测试与优化

当循环完成了，函数封装完毕，需要检测函数性能。鉴于现阶段编写的程序规模相对较小，不需要用到太多复杂的优化方法，只介绍两个用于计算时间的函数，分别为system.time()和microbenchmark()。

MICROBENCHMARK

需要装载包microbenchmark。函数格式为：microbenchmark(函数1, 函数2, 函数3,...,times = , unit = , check =)

函数运行的结果是这些待测试的函数在times 次运行中，所需时间的统计信息，times 输入的是正整数, 表示需要测试的次数。unit控制的是函数显示的部分内容，默认显示的是以每次运行的时间统计，设置为"eps"后为每秒运行的次数。check默认为NULL，当输入all.equal或者identical时，可以检验待测试结果是否相等。

相应的，system.time()只需包含待测试函数即可。

性能提升：向量化

在R中，很多情况下循环和控制结构可以通过向量化避免(简化)：向量化使得循环隐含在表达式中。当使用向量化表达时，R会自动调用C语言进行计算，一般C运行比R快100倍。

举例：

apply系列函数即是向量化了循环操作；

针对判断语句，可以改为针对向量某一位进行判断，如使用y(x==b) <- 0代替for(i in 1:length(x)){if(x[i]==b){y[i]=0}}

完整程序框架

编写一套完整的程序往往有以下步骤。

- 明确目标：编程的目的。
- 设计算法：如何计算出目标值。
- 编程实现：正确将算法转化为程序。
- 性能评价：debug与适当优化。
- 规范输出：添加适当的注释以及整理输出结果。
- 代码模块化：将实现不同功能的部分封装为函数，主程序调用函数即可。

练习：估计几组数据的统计量

生成服从特定分布并存在特定线性关系的随机数，并分别估计基于梯度下降法计算的最小二乘解估计。