统计计算及其R实现笔记

Statistical Computation with R

马学俊[0]

[0]副教授，苏州大学数学科学学院统计系，主要从事海量数据分析、高维数据分析、统计计算、非参数回归等统计模型及其应用等研究。个人主页https://xuejunma.github.io.

献给我的家人、恩师和所有在学术道路上帮助我的人

缩写与记号

- s.t. ：约束
- $\boldsymbol{Y}$：向量
- $\mathbf{X}$：矩阵（不包含向量）
- $\mathbb{R}^k$：k维欧式空间
- $\top$：转置

目　录

表　　格

第一部分

R语言

第1章　R语言

这一部分，我们将详细阐述R的高级编程。主要参考Wickham, H. (2014) [1].

1.1　简介

- R是为统计分析、绘图和数据挖掘而产生的语言和环境，它由Ross Ihaka和Robert Gentleman 开发，现在由“R开发核心团队”负责开发。它的老祖先是S语言(John Chamber 和他的同事，贝尔实验室)，通常用S语言编写的代码都可以不作修改的在R环境下运行

- R软件的主页是`https://www.r-project.org/`，您可体验一下R网址的朴实而华丽，平凡而伟大。如果您打算和它相识，那么这将是最明智的选择，因为它会让您知道什么什么是自由、什么是奉献、什么是“走自己的路让别人也走自己的路”。

R不仅仅是一个软件，更是一个学习方式和生活态度。学习它，不仅仅可以提高我们的业务水平或者学术水平，也让我们深深感受到自由而快乐，奉献的快乐。写一串代码实现一个算法或模型，写一个包实现一个想法，留下的不是代码而是对知识和生活的享受和快乐。下面，我们一起看看R的朋友圈：

- 体积小。它只几十MB

- 免费，任何人都可以下载并且使用。

- 学习的资源多，算法更新非常快。如：

  - `https://blog.rstudio.org/`
  - `https://www.r-bloggers.com/`
  - `http://www.jstatsoft.org/`

[1] Wickham, H. (2014). Advanced R. CRC Press.

1.1.1 安装R包

我们在它主页上下载R并安装。R包是R软件的细胞。例如：

```
install.packages(“quantreg”)
```

安装quantreg。需要注意安装包并不意味着可以使用该包函数，必须使用

```
library(quantreg)
```

将quantreg载入运行环境。比如我们利用该包rq建立分位数模型。

```
library(quantreg)
quantreg::rq
```

A::B表示使用A包中的函数B。这个写法虽然比较麻烦，但有时很重要。因为有时不同的包的函数名可能相同。

1.1.2 查看帮助文档

?x可以查看函数的使用方法比如，我们可以是使用?mean查看mean函数的帮助手册。它主要包括以下内容：

Title 标题：主要函数的用途，如mean的标题是Arithmetic Mean

Description 描述：更加详细介绍函数的用途

Usage 用法：函数的使用方法，如mean的用法是mean(x, ...)

Arguments 参数设置：函数的参数进行怎么设置或者要求

References 参考文献：函数的参考文献。这一部分很重要，我们通过它可以看到该函数或实现的模型的参考文献，可以进一步了解函数或者模型的表达、求解等细节，有助于我们学习新的方法或者模型。

Examples 例子：给一个实际或模拟数据的实现

x查些函数源代码。有些函数可以看到，有些看不到。如mean看不到源代码，lprq 可以看到源代码。

```
> library(quantreg)
> mean
function (x, ...)
UseMethod("mean")
<bytecode: 0x0000000016cc6cd8>
<environment: namespace:base>
> quantreg::lprq
function (x, y, h, tau = 0.5, m = 50)
{
    xx <- seq(min(x), max(x), length = m)
    fv <- xx
    dv <- xx
<environment: namespace:quantreg>
```

除了上面这些还有：

getwd() ：显示当前工作目录

setwd()：修改当前工作目录。比如，我们处理的数据文件都在E中myr文件中，我们可以设置setwd("E:/myr")

source() :运行R的脚码(Script)。比如运行E中myr文件中my.R脚码，我们可以用source("E:/myr/my.R")

1.2　R编程风格

有时，我们感觉修改别人代码比自己写还要麻烦，其中有部分原因就是编程的习惯不一样。编程的风格人人不同，但分享代码时，需要一致的表达。一般来说，看一段代码就大概知道编程者的水平。着犹如《一代宗师》，无形胜有形呀，高手交手，点到为止。

formatR可以实现整理代码。常见方法有：

第一种：打开https://yihui.name/formatr/，如图1.2 所示，复制下面代码，然后在点击Tidy My Code可以自动修改代码格。如：

图 1.1: 一代宗师

```
1+1

if(TRUE){
  x=1  # inline comments
}else{
  x=2;print('Oh no... ask the right bracket to go away!')}
1*3 # one space before this comment will become two!
2+2+2    # only 'single quotes' are allowed in comments

lm(y~x1+x2, data=data.frame(y=rnorm(100),x1=rnorm(100)))
1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
```

第二种：我们也可以通过加载formatR实现。将上面代码存放在test1.r中，然后运行下面命令，也可以实现整理代码的效果。

```
install.packages("formatR",
                 repos = "http://cran.rstudio.com")
library(formatR)
#tidy_source(test1.r)
formatR::tidy_app()
```

下面，我们介绍常见的编程规则。

图 1.2: formatR 界面

1.2.1　符号和名字

- 文件名：有一定意义。如果按照顺序执行，最好在命名前面加上数字前缀。

```
# 好
fit-models.R
utility-functions.R
# 不好
foo.r
stuff.r
0-download.R
1-parse.R
2-explore.R
```

- 对象名：变量和函数名称应该小写字母，使用_将名字单词分开。
- 变量名最好是名词，函数名最好是动词。
- 避免使用系统的函数或特殊意义的字母命名

```
# 好
day_one
day_1 # 不好
first_day_of_the_month
DayOne
dayone
djm1

T <- FALSE
c <- 10
mean <- function(x) sum(x)
```

1.2.2 语法

1.2.2.1 空格

- 空格：是个好东西
- 在中缀操作符($+, -, *, , < -$等)两边都留空格。在函数调用里使用=时，也使用同样的规则。
- 逗号后面留一个空格，而逗号前面则不要留. 前留后不留。
- :, :: 两侧都不需要空格

```
# 好
average <- mean(feet / 12 + inches, na.rm = TRUE)
# 不好
average<-mean(feet/12+inches,na.rm=TRUE)

# 好
x <- 1:10 base::get
# 不好
x <- 1 : 10 base :: get
```

- 在左括号前面留一个空格(函数调用除外）

- 圆括号或者方括号内的代码两边留空格（除非有逗号）
- $<-$ 前后根据需要可以任意添加空格

```
# 好
if (debug) do(x)
plot(x, y)
diamonds[5, ] #

不好
if ( debug ) do(x) # debug两边不要留空格
if(debug)do(x)
plot (x, y)
x[1,] # 逗号后面需要留一个空格
x[1 ,] # 空格要留在逗号后面，而不是前面
```

1.2.2.2　大括号

- 左花括号后面应该新起一行。
- 右花括号后面独占一行，除非它后面跟着的是else语句。
- 可以在一行写非常短的语句。

```
# 好
if (y < 0 && debug) {
    message("Y is negative")
}
if (y < 0 && debug) message("Y is negative")
# 不好
if (y < 0 && debug)
message("Y is negative")
```

```
# 好
if (y == 0) {
  log(x)
} else {
 y ^ x
}

# 不好
if (y == 0) {
  log(x)
}
else {
   y ^ x
}
```

1.2.2.3 行的长度

尽量使每行代码不超过80个字符。

1.2.2.4 缩进

- 缩进代码时，使用两个空格。不要使用制表符或者把制表符和空格混合使用。
- 唯一的例外是一个函数定义分成了多行的情况。在这种情况下，第二行缩进到函数定义开始的位置

```
lo_fun_n <- function(a = "a long argument",
                     b = "another argument",
                     c = "another long argument")
```

1.2.2.5 赋值

尽量使用< −，而不要使用=进行赋值。

```
# 好
 x <- 5

#不好
x = 5
```

两者的区别<－=。多数情况下，两者可以互相替代，只有在函数参数列表中时不同.

```
plot(x <- 1:10)
```

就是在作图之前把1:10赋给x，这个表达式在绘图之前就被执行了，相当于

```
x<-1:10
plot(x)
```

```
plot(x = 1:10)
```

- 就不能达到对x赋值的目的，这时表达的含义是，为plot函数中名字为x的形式参数赋值1:10，作完图之后，x 这个对象仍然不存在。
- 只有在函数外部直接执行x=1:10才能达到对x赋值的目的。

```
> plot(x=1:10) #x是形式参数，不是新变量
> x ##错误: 找不到这个对象"x"
#<-不会用于形式参数，所以这是为新变量x赋值的语句，不会引起歧义
> plot(x<-1:10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

总之，<－在任何地方都可以完成赋值的工作，而=在函数参数列表则不行，所以，推荐任何时候都用<－进行赋值操作，而=仅用于形式参数?

1.2.3 结构

- 给你的代码添加注释。每一行注释都应该以一个注释符号和单个空格开头：♯。注释应该解释代码的原理，而不是每行代码做了什么。
- 使用带有-和=的注释线把你的文件分隔成容易阅读的块。

```
# 加载数据 ---------------------------
# 绘图 ---------------------------

# myfunction 开始----------
# myfunction 结束----------
```

1.3 对象

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. 对象的类塑和长度可以分别通过函数mode和length 得到.

对象的简单介绍（表1.1）：

- 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

表 1.1: 对象存储数据类型的区别

对象	类型	是否允许同一个对象中有多种类塑?
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

1.3.1　数据类型

数据类型(Data Type)，它是每个数据的所属的类型，基本的数据类型主要有四种：逻辑性(Logical)、整数型(Integer)、双精度型(Double)和字符型(Character)，其中整数型和双精度型合成数值型(Numeric)。还有两种不常见的类型：原型(Raw)和复数型(Complex)。

逻辑型只有两个取值，TURE(简称T)和FALSE(简称F)。

```
c(F,T)
```

整数型取值是整数，一个整数的长度有4-byte或者23-bit(bit就只0和1)，它可以表示为

$$\sum_{i=1}^{32} x_i 2^{i-1} - 2^{31}$$

其中$x_i \in \{0, 1\}$(注意的是+和−各占一个bite)。

```
#最大的整数
max_inter <- .Machine$integer.max
max_inter
#错误，因为已经是最大了
max_inter + 1L
```

双精度型取值为实数，一般可以表示为

$$(-1)^{x_0}\Big(\sum_{i=1}^{t} x_i 2^{-1} 2^k\Big)$$

其中$x_i \in \{0, 1\}$，k是一个整数，称为指数部分(Exponent)，x_0称为符号bit(Sign Bit)，$\sum_{i=1}^{t} x_i 2^{-1}$是significand。一个标准的双精度型可以有一个sign bit 、11 bit 和52 bits significand，组成。$2^{11} = 2048$，所以，Exponent可以从-1022取到1024。

```
  #最小的正双精度型取值
.Machine$double.eps
.Machine$double.base


.Machine$double.min.exp
.Machine$double.max.exp
```

字符型取值的是字符串。如

```
c("R","statistics")
```

原型是计算机识别的最直接的数据类型，它是以二进制形式保存。如

```
raw(2)
```

复数型取值是复数。如

```
3 + 5i
```

判断数据类型，可以使用mode和typeof(思考这两个命令的区别)。逻辑性(Logical)、整数型(Integer)、双精度型(Double)和字符型(Character)四种数据类型与运算符号一样，也有高低。我们可以用is.x看是否是x类型(如as.integer)，用as.x将数据强制转化为x类型。但这种转化不能将含有字母的字符串的转化为整数型。大多数数学函数(+,- log)会将数据转化为整数型或双精度型；大多数逻辑运算符(&,any)会将数据转化为逻辑型。运行下面代码，看看会出现什么样的结果。

```
x_int <- c(1L, 2L)
mode(x_int)
typeof(x_int)
x_dou <- c(1, 2)

x_log <- c(T, F)
as.integer(x_log)

x_cha <- c("R","S")
as.integer(x_cha)
x_cha1 <- c("1","2")
as.integer(x_cha1)

as.integer(x_dou)
```

1.3.2　向量

1.3.2.1　创建方式

向量(vextor)是常见的数据结构，创建它的方法常见有：

(i) c() 函数比较常见，将需要的数据拼接成向量。

```
#数值
> a <- c(1, 2, 5, 3, 6, -2, 4)
#字符
> b <- c("one", "two", "three")
#逻辑
> c <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE)
```

(ii) : : 产生的向量是数值的，产生的自然数，并且产生的自然数最小是1，前后两个数的间隔是1。

```
> #生产1-10个数
> 1:10#错误
> 1:10
> 5:10
```

(iii) seq 产生等差数列，一般用法是：

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL,
```

from: 开始的数值
to : 结束的数值
by : 步长， 如0.1 也就说前后两个数之间的差是0.1.
length.out : 总长度。 一般来说by和length.out 参数只需要设置一个。

```
> seq(1, 10, by=0.5)#在1-10之间产生间隔是0.5的等差数列
```

1.3.2.2 下标

下标是非常重要的重要，例如提取一个向量的第三个数据，寻找向量等于某一个数，或字符串的下表。

- 提取某一个下标用[index]函数其中*index* 是小标，可以是一个数，也可以是一个向量。
- 寻找某一个数的下标用which函数。

```
> x <- 100:110
> x[3]# 提取第三个下标的数值
> x[c(1, 3, 5)] # 提取第一个，第三个和第五个下标数值
> which(x==103) #寻找x中等于103的下标
```

1.3.2.3　添加、剔除和修改向量元素

- 添加：”,”添加
- 剔除：用[- index] 其中$index$ 是小标，可以是一个数，也可以是一个向量
- 修改用[index]

```
> x <- 1:10
> c(x, 100)#在向量x最后添加一个元素
> x[-3]# 剔除第三个下标的数值
> x[-c(1, 3, 5)] # 剔除第一个，第三个和第五个下标数值
> x[3] <- 1 #将第三个元素修改为1.
```

1.3.2.4　常见的函数

```
length:    向量的长度
all any:       判断向量的所有元素的性质
> x <- 1:10
> length(x) #x的长度
> any(x>3)  #判断x的元素是否有大于3，返回值TURE表示存在，FALSE表示不存在
> all(x>3)  #判断x的元素是否全部大于3
> x > 3     #半段x的每一个元素是否大于3
```

1.3.2.5　运算

- A”+”B：加
- A”-”B：减

- A"*"B：乘
- A"/"B：除

需要注意的时，如果A或B的长度不一致，那么长度短的将循环。

```
c(1, 2) + 1:10
2 * c(1, 2)
```

1.3.3 矩阵

dim rownames, colnames查看维度，行和列的变量名

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
> cells <- c(1,26,24,68)
> rnames <- c("R1", "R2")
> cnames <- c("C1", "C2")
> mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
                     dimnames=list(rnames, cnames))
> mymatrix
C1 C2
R1 1 26
R2 24 68
>mymatrix[,1]
```

1.3.4 数组

```
array(data = NA, dim = length(data), dimnames = NULL)
dim1 <- c("A1", "A2")
dim2 <- c("B1", "B2", "B3")
dim3 <- c("C1", "C2", "C3", "C4")
z <- array(1:24, c(2,3,4), dimnames=list(dim1,dim2,dim3))
z
, , C1
```

```
   B1 B2 B3
A1  1  3  5
A2  2  4  6
```

1.3.4.1　数据框

colnames ,rownames，cbind,rbind查看列，行变量名，合并列和行。

```
data.frame(col1, col2, col3,…)
```

```
> age <- c(25, 34, 28, 52)
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor", "Improved", "Excellent", "Poor")
> patientdata <- data.frame(age, diabetes, status)
> patientdata
  age diabetes    status
1  25    Type1      Poor
2  34    Type2  Improved
3  28    Type1 Excellent
4  52    Type1      Poor
```

1.3.5　因子

```
x <- factor(c("a", "b", "b", "a"))
x
#> [1] a b b a
#> Levels: a b
class(x)
#> [1] "factor"
levels(x)
#> [1] "a" "b"
# You can't use values that are not in the levels
x[2] <- "d"
```

```
#> Warning: invalid factor level, NA generated
x
#> [1] a <NA> b a
#> Levels: a b
```

怎样将添加因子水平d

1.4 读入数据

1.4.1 txt 和csv格式数据

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".",
           stringsAsFactors = default.stringsAsFactors())

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", )
```

sep separation 的简写，表示文件的分隔符。默认是空格，我们可以设置为逗号。

header 取值为T表示第一行为变量名

stringsAsFactors 默认TRUE将字符型数据转化为因子变量。

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

widths有时很有用。

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

```
C1.701.6
C1.751.7
> read.fwf("data.txt",  widths=c(1,  4,  3))
```

1.4.2　xlsx格式

xlsx数据，需要下载安装xlsx包，read.xlsx可以读入。

```
read.xlsx(file, sheetIndex, sheetName=NULL, rowIndex=NULL,
          startRow=NULL, endRow=NULL, colIndex=NULL)
```

file 指定的工作博，一种是数字形式指标第几个工作簿；另一种指定工作簿的名字

rowIndex，colIndex 指定读取行和列

startRow，endRow 没有指定rowIndex时，指定读入数据的行范围

1.4.3　SAS,SPSS Stata格式等

foreign包:

read.ssd SAS数据

read.spss SPSS数据

read.dta Stata数据

1.5　数据导出

```
write.table(x, file = "")
```

1.6　循环语句

1.6.1　for

for在R语言中比较常见，一般的用法

```
for( i in a:b){
  命令组
}
```

其中：

- i 是指针
- a:b是指针的范围。每循环一次，i便加1。指针范围也可以是一个向量。
- 命令组是需要执行的命令

下面，我们用for编写$1+2+\cdots+200$的代码。

```
> i_sum <- 0
> for(i in 1:200){
+   i_sum <- i_sum + i
+ }
> print(i_sum)
[1] 20100
```

1.6.2 while

```
while(循环条件){
  命令组
}
```

循环条件是一个判断语句，R会不断执行，知道循环条件为FALSE

下面，我们用while编写编写$1+2+\cdots+200$的代码。

```
> i_sum <- 0
> i <- 1
> while(i <= 200){
+   i_sum <- i_sum + i
+   i <- i + 1
+ }
> print(i_sum)
[1] 20100
```

1.6.3　if类

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

简单的例子：

```
i  <- 5
if(i == 1){
  5
}else if(i==2){
  5
}else{
  100
}
```

1.7　自编函数

我们可以是用function命令编写自己的函数，格式如下

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

比如我们要编写一个函数来计算自然数的阶乘:

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

```
  return(rlt)
}

ft(3)
```

1.8 画图

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)
hist(x)　x的频率直方图
barplot(x) x的值的条形图
qqnorm(x) 正态分位数一分位数图
qqplot(x, y) y对x的分位数一分位数图

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:

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，
xlim=, ylim=　指定轴的上下限，
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)

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=I1nI1); text(x, y, names)

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

legend(x, y, legend) 在点(x, y)处添加图例，说明内容由legend给定.

第2章　高性能编程

这一章，我们主要参考Aloysius 和AWilliam 写的高性能编程[1]。

2.1　R慢的原因

- 计算性能的三个限制因素：CPU、RAM 和磁盘I/O (输入和输出)
  - CPU 决定了命令执行的速度，它将代码翻译成机器代码，以及机器代码实际处理数据的执行过程
  - RAM 决定处理数据的规模
  - I/O速度影响数据载入到内存的速度或数据存回磁盘的数据。

R运行下面代码，具体的过程是（图2.1）：

```
data <- read.csv("mydata.csv")
totals <- colSums(data)
write.csv(totals, "totals.csv")
```

1. 将代码装入内存(RAM)
2. R解释器将代码翻译成机器代码，并将机器代码装载到CPU
3. CPU执行
4. 程序将处理数据从硬盘装载到内存(`read.csv`)
5. 数据以小型数据形式转载到CPU处理
6. CPU一次处理一个数据块，全部处理完毕后(由于`colSums`处理列和)之后，交换数据块到内存
7. 处理后的数据存回磁盘(`write.csv`)

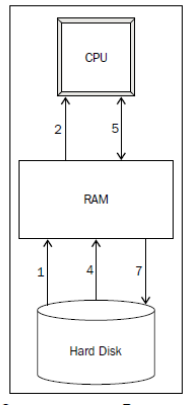

图 2.1: R运行代码过程

解释型语言(Interpreted Language)和编译型语言(Complied Languag)的区别。

- R是一种动态的、交互式的解释型语言。将R语句输入R 控制体内(Console)，就会有结果。因为，R会即时解析和执行。但不足时运行相对较慢，因为，每一次运行都需要重新解释，即使代码不变化。
- 与R对照的是编译型语言，它在执行前已经将源代码编译成机器代码。所以交互性不好，因为大型程序的编译需要几分钟的时间，即使代码没有变化。然而一旦编译，在CPU 运行很快，因为它已经是计算机本机语言(Computer's native language)。
- 解释性语言CPU需要做两件事：解释代码和执行代码。相对编译型语言，多做了解释代码。

2.2 性能分析

分析自编函数的运行时间

```
> system.time(runif(1e8))
用户 系统 流逝
3.37 0.22 3.61
```

[1]Aloysius L., William T. R High Performance Programming, Packt Publishing, 2015

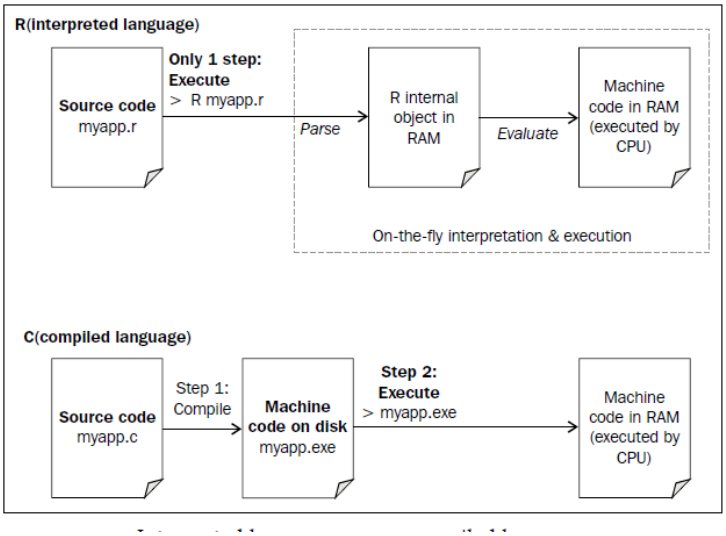

图 2.2: 解释性语言和编译语言

用户时间(User time) 给定一个表达式，执行指令所需要的CPU时间。这个时间不包含其它进程的CPU时间。如，计算机后台杀毒，这个时间不包括在内

系统时间(System time) 给定一个表达式，如打开或关闭文件或者分配和释放内存，执行指令所需的CPU时间。同样，这个时间不包含其它进程的CPU时间。

流逝时间(Elsped time) 给定表达式，实际执行指令所需要的总时间。这个时间包括其他进程的时间。有时，流逝> 用户+系统，因为CPU处理多个任务。

有时需要，反复运行代码，计算平均值。`rbenchmark`包中的`benchmark`可以实现。

```
install.packages("rbenchmark")
library(rbenchmark)
bench1 <- benchmark(runif(1e8), replications=10)
bench1
> bench1
```

```
          test replications elapsed relative user.self sys.self user.child sys.child
1 runif(1e+08)           10   37.83        1     35.44     2.36         NA        NA
```

流逝时间(Elsped time) 运行10次的总时间

microbenchmark包中的microbenchmark可以实现运行时间的分布。

```
install.packages("microbenchmark")
library(microbenchmark)
microbenchmark(runif(1e8), times=10)
> microbenchmark(runif(1e8), times=10)
Unit: seconds
         expr      min       lq     mean   median
 runif(1e+08) 3.445197 3.464214 3.503712 3.496884
  uq      max       neval
  3.515112 3.651712    10
```

分别是最小值，下四分位数、均值，中位数，上四分位数，最大值

如果函数比较复杂，包含多个部分，得到每一部分的运行时间，从而改进编写函数

```
sampvar <- function(x) {
  # 向量x的和
  my.sum <- function(x) {
    sum <- 0
    for (i in x) {
      sum <- sum + i
    }
    sum
  }
  # 向量x的均值
  sq.var <- function(x, mu) {
    sum <- 0
```

```
    for (i in x) {
      sum <- sum + (i - mu) ^ 2
    }
    sum
  }
  mu <- my.sum(x) / length(x)
  sq <- sq.var(x, mu)
  sq / (length(x) - 1)
}

x <- runif(1e7)
#通知R开始性能分析,
#Rprof.out是存储性能文件名, 可以设置路径
Rprof("Rprof.out")
y <- sampvar(x)
#通知R停止性能分析
Rprof(NULL)
summaryRprof("Rprof.out")
```

```
> summaryRprof("Rprof.out")
$by.self
         self.time self.pct total.time total.pct
"sq.var"      7.30    59.16       8.46     68.56
"my.sum"      3.64    29.50       3.88     31.44
"^"           0.60     4.86       0.60      4.86
"+"           0.40     3.24       0.40      3.24
"-"           0.32     2.59       0.32      2.59
"("           0.08     0.65       0.08      0.65
$by.total
          total.time total.pct self.time self.pct
"sampvar"      12.34    100.00      0.00     0.00
"sq.var"        8.46     68.56      7.30    59.16
"my.sum"        3.88     31.44      3.64    29.50
```

```
"^"             0.60      4.86      0.60     4.86
"+"             0.40      3.24      0.40     3.24
"-"             0.32      2.59      0.32     2.59
"("             0.08      0.65      0.08     0.65
$sample.interval
[1] 0.02
$sampling.time
[1] 12.34
```

self.time，self.pct 每个函数运行的时间，不包含函数中条用其他函数函数的运行时间

total.time，total.pct 每个函数运行的时间，包含函数中条用其他函数函数的运行时间

结果分析 "sampvar" 运行时间是，12.34，其中"sq.var" 运行时间达到8.46

设置memory.profiling参数，可以看到内存分配情况。

```
Rprof("Rprof-mem.out", memory.profiling=TRUE)
y <- sampvar(x)
Rprof(NULL)
summaryRprof("Rprof-mem.out", memory="both")
```

```
> summaryRprof("Rprof-mem.out", memory="both")
$by.self
         self.time self.pct total.time total.pct mem.total
"sq.var"      7.42    60.42       8.44     68.73    2519.5
"my.sum"      3.38    27.52       3.84     31.27    1103.3
"+"           0.56     4.56       0.56      4.56     197.9
"^"           0.50     4.07       0.50      4.07     122.8
"-"           0.42     3.42       0.42      3.42     141.6
$by.total
          total.time total.pct mem.total self.time self.pct
```

```
"sampvar"      12.28    100.00    3622.8      0.00      0.00
"sq.var"        8.44     68.73    2519.5      7.42     60.42
"my.sum"        3.84     31.27    1103.3      3.38     27.52
"+"             0.56      4.56     197.9      0.56      4.56
"^"             0.50      4.07     122.8      0.50      4.07
"-"             0.42      3.42     141.6      0.42      3.42
$sample.interval
[1] 0.02
$sampling.time
[1] 12.28
```

- sampvar需要3622.8MB内存，sq.var 需要的内存是2519.5MB。
- mem.total是一个误导指标。因为计算正在运行函数的内存使用量，而此时其它函数可能还在使用这些内存尚未释放。R的垃圾收集器(garbage collector)会将不再使用的内存定期释放辉系统。
- 运行sampvar时，垃圾收集器一共被激活272次。虽然Rprof报告的分配3622.8MB内存，但垃圾收集器努力释放为使用的内存，使得R的实际总内存消耗是109.1MB(26.3+82.8)左右。

```
> gcinfo(TRUE)
[1] FALSE
> y <- sampvar(x)
Garbage collection 541 = 497+29+15 (level 0) ...
26.3 Mbytes of cons cells used (52%)
82.8 Mbytes of vectors used (7%)
> gcinfo(FALSE)
```

2.3　加快R运行的方法

2.3.1　向量化

R擅长矩阵的运算，我们应该熟练矩阵的各种运行，例如点乘等。下面以一个简单的例子：计算一个数据每一个元素的平方。

```
> N <- 1E6
> data <- sample(1:30, size=N, replace=T)
> system.time({
+   data_sq1 <- numeric(N)
+   for(j in 1:N) {
+     data_sq1[j] <- data[j]^2
+   }
+ })
用户 系统 流逝
1.53 0.00 1.56
>
> system.time(data_sq2 <- data^2)
用户 系统 流逝
0.02 0.00 0.02
```

可以看出，向量化的速度快很多。

无论向量规模怎么变化，非向量花的计算时间大约是向量化的200倍左右（表2.1）。

表 2.1: 向量化和非向量化耗时的比较

向量规模	100,000	1,000,000	10,000,000	100,000,000
非向量化	120 ms	1.19 s	11.9 s	117 s
向量化	508 µs	5.67 ms	52.5 ms	583 ms

2.3.1.1 adaplasso截断稀疏协方差估计

协方差矩阵是统计学研究的重要问题之一，例如主成分分析。假设$\boldsymbol{X} = (X_1, \ldots, X_p)^\top$是$p$维随机向量，并且其协方差矩阵是$\Sigma = (\sigma_{ij})$。给定独立同分布样本$\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}$，则样本的协方差矩阵是

$$\Sigma_n = (\hat{\sigma}_{ij}) = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{X}_k - \overline{\boldsymbol{X}})(\boldsymbol{X}_k - \overline{\boldsymbol{X}})^\top$$

当$p > n$，上面的估计不是好的估计。Bickel和Levina (2008)[2]提出截断估计(thresholding estimator)

$$T_\delta(\Sigma) = (\hat{\sigma}_{ij} I(|\hat{\sigma_{ij}}| \geq \delta))$$

其中$I(A)$是示性函数，A成立，其值为1，否则为0. δ是某一个常数。我们可以使用交叉核实(Cross Validation, CV), 将样本随机分为K份，计算

$$CV(\delta) = \frac{1}{K} \sum_{k=1}^{K} \|T_\delta(\Sigma)_{-k} - \Sigma_k\|_F^2$$

其中Σ_k是第k部分样本的协方差，Σ_{-k} 是去掉k部分的样本的剩下样本的样本协方差矩阵。由于所有的元素，阈值δ是一样，这显然不合理。每一个元素的阈值应该根据自身的不同而不同。自然想起每一个元素的方差的信息。Cai和Liu (2011)[3]提出adaplasso截断估计，

$$\hat{\Sigma}^*(\delta) = (\hat{\sigma}_{ij}^*), \ \ \hat{\sigma}_{ij}^* = s_{\lambda_{ij}}(\hat{\sigma}_{ij})$$

其中

$$\lambda_{ij} = \lambda_{ij}(\delta) = \delta \sqrt{\frac{\hat{\theta}_{ij} \log p}{n}}$$

$\theta_{ij} = Var[(X_i - \mu_i)(X_j - \mu_j)]$是$\sigma_{ij}$方差，其估计值

$$\hat{\theta}_{ij} = \frac{1}{n} \Big[(X_{ki} - \overline{X}^i)(X_{kj} - \overline{X}^j) \Big], \overline{X}^i = \frac{1}{n} \sum_{k=1}^{n} X_{ki}$$

其中δ可以使用CV确定，这里我们不去选择它，而是令其等于2.

下面我们实现Cai和Liu (2011)文章中的一个模拟例子。$\Sigma = diag(A_1, A_2)$，其中$A_1 = (\sigma_{ij})_{1 \leq i,j \leq p/2}$, $\sigma_{ij} = (1 - \frac{|i-j|}{10})_+$ $a_+ = \max\{a, 0\}$表示a的正部。$A_2 = 4I_{p/2 \times p/2}$。$\boldsymbol{X}$来自于均值为0，协方差矩阵是$\Sigma$的多元正态分布。我们使用的adaplasso函数是$s_\lambda(z) = z(1 - |\lambda/z|^4)_+$我们设置$n = 100$,$p = 30, 100,$ 使用损失矩阵(估计值减去真实值)的Operator, l_1 和Frobeninus 矩阵范数(norm) 评

[2]Bickel, P., and Levina, E. (2008), "Covariance Regularization by Thresholding," The Annals of Statistics, 36, 2577 - 2604

[3]Cai, T.,Liu, W.D. and Luo, X. (2011), A constrained L1 minimization approachto sparse precision matrix estimation. Journal of the American Statistical Association. 106: 594-607.

价效果。重复模拟100 次，计算损失矩阵的均值和标准误(Standard Error)Cai和Liu (2011) 的结果是

表 2.2: 损失矩阵的均值和标准误

	$p = 30$	$p = 100$
Operator norm	1.72 (0.05)	2.72 (0.05)
l_1 norm	2.57 (0.08)	4.15 (0.07)
Frobenius norm	3.15 (0.05)	6.57 (0.05)

```
library(MASS)
dat4 <- function(n, p){
  p1 <- p / 2
  ind <- 1: p1
  A1 <- pmax(1 - abs(outer(ind, ind, "-"))/ 10, 0)
  A2 <- 4 * diag(p1)
  A <- matrix(0, p, p)
  A[1:p1, 1:p1] <- A1
  A[(p1+1):p, (p1+1):p] <- A2
  mux <- rep(0, p)
  x <- MASS::mvrnorm(n=n, mu=mux, Sigma=A)
  return(list(A=A, x=x))
}


adacov <- function(x, delt=2){
  p <- dim(x)[2]
  n <- dim(x)[1]
  # estimate A
  hatA <-  cov(x)
  # adaptive estimate A begin
  # theta begin
```

```
    x.s <- scale(x, center = TRUE, scale = FALSE)
    b1 <- t(x.s ^ 2) %*% x.s ^ 2 / n
    b2 <- t(x.s) %*% x.s * hatA / n
    # theta end
    hattheta <- b1 - 2* b2 +  hatA ^2
    lam <- delt * sqrt( hattheta * log(p) / n )
    adasig <- hatA * pmax( 1 - abs( lam / hatA) ^ 4 , 0)## adaptive lasso
    diag(adasig) <-  diag(hatA)
    # adaptive estimate A end
    list(lam=lam, hatA = hatA, adasig=adasig)
  }
```

2.3.2　使用内置函数

计算数据的矩阵行截断均值。

- apply
- rowMeans

```
n <- 400000
p <- 20
X <- matrix(rnorm(n * p), n, p)

system.time(rowMeans(X, na.rm = 0.1))
system.time(apply(X, 1, mean, na.rm = 0.1))
```

```
> system.time(rowMeans(X, na.rm = 0.1))
用户 系统 流逝
0.06 0.00 0.06
> system.time(apply(X, 1, mean, na.rm = 0.1))
用户 系统 流逝
3.11 0.00 3.12
```

2.3.3 预分配内存

- 大部分Strongly typed programming，比如C,C++等，通常生声明向量，然后才执行。
  - 预先分配向量所需的内存空间
  - 动态分配内存
- R作为一个解释性语言，控制性比较小，用户易忽略这个问题。如果向量没有预先分配内存，R并不会报错，但是执行时间会明显增强。

```
> N <- 1E4
> data_series1 <- 1
> system.time({
+   for (j in 2:N) {
+     data_series1 <- c(data_series1,
+                       data_series1[j-1]+sample(-5:5, size=1))
+   }
+ })
用户 系统 流逝
0.11 0.11 0.22
>
> data_series2 <- numeric(N)
> data_series2[1] <- 1
> system.time({
+   for (j in 2:N) {
+     data_series2[j] <- data_series2[j-1]+sample(-5:5, size=1)
+   }
+ })
用户 系统 流逝
0.05 0.00 0.05
```

- 预分配运行时间是动态分配的10倍。

- apply，lapply，sapply 自动处理内存预分配。

表 2.3: 向量大小需要的时间(秒)

向量大小	10	100	1000	10,000
动态分配	0	0.006	0.288	25.373
预分配	0.001	0.006	0.062	0.577

2.3.4　使用更简单的数据结构

- data.frame是数据分析的主要结构，也是做常见的数据结构
- 它可以存储不同类别的数据，但如果数据类别都相同，最好使用矩阵数据结构
- data.frame进行矩阵运算比matrix慢
  - data.frame 转换成matrix

```
> data <- rnorm(1E4*1000)
> dim(data) <- c(1E4,1000)
> system.time(data_rs1 <- rowSums(data))
用户 系统 流逝
0.02 0.00 0.02
>
> data_df <- data.frame(data)
> system.time(data_rs2 <- rowSums(data_df))
用户 系统 流逝
0.04 0.02 0.06
```

第3章　并行计算

谈到并行计算似乎很高大上。因为它似乎是计算机专业的用语。其实不然，它不过是提高计算速度的方法。我们进行统计模拟时，为了比较不同方法的优劣，通常会重复实验几百次。遇到这样的问题，我们应该怎么办？

- 检查代码，将与循环无关的代码或者变量全部移除，这样可以有效减少重复计算。
- 尽量不要重复赋值，这样会占用存储空间，也加重了内存的负担。此时，我们可以清除不需要的对象。
- 并行计算

R是一个单线程的程序。比如电脑是16个核(Core)，而R只使用了一个核。这是不是找到R慢的原因了。一个任务交给一个核，它肯定没有交给几个核快。但不是所有的任务都可以交给几个核。一般，可以多次独立的任务可以并行。这有点抽象。我们可以简单理解只要有循环就可以有并行。这虽然有点侠义，但至少知道什么时候用并行。下面我们介绍R的包。

3.1　R的实现

3.1.1　parallel

lapply类运行采用了并行计算，这也是通常建议使用的原因。在parallel包里，parLapply 和mclapply类函数可以实现并行化，但后面函数不能在windows使用。

```
parLapply(cl = NULL, X, fun, ...)
```

其中

其中

cl: 使用的计算机核的数目
X: 需要计算参数
fun: 函数

需要注意的，这里的参数只有一个，不能是两个。

下面，我们介绍怎么使用。

R 代码

```
##编写测试函数
testfun <- function(x){
  return (abs(x));
}

##rnorm(10^7)随机产生正态分布的随机数
##没有用并行计算
system.time({
  absx <- lapply(rnorm(10^7), testfun)
})

#加载parallel包
library(parallel)

#detectCores函数可以告诉你你的CPU可使用的核数
clnum<-detectCores()

#设置参与并行的CPU核数目
cl <- makeCluster(getOption("cl.cores", clnum));

#运行
system.time({
  absx1 <- parLapply(cl, rnorm(10^7),  testfun)
```

```
})

#关闭并行计算
stopCluster(cl)
```

输出结果

```
> ##没有用并行计算
> system.time({
+   absx <- lapply(rnorm(10^7), testfun)
+ })
 用户  系统  流逝
20.22  0.18 20.51

> system.time({
+   absx1 <- parLapply(cl, rnorm(10^7),  testfun)
+ })
 用户  系统  流逝
 9.33  0.89 19.25
> stopCluster(cl)
>
```

从结果来看，利用parLapply 计算快很多。由于lapply本身已经利用了并行计算，结果没有那么明显。感兴趣的读者可以试试for循环的需要时间。

3.1.2 foreach

foreach包比parallel包灵活一些。

```
foreach(..., .combine, .init, .final=NULL, .inorder=TRUE,
        .multicombine=FALSE,
        .maxcombine=if (.multicombine) 100 else 2,
        .errorhandling=c('stop', 'remove', 'pass'),
        .packages=NULL, .export=NULL, .noexport=NULL,
        .verbose=FALSE)
when(cond)
e1 %:% e2
obj %do% ex
obj %dopar% ex
times(n)
```

其中

.combine: 结果整合方式 可以使用c, rbind, cbind,list等。默认使用list

.packages: 加载R包，如.packages="car"，使用car包中的函数。如果加载多个包，可以用c连接。如.packages=c("car","quantreg").

比如：

```
> library(foreach)
> library(doParallel)
> cl <- makeCluster(8)#使用8个核
> registerDoParallel(cl)
> # 并行计算方式
> absx2 <- foreach(x=rnorm(10^3), .combine=c)  %dopar%  abs(x)
> #别忘了结束并行
> stopCluster(cl)
```

3.2　应用

3.2.1　分位数变量选择

下面，以分位数变量选择为例，阐述并行计算的运用。Wu和Liu(2008)提出分位数变量选择，求下面目标函数的最小值。

$$\sum_{i=1}^{n}\rho(y_i-\boldsymbol{x}_i^{\top}\boldsymbol{\beta})+\sum_{j=1}^{p}p_{\lambda_n}(\beta_j)$$

其中λ_n是截断参数，我们可以$HBIC$(Peng 和Wang, 2015)进行选择.

$$HBIC=\log\Big(\sum_{i=1}^{n}\rho(y_i-\boldsymbol{x}_i^{\top}\boldsymbol{\beta})\Big)+s_{\lambda_n}\frac{\log(\log n)}{n}\log p$$

其中s_{λ_n}是估计参数非零的个数。模拟中，我们惩罚函数是SCAD (Fan和Li, 2001)。

R 代码

```
library(quantreg)#加载包
n <- 60
p <- 7
rho <- .5
beta <- c(3,1.5,0,2,0,0,0)
R <- matrix(0,p,p)
###for 这个可以用outer简化
for(i in 1:p){
  for(j in 1:p){
    R[i,j] <- rho^abs(i-j)
  }
}
set.seed(1234)
x <- matrix(rnorm(n*p),n,p) %*% t(chol(R))
y <- x %*% beta + rnorm(n)
```

```
HBIClam <- function(y=y, x=x,lam=lam){
  n <- dim(x)[1]
  p <- dim(x)[2]
  fit <- quantreg::rq(y ~ x, method="scad",lambda = lam)
  slam <- sum(abs(fit$coefficients) <= 10^(-5))
  HBIC <- log(fit$rho) + slam * log(log(n))/n * log(p)
  return(HBIC)
}

HBIClam(y=y, x=x,lam=0.5)

library(foreach)
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
lamv <- seq(0.01, 3, by=0.1)
# 并行计算方式
HBC0 <- foreach(lam=lamv, .combine=c, .packages = "quantreg")  %dopar%  {
   HBIClam(y=y, x=x,lam=lam)
}

#别忘了结束并行
stopCluster(cl)

index <- min(which(HBC0==min(HBC0)))
lam.opt <- lamv[index]
rq(y ~ x, method="scad",lambda = lam.opt)$coefficients
```

输出结果

```
> rq(y ~ x, method="scad",lambda = lam.opt)$coefficients
```

```
(Intercept)          x1          x2          x3          x4
-0.04657366  3.16378978  1.19977584  0.06871417  2.28364354
         x5          x6          x7
 0.27274699 -0.06088954  -0.12745774
```

3.2.2 Box-Cox变换

这个例子比较复杂。

3.3 参考文献

1. Wickham, H. (2014). Advanced R. CRC Press.

2. Wu, Yichao, and Yufeng Liu. "Variable selection in quantile regression." Statistica Sinica (2009): 801-817.

3. Peng, Bo, and Lan Wang. "An iterative coordinate descent algorithm for high-dimensional nonconvex penalized quantile regression." Journal of Computational and Graphical Statistics 24.3 (2015): 676-694.

4. Fan, Jianqing, and Runze Li. "Variable selection via nonconcave penalized likelihood and its oracle properties." Journal of the American statistical Association 96.456 (2001): 1348-1360.

第4章　常见函数的用法

这一部分，我们将介绍常见的函数。有些函数也许不常见，但这些都是我遇到的一些问题的积累。R中有很多函数，这些函数吸引了很多的人。

4.1　运算函数

4.2　某些函数

4.2.1　set.seed

set.seed在随机模拟中经常用到。有时候我们进行实现时不希望产生的随机数发生变化。比如：我们计算100个正态分布的随机数的均值，我们希望下次运行时，这个均值不发生变化。那我们的命令是：

```
set.seed(10.00)#10可以更换100等整数。
mean(rnorm(10))
```

我们可以拷贝这个命令运行两个看看，是不是均值没有变化。你运行一次，然后关闭R，在打开运行一次看看结果。

第5章　黑色魔法

之所以成为黑色魔法，因为这些知识不常见，不易掌握，其实难度没有想象的那么大。但掌握它们对我们进行复杂计算有帮助。

5.1　管道函数

管道(Piper)函数将物质传递一的函数。下面介绍magrittr包的四个管道函数：%>%，%T>%，%$% 和%<>% [1] 。

[1] http://blog.fens.me/r-magrittr/

第6章　爬数据

这一章，我们将介绍怎么读取网页的数据。R有很多包可以实现，其中XML和rvest。其中后面一个包是大神Hadley Wickham的杰作。

6.1　简单的例子

我们下面介绍怎么爬北京PM2.5的数据，http://www.86pm25.com/city/beijing.html.

R 代码

```
rm(list = ls())
library(rvest)
#读入网址数据
web <- read_html("http://www.86pm25.com/city/beijing.html")
#提取网址第一个表格内容
dat <- html_table(html_nodes(web, "table")[[1]])
class(dat)

## %>% 管道函数，左件的值发送给右件的表达式，并作为右件表达式函数的第一个参数
#html_nodes 提取网址的node
#html_text  提取网址的attributes, text 和tag名字
time1 <- web %>% html_nodes(".remark") %>% html_text()
##substr截取字符串
time2 <- substr(time1, 4, nchar(time1))
##转化为时间格式
time <- strptime(time2,"%Y年%m月%d日%H时")

pm25 <- cbind(time, dat)
pm25
```

```
#将数据保存在D盘pm25.csv文件中
write.csv(pm25, file="d:/pm25.csv")
```

输出结果

```
> pm25
                  time     区／县     监测站点 AQI 污染等级 PM2.5浓
度 PM10浓度
1  2017-11-19 15:00:00    海淀区       官园  88       NA   65μg/m3 101μg/m3
2  2017-11-19 15:00:00    海淀区 海淀区万柳  99       NA   74μg/m3 118μg/m3
3  2017-11-19 15:00:00    海淀区   北部新区  96       NA   72μg/m3   -μg/m3
4  2017-11-19 15:00:00    海淀区     植物园  98       NA   73μg/m3  84μg/m3
5  2017-11-19 15:00:00    海淀区   西直门北 110       NA   83μg/m3  91μg/m3
6  2017-11-19 15:00:00    昌平区     昌平镇  73       NA   53μg/m3  70μg/m3
7  2017-11-19 15:00:00    昌平区       定陵  64       NA   46μg/m3  55μg/m3
8  2017-11-19 15:00:00    朝阳区     东四环 106       NA   80μg/m3 143μg/m3
9  2017-11-19 15:00:00    朝阳区 美国大使馆 112       NA   85μg/m3     μg/m3
10 2017-11-19 15:00:00    朝阳区   奥体中心  95       NA   71μg/m3  84μg/m3
11 2017-11-19 15:00:00    朝阳区     农展馆 109       NA   82μg/m3 112μg/m3
12 2017-11-19 15:00:00    大兴区       榆垡 106       NA   80μg/m3   -μg/m3
13 2017-11-19 15:00:00    大兴区       大兴  93       NA   69μg/m3 104μg/m3
14 2017-11-19 15:00:00    大兴区       亦庄 120       NA   91μg/m3 108μg/m3
15 2017-11-19 15:00:00    东城区       东四 102       NA   76μg/m3 104μg/m3
16 2017-11-19 15:00:00    东城区       天坛 104       NA   78μg/m3 106μg/m3
17 2017-11-19 15:00:00    东城区   永定门内 106       NA   80μg/m3  93μg/m3
18 2017-11-19 15:00:00    房山区       房山  88       NA   65μg/m3  75μg/m3
19 2017-11-19 15:00:00    房山区     琉璃河  86       NA   64μg/m3  82μg/m3
20 2017-11-19 15:00:00    丰台区   万寿西宫 107       NA   80μg/m3  97μg/m3
21 2017-11-19 15:00:00    丰台区   丰台花园  96       NA   72μg/m3 120μg/m3
```

22	2017-11-19 15:00:00	丰台区	云岗	110	NA	83 μg/m3	99 μg/
23	2017-11-19 15:00:00	丰台区	南三环	97	NA	70 μg/m3	143 μg/
24	2017-11-19 15:00:00	怀柔区	怀柔镇	132	NA	100 μg/m3	113 μg/
25	2017-11-19 15:00:00	门头沟区	门头沟	106	NA	80 μg/m3	113 μg/
26	2017-11-19 15:00:00	密云县	密云	108	NA	81 μg/m3	89 μg/
27	2017-11-19 15:00:00	密云县	密云水库	58	NA	41 μg/m3	57 μg/
28	2017-11-19 15:00:00	平谷区	平谷	111	NA	84 μg/m3	91 μg/
29	2017-11-19 15:00:00	平谷区	东高村	136	NA	104 μg/m3	125 μg/
30	2017-11-19 15:00:00	石景山区	古城	123	NA	93 μg/m3	104 μg/
31	2017-11-19 15:00:00	顺义区	顺义新城	168	NA	127 μg/m3	134 μg/
32	2017-11-19 15:00:00	通州区	永乐店	170	NA	129 μg/m3	141 μg/
33	2017-11-19 15:00:00	通州区	通州	157	NA	120 μg/m3	159 μg/
34	2017-11-19 15:00:00	西城区	前门	106	NA	80 μg/m3	92 μg/
35	2017-11-19 15:00:00	延庆县	延庆	58	NA	27 μg/m3	66 μg/
36	2017-11-19 15:00:00	延庆县	八达岭	56	NA	33 μg/m3	62 μg/

6.1.1　动态读入PM2.5数据

```
rm(list = ls())
library(rvest)
task <- function() {
  Sys.sleep(60)
  web <- read_html("http://www.86pm25.com/city/beijing.html")
  dat <- html_table(html_nodes(web, "table")[[1]])
  time1 <- web %>% html_nodes(".remark") %>% html_text()
  time <- substr(time1, 4, nchar(time1))
  pm25 <- cbind(time, dat)
  write.table(pm25, file="d:/pm25.txt", append=TRUE, col.names = F)
  print(Sys.time( ))
}
```

```
library(tcltk2)
t1 <- tclTaskSchedule(wait = 5, expr = task(), redo = TRUE)

##停止的命令
tclTaskDelete(id = t1$id)
```

6.1.2 read_html函数

第二部分

矩阵计算

第7章　广义逆运算

7.1　引言

矩阵的逆运算在统计学中非常常见，OLS求解，变系数模型求解等。它主要包括逆和广义逆。

7.2　广义逆

定义7.2.1 (广义逆,Generalized Inverse)　设$\mathbf{A} \in \mathbb{R}^{n\times m}$，如果$\mathbf{G} \in \mathbb{R}^{m\times n}$满足：

(1) $\mathbf{AGA} = \mathbf{A}$

(2) $\mathbf{GAG} = \mathbf{G}$

(3) $(\mathbf{GA})^\top = \mathbf{GA}$

(4) $(\mathbf{AG})^\top = \mathbf{AG}$

则称$\mathbf{G}$是$\mathbf{A}$的*Penrose*广义逆，并且记为$\mathbf{A}^-$。

若$\mathbf{G}$只满足条件(1)，则$\mathbf{G}$为$\mathbf{A}$的{1}-逆，记为$\mathbf{G} \in \mathbf{A}\{1\}$。若$\mathbf{G}$只满足条件(1)和(2), 则$\mathbf{G}$为$\mathbf{A}$的{1,2}-逆，记为$\mathbf{G} \in \mathbf{A}\{1,2\}$。满足条件(1), (2), (3), (4)的部分或全部的广义逆矩阵共有15 类。常用的广义逆有5 类：$\mathbf{A}\{1\}$、$\mathbf{A}\{1,2\}$、$\mathbf{A}\{1,3\}$、$\mathbf{A}\{1,4\}$和$\mathbf{A}^-$。只有$\mathbf{A}^-$ 是唯一的，而其他各种广义逆矩阵都不是唯一的。当$\mathbf{A}$可逆时, 它的所有广义逆矩阵都等于$\mathbf{A}^{-1}$。

7.3　R的实现

广义逆可以用R 包`limSolve` (Soetaert等2014)中`Solve`实现。我们随机产生矩阵，验证定义中的(1)-(4)。从输出结果来看，`Solve` 的结果满足定义的要求。

R 代码

```
library(limSolve)
A <- matrix(rnorm(16),4,4)#随机产生矩阵
A[,4] <- A[,1] + A[,2] #第4列是第1列和第2列相加之和
solve(A)#R自带的求矩阵逆，不能运行
G <- Solve(A)
A %*% G %*% A - A #验证定义(1)
G %*% A %*% G - G #验证定义(2)
t(G %*% A) -  G%*% A #验证定义(3)
t(A %*% G) -  A%*% G #验证定义(4)
```

输出结果

```
> solve(A)#R自带的求矩阵逆，不能运行
Error in solve.default(A) :
  system is computationally singular:
  reciprocal condition number = 1.04046e-17
> A %*% G %*% A - A
              [,1]          [,2]          [,3]          [,4]
[1,]  0.000000e+00 2.220446e-16 0.000000e+00 4.440892e-16
[2,]  4.440892e-16 4.996004e-16 1.110223e-16 8.881784e-16
[3,]  0.000000e+00 4.440892e-16 2.220446e-16 6.661338e-16
[4,] -2.220446e-16 4.440892e-16 2.220446e-16 1.110223e-16
> G %*% A %*% G - G
              [,1]          [,2]          [,3]          [,4]
[1,]  1.110223e-16  7.632783e-17 1.110223e-16 -5.551115e-17
[2,]  1.110223e-16 -2.775558e-17 0.000000e+00  9.367507e-17
[3,] -2.220446e-16  0.000000e+00 0.000000e+00 -1.110223e-16
[4,]  8.326673e-17  4.163336e-17 1.734723e-17  2.081668e-17
> t(G %*% A) -  G%*% A
              [,1]          [,2]          [,3]          [,4]
[1,]  0.000000e+00  5.551115e-17 -1.294944e-16 -5.551115e-17
```

```
[2,] -5.551115e-17  0.000000e+00  5.856725e-16  1.110223e-16
[3,]  1.294944e-16 -5.856725e-16  0.000000e+00 -3.911530e-16
[4,]  5.551115e-17 -1.110223e-16  3.911530e-16  0.000000e+00
> t(A %*% G) -  A%*% G
              [,1]          [,2]          [,3]          [,4]
[1,]  0.000000e+00 -4.163336e-16 -1.387779e-16 5.551115e-17
[2,]  4.163336e-16  0.000000e+00  1.110223e-16 0.000000e+00
[3,]  1.387779e-16 -1.110223e-16  0.000000e+00 2.775558e-17
[4,] -5.551115e-17  0.000000e+00 -2.775558e-17 0.000000e+00
```

7.4　应用

7.4.1　变系数模型

变系数模型(Varying Coefficient Model)是线性模型的推广，最早由Shumway (1988)提出，Hastie和Tibshirani (1993) 进行了详细的研究，其拓展了局部回归技术从一维到高维的应用(Fan和Zhang 2008)。与经典的线性模型相比，变系数模型具有以下优点：（1）不需要假设误差项服从某个分布，而经典线性模型需要误差项来自于正态分布；（2）它不像经典线性回归那样，假设自变量与因变量成线性关系，它只假设自变量对因变量有影响，但是这种影响不再是固定的和线性的，它是随着某个指示变量（如时间）而变化，并且这种变化是未知的。可见，变系数模型放宽了模型的假设，更加自由和灵活。

令$\{(Y_i, \boldsymbol{X}_i, U_i), i = 1, 2, \ldots, n\}$是来自$\{Y, \boldsymbol{X}, U\}$的观测值，其中$Y_i \in R$ 是因变量, $\boldsymbol{X}_i = (X_{i1}, X_{i2}, \ldots, X_{ip})^\top \in \boldsymbol{R}^p$ 是 p 维自变量。如果需要可以假定$X_{i1} \equiv 1$。$U_i \in R$ 是一维的指示变量(Univariate Index Variable)，如测量时间等。我们考虑如下变系数函数模型(Varying Coefficient Models,VCM):

$$Y_i = \boldsymbol{X}_i^\top \boldsymbol{a}(U_i) + \varepsilon_i$$

其中ε_i是随机误差项，并且满足$E(\varepsilon_i|\boldsymbol{X}_i, U_i) = 0$。

对于变系数函数模型，我们利用局部线性光滑技术，即将$a_j(v)$在u点展开

$$a_j(v) \approx a_j(u) + b_j(u)(v - u),$$

其中$b(\bullet)=a'(\bullet)$。变系数模型是求下面目标函数的最小值：

$$\sum_{i=1}^{n}\Big\{Y_i-\boldsymbol{X}_i^{\top}\boldsymbol{a}(U)-(U_i-U)\boldsymbol{X}_i^{T}\boldsymbol{b}(U)\Big\}^2K_h(U_j-U)$$

其中$K_h(\bullet)=\frac{1}{h}K(\frac{\bullet}{h})$是核函数，$h$是带宽(Bandwidth)。

令W_t=diag$\Big((U_1-U)^tK_h(U_2-U),\ldots,(U_n-U)^tK_h(U_i-U)\Big)$，通过求导等运算，可以得到：

$$\begin{pmatrix}\mathbf{a}(u)\\ \mathbf{b}(u)\end{pmatrix}=\begin{pmatrix}\mathbf{X}^{\top}W_0\mathbf{X} & \mathbf{X}^{\top}W_1\mathbf{X}\\ \mathbf{X}^{\top}W_1\mathbf{X} & \mathbf{X}^{\top}W_2\mathbf{X}\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{X}^{\top}W_0\mathbf{Y}\\ \mathbf{X}^{\top}W_1\mathbf{Y}\end{pmatrix}$$

h比较重要，其越大估计越光滑，但估计偏差会增大。我们利用Cai等(2000)提出的多核实交叉核实的方法(Modified Multi-fold Cross-validation)选择h，其是最小化：

$$AMS(h_2)=\sum_{q=1}^{Q}AMS_q(h_2)$$

其中

$$AMS_q(h_2)=\frac{1}{m}\sum_{i=n-qm+1}^{n-qm+m}\Big\{Y_i-\boldsymbol{X}_i^{\top}\widehat{\boldsymbol{a}}_q(U_i)\Big\}^2$$

和$\widehat{\boldsymbol{a}}_q$ 利用$\{\boldsymbol{X}_i,Y_i,U_i\}_{i=1}^{n-qm}$计算得到。Cai等(2000)建议$m=[0.1n]$ 和$Q=4$ ($[c]$表示c的整数部分)。主要$n>Qm$。下面我们以一个模拟的例子说明如何进行编程。

我们考虑如下变系数模型

$$Y_i=8\exp(-(U_i-0.5)^2)X_{i1}+2\sin(2\pi U_i)+5U_i(1-U_i)X_{i2}+\varepsilon_i$$

其中$X_{i1}=1$，$(X_{i2},\ldots,X_{i3})^T$ 是独立同分布，来自于标准正态分布，$\varepsilon_i\overset{\text{iid}}{\sim}N(0,1)$，$U$是指示变量，其来自于[0,1]区间的均匀分布。我们设置样本量$n=200$。

R 代码

```
library(MASS)
library(limSolve)
```

```
#产生数据开始
Data1 <- function(n)
{
  p <- 2
  #sigm=0.5^abs(outer(1:3,1:3,"-"))
  sigm  <- diag(2)
  muz <- rep(0,2)
  x=mvrnorm(n = n, mu=muz, Sigma=sigm)
  x <- cbind(1,x)
  e <- rnorm(n,0,0.8)
  u <-  runif(n,0,1)
  beta0 <- 8*exp(-(u-0.5)^2)
  beta1 <- 2*cos(2*pi*u)
  beta2 <- 5*(u-0.5)^2
  yture <- beta0 + beta1 * x[,2] + beta2 * x[,3]
  y <- yture + e
  dat = list(y=y, u=u, x=x,beta0=beta0,beta1=beta1,beta2 =beta2)
  return(dat)
}
#产生数据结束


####变系数函数大函数开始
#核函数开始
kern <- function(t){
  0.75*pmax(1-t^2,0)
}
#核函数结束

#W函数开始
#这个函数可以不需要for循环，这样编写方便理解
wwi=function(n,s,j,h){
```

```
  ww=NULL
  for(i in 1:n){
    ww[i]=(u[i]-u[j])^s*kern((u[i]-u[j])/h)
  }
  ww
}
#W函数结束

#变系数函数开始
VCM=function(y,x,u,h){
  n=dim(x)[1]
  p=dim(x)[2]
  AB=matrix(0, nrow =2*p, ncol = n)
  for(j in 1:n){
    w0=w1=w2=matrix(0,n,n)
    diag(w0)=wwi(n=n,s=0,j=j,h=h)
    diag(w1)=wwi(n=n,s=1,j=j,h=h)
    diag(w2)=wwi(n=n,s=2,j=j,h=h)
    #合并矩阵a
    a11=t(x)%*%w0%*%x
    a12=t(x)%*%w1%*%x
    a22=t(x)%*%w2%*%x
    aa1=cbind(a11,a12)
    aa2=cbind(a12,a22)
    a=rbind(aa1,aa2)
    ###合并b
    b11=t(x)%*%w0%*%y
    b21=t(x)%*%w1%*%y
    b=rbind(b11,b21)
    AB[,j]=Solve(a) %*% b
  }
  list(AB=AB)
```

```
}
#变系数函数结束

##编写固定的h的MCV函数开始
VCM.MCV=function(y,x,u,h){
  n <- dim(x)[1]
  p <- dim(x)[2]
  m <- floor(0.1*n)
  Q <- 4
  AMS <- NULL
  for (q in 1:Q){
    ind  <- n - q * m
    xx <- x[1:ind, ]
    yy <- y[1:ind]
    uu <- u[1:ind]
    AB=matrix(0, nrow =2*p, ncol = m)
    for(j in 1:m){
      w0=w1=w2=matrix(0,ind,ind)
      diag(w0)=wwi(n=ind,s=0,j=ind+j,h=h)
      diag(w1)=wwi(n=ind,s=1,j=ind+j,h=h)
      diag(w2)=wwi(n=ind,s=2,j=ind+j,h=h)
      #合并矩阵a
      a11=t(xx)%*%w0%*%xx
      a12=t(xx)%*%w1%*%xx
      a22=t(xx)%*%w2%*%xx
      aa1=cbind(a11,a12)
      aa2=cbind(a12,a22)
      a=rbind(aa1,aa2)
      ###合并b
      b11=t(xx)%*%w0%*%yy
      b21=t(xx)%*%w1%*%yy
      b=rbind(b11,b21)
```

```
      AB[,j]=Solve(a) %*% b
    }
    AMS0=NULL
    for(k in 1:m){
      AMS0[k] <- y[ind+k] - x[ind+k,] %*% AB[1:3,k]
    }
    AMS[q] <- mean(AMS0^2)
  }
  list(h=h,AMS=sum(AMS))
}
##编写固定的h的MCV函数结束

##编写多个h的MCV函数开始
AMS <- function(y,x,u,h=seq(0.01,0.2,by=0.02)){
  MAMS <- NULL
  for(t in 1:length(h)){
    MAMS[t] <- VCM.MCV(y=y,x=x,u=u,h=h[t])$AMS
  }

  idex=which(MAMS==min(MAMS))
  h.opt <- h[idex]
  list(h=h.opt)
}
##编写多个h的MCV函数结束

n <- 200
dat <- Data1(n)
x <- dat$x
y <- dat$y
u <- dat$u
beta0 <- dat$beta0
beta1 <- dat$beta1
```

```
beta2 <- dat$beta2

hopt <- AMS(y=y,x=x,u=u)$h #选择h
hopt#查看h

fit=VCM(y=y,x=x,u=u,h=hopt)#拟合VCM

par(mfrow=c(1,3))#图省略
uu <- order(u)
plot(u[uu],fit$AB[1,][uu],ylim=c(6,8.5),ylab="a1",type="o",xlab="")
points(u[uu],beta0[uu],col=2)
plot(u[uu],fit$AB[2,][uu],ylim=c(-2,3), ylab="a2",type="o",xlab="")
points(u[uu],beta1[uu],col=2)
plot(u[uu],fit$AB[3,][uu],ylim=c(-1,3),ylab="a3",type="o",xlab="")
points(u[uu],beta2[uu],col=2)

mean((fit$AB[1,]-beta0)^2)#查看估计差异
mean((fit$AB[2,]-beta1)^2)
mean((fit$AB[3,]-beta2)^2)
```

输出结果

```
> hopt#查看h
[1] 0.11
> mean((fit$AB[1,]-beta0)^2)#查看估计差异
[1] 0.01669084
> mean((fit$AB[2,]-beta1)^2)
[1] 0.01960522
> mean((fit$AB[3,]-beta2)^2)
[1] 0.03013411
```

7.5 参考文献

1. Cai, Z., Fan, J., & Yao, Q. (2000). Functional-coefficient regression models for nonlinear time series. Journal of the American Statistical Association, 95(451), 941-956.

2. Fan, J., & Zhang, W. (1999). Statistical estimation in varying coefficient models. Annals of Statistics, 1491-1518.

3. H*ä*rdle, W. K., M*ü*ller, M., Sperlich, S., & Werwatz, A. (2012). Nonparametric and semiparametric models. Springer Science & Business Media.

4. Shumway R. Applied statistical time series analysis. 1988, Prentice-Hall.

5. Soetaert K.等(2014). limSolve. R package version 1.5.5.1, `http://CRAN.R-project.org/package=limSolve`.

第8章　奇异值分解

8.1　引言

奇异值分解(Singular value decomposition)非常重要的分解，它广泛应用到统计学中。

8.2　奇异值分解

定义8.2.1 (矩阵的奇异值分解)　设$\mathbf{A} \in \mathbb{R}^{n\times m}$，则存在正交矩阵$\mathbf{U} \in \mathbb{R}^{n\times n}$和$\mathbf{U} \in \mathbb{R}^{m\times m}$ 使得

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$$

其中

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

且$\mathbf{\Sigma}_1 = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$，其对角线元素(奇异值)按照顺序从小到大顺序排列。

$$\sigma_1 \geq \cdots \geq \sigma_r \quad r = \mathrm{rank}(\mathbf{A})$$

奇异值分解的性质：

(1) $\mathbf{A}^\top = \mathbf{V}\mathbf{D}\mathbf{U}^\top$

(2) $\mathbf{A}\mathbf{A}^\top = \mathbf{U}\mathbf{D}\mathbf{U}^\top$

(3) $\mathbf{A}^\top\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$

奇异值与范数和行列式之间的关系

- $\mathbf{A}$的谱范数等于矩阵最大奇异值，即$\|\mathbf{A}\|_{spec} = \sigma_1$.
- $\mathbf{A}$的Frobenius范数等于矩阵的奇异值的欧式范数，即$\|\mathbf{A}\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$.
- $\mathbf{A}$行列式的绝对值等于矩阵的奇异值乘积，即$|\det(\mathbf{A})| = \prod_{i=1}^r \sigma_i$.

8.3 R的实现

svd可以实现矩阵奇异值分解。我们以一个简答的例子简单介绍软件实现。

奇异值分解代码

```
> rm(list=ls())
> A <- matrix(1:6, 2, 3)
> svd(A)
$`d`
[1] 9.5255181 0.5143006

$u
           [,1]       [,2]
[1,] -0.6196295 -0.7848945
[2,] -0.7848945  0.6196295

$v
           [,1]       [,2]
[1,] -0.2298477  0.8834610
[2,] -0.5247448  0.2407825
[3,] -0.8196419 -0.4018960

> ###check
> A.svd <- svd(A)
> A.svd$u %*% diag(A.svd$d) %*% t(A.svd$v)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> # t(A.svd$u)%*%A.svd$u
> crossprod(A.svd$u, A.svd$u)
              [,1]          [,2]
[1,] 1.000000e+00 2.220446e-16
```

```
28 [2,] 2.220446e-16 1.000000e+00
```

8.4 应用

8.4.1 降秩回归

降秩回归(Reduced-rank regression)主要解决多元因变量的模型(Izenman (1975))。为了简单我们讨论普通的线性回归[1]。

[1]内容来自于Yanjun老师《Statistical Computing》奇异值分解案例https://jun-yan.github.io/stat-5361//

参考文献

Izenman, A. J. (1975). Reduced-rank regression for the multivariate linear model. Journal of multivariate analysis, 5(2), 248-264.

第9章　QR分解

9.1　引言

QR(Singular value decomposition)非常重要的分解，它广泛应用到统计学中。

9.2　奇异值分解

定义9.2.1 (矩阵QR分解)　设$\mathbf{A} \in \mathbb{R}^{n\times m}$，则存在$\mathbf{Q} \in \mathbb{R}^{n\times p}$，且$\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_p$，以及一个上三角矩阵$\mathbf{R} \in \mathbb{R}^{p\times p}$ 使得

$$\mathbf{A} = \mathbf{QR}$$

9.3　R的实现

QR可以实现矩阵奇异值分解。我们以一个简答的例子简单介绍软件实现。

奇异值分解代码

```
> rm(list=ls())
> A <- matrix(1:6, 2, 3)
> A.qr <- qr(A)
>
> Q <- qr.Q(A.qr)
> crossprod(Q)
              [,1]          [,2]
[1,] 1.000000e+00 5.551115e-17
[2,] 5.551115e-17 1.000000e+00
>
> R <- qr.R(A.qr)
>
> Q %*% R
```

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

9.4 应用

9.4.1 线性回归

线性回归求解是一个技术问题，参数估计表达式$\widehat{\beta} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y}$。逆的解法比较复杂。假设

$$\mathbf{X} = \mathbf{QR}$$

则

$$\begin{aligned}
\widehat{\beta} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y} \\
&= (\mathbf{R}^\top\mathbf{Q}^\top\mathbf{QR})^{-1}\mathbf{R}^\top\mathbf{Q}^\top\mathbf{Y} \\
&= (\mathbf{R}^\top\mathbf{R})^{-1}\mathbf{R}^\top\mathbf{Q}^\top\mathbf{Y} \\
&= \mathbf{R}^{-1}\mathbf{R}^{-\top}\mathbf{R}^\top\mathbf{Q}^\top\mathbf{Y} \\
&= \mathbf{R}^{-1}\mathbf{Q}^\top\mathbf{Y}
\end{aligned}$$

第三步使用逆的性质$(\mathbf{AB})^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{-1}$。这样大大降低了计算成本。上三角矩阵的逆求解比普通矩阵逆好很多。下面我们模拟数据看看这个计算效果。我们比较了lm和QR分解，发现结果是一致。原因前者求解是基于QR分级得到的(是否？看看?rm文档)

QR代码

```
> rm(list=ls())
> library(MASS)
> ###data begin
> p <- 5
> n <- 100
> mu <- rep(0, p)
> sigm <- diag(p)
```

```
> x <- MASS::mvrnorm(n=n, mu=mu, Sigma=sigm)
> beta <- 1 + runif(p, min = 0, max = 1)
> y <- x %*% beta + rnorm(n)
> ###data end
>
> ###X QR begin
> x.qr <- qr(x)
> Q <- qr.Q(x.qr)
> R <- qr.R(x.qr)
> ###X QR end
>
> beta_hat <- solve(R) %*% t(Q) %*% y
> beta_ols <- coef(lm(y~x-1))
> cbind(beta_ols, beta_hat)
   beta_ols
x1 1.767375 1.767375
x2 1.641430 1.641430
x3 1.272231 1.272231
x4 1.335175 1.335175
x5 1.450043 1.450043
```

第三部分

优化算法

第10章　线性规划

线性规划(Linear Programming)是最常见的优化方法。统计学中很多问题都可以转化为线性规划，如分位数回归、复合分位数回归等。本章安排如下：1.1给出线性规划的一般表达式；1.2 介绍R的软件实现；1.3 阐述线性规划在统计学中的应用。

10.1　一般表达式

线性规划的一般表达式是

$$\min_{\boldsymbol{x}} \mathbf{A}\boldsymbol{x}$$
$$s.t.\ \mathbf{B}\boldsymbol{x} \leq \boldsymbol{w}$$

其中$\mathbf{A} \in \mathbb{R}^{n\times p}$，$\boldsymbol{x} \in \mathbb{R}^{p}$，$\mathbf{B} \in \mathbb{R}^{m\times p}$和$\boldsymbol{w} \in \mathbb{R}^{m}$。$\leq$ 可以是$\geq$。一般来说，我们需要将统计问题转化为线性规划的一般表达式。

10.2　R的实现

`lpSolve`包(Berkelaar等2015)中`lp` 函数可以实现线性规划，其一般形式是

```
lp(direction = "min", objective.in, const.mat,
                      const.dir, const.rhs)
```

用法：

`objective.in`:A

`const.mat`: B

`const.dir`: 条件的符号（"<," "<=," "=," "==," ">," 或 ">="）

`const.rhs`: w

10.3　应用

这一部分，我们将介绍线性规划在分位数回归、复合分位数回归中的应用。

10.3.1 分位数回归

分位数回归(Quantile Regression)由Koenker和Bassett(1978)。相比均值回归，它可以全面刻画自变量对条件因变量的分布，对异常值有很强的抵抗性。Yu 等(2003)对分位数回归的发展做了综述。R包`quantreg` (Koenker等2015)可以实现分位数回归、惩罚分位数回归和分位数可加模型等统计模型。设$\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_n)^\top$, $\boldsymbol{X}_i = (X_{i1}, X_{i2}, \ldots, X_{ip})^\top$ ，我们考虑线性模型：

$$Y_i = \boldsymbol{X}_i^T\boldsymbol{\beta} + \varepsilon_i \;\; i = 1, 2, \ldots, n$$

其中τ是分位点(Quantile)，$\boldsymbol{\beta}$是p维参数向量。ε_i 是随机误差项，且满足$P(\varepsilon_i \leq 0|\boldsymbol{X}_i) = \tau$；则分位数回归是最小化

$$\sum_i^n \rho_\tau(Y_i - \boldsymbol{X}_i^\top\boldsymbol{\beta}) \tag{10.1}$$

其中$\rho_\tau(t) = \tau t I(t \geq 0) + (\tau - 1)tI(t < 0)$。我们引入松弛因子$\boldsymbol{u} = (u_1, u_2, \ldots, u_n)^\top$和$\boldsymbol{v} = (v_1, v_2, \ldots, v_n)^\top$，将(10.1)式转化为线性规划问题。令

$$Y_i - \boldsymbol{X}_i^\top\beta = u_i - v_i$$

其中$u_i = \max(0, Y_i - X_i^\top\boldsymbol{\beta})$表示正部，$v_i = \max(0, -(Y_i - \boldsymbol{X}_i^\top\boldsymbol{\beta}))$表示负部；则

$$\begin{aligned}
\sum_i^n \rho_\tau(Y_i - \boldsymbol{X}_i^\top\boldsymbol{\beta}) &= \sum_{i=1}^n(\tau u_i - (\tau - 1)v_i) \\
&= \tau\boldsymbol{I}_n^\top\boldsymbol{u} + (1 - \tau)\boldsymbol{I}_n^\top\boldsymbol{v} \\
&= (\mathbf{0}_{p\times 1}^\top\boldsymbol{\beta} + \tau\boldsymbol{I}_n^\top\boldsymbol{u} + (1 - \tau)\boldsymbol{I}_n^\top\boldsymbol{v} \\
&= (\mathbf{0}_{p\times 1}^\top, \tau\boldsymbol{I}_n^\top, (1 - \tau)\boldsymbol{I}_n^\top)(\boldsymbol{\beta}^\top, \boldsymbol{u}^\top, \boldsymbol{v}^\top)^\top \\
&\doteq \mathbf{A}^\top\boldsymbol{\gamma}
\end{aligned}$$

其中$\boldsymbol{I}_n$是n维单位向量，$\mathbf{A} = (\mathbf{0}_{p\times 1}^\top, \tau\boldsymbol{I}_n^\top, (1 - \tau)\boldsymbol{I}_n^\top)^\top$，$\boldsymbol{\gamma} = (\boldsymbol{\beta}^\top, \boldsymbol{u}^\top, \boldsymbol{v}^\top)^\top$。约束条件是

$$\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta} = \boldsymbol{u} - \boldsymbol{v}$$

即

$$\mathbf{X}\boldsymbol{\beta} - \boldsymbol{u} - \boldsymbol{v} = \boldsymbol{Y}$$

从而

$$\left[\begin{array}{ccc} \mathbf{X} & \mathbf{E}_n & -\mathbf{E}_n \end{array}\right] \left[\begin{array}{c} \boldsymbol{\beta} \\ \boldsymbol{u} \\ \boldsymbol{v} \end{array}\right] = \boldsymbol{Y}$$

令$\mathbf{B} = (\mathbf{X}, \mathbf{E}_n, -\mathbf{E}_n)$，其中$\mathbf{E}_n$是单位阵，则约束条件转化为$\mathbf{B}\boldsymbol{\gamma} = \boldsymbol{Y}$，从而分位数求解转化为

$$\begin{aligned} &\min \mathbf{A}\boldsymbol{\gamma} \\ &s.t. \mathbf{B}\boldsymbol{\gamma} = \boldsymbol{Y} \end{aligned}$$

R 代码

```
rm(list=ls(all=TRUE))#清空所有对象
library(lpSolve)#加载lpSolve包
####产生数据
n <- 20
p <- 2
tau <- 0.5
x <- rnorm(n)
y <- 1 + 2 * x + rnorm(n)
#线性规划
A <- c(rep(0, p), tau * rep(1, n), (1-tau) * rep(1, n))
B <- cbind(1, x, diag(n), -diag(n))
signe <- rep("=", n)#定义符号
fit <- lp(objective.in=A, const.mat=B,
          const.dir=signe, const.rhs=y)
#利用rq命令求解
library(quantreg)
fit.rq <- rq(y~x)
#比较结果
#求解的比较
fit$solution[1: p]#线性规划的解
fit.rq$coefficients#rq的解
```

```
#残差的比较
u <- fit$solution[(p+1):(p+n)]
v <- fit$solution[(p+1+n):(p+n+n)]
u-v#线性规划得到的残差
fit.rq$residuals#rq得到的残差
#检验残差与fit.rq$residuals一致
sum(abs(u - v - fit.rq$residuals))
```

输出结果

```
> fit$solution[1: p]#线性规划的解
[1] 0.6983842 1.8315662
> fit.rq$coefficients#rq的解
(Intercept)           x
  0.6983842   1.8315662
> u-v#线性规划得到的残差
 [1]  0.23533171 -0.18420287 -0.41254109 -0.60389082 -1.02674518
 [6] -0.50596114 -2.28958999  1.38345430  1.35550268  0.07200396
[11] -0.14516840  0.00000000  0.42308688 -0.36143752  0.34093697
[16]  0.00000000 -1.47848291  1.83394280  0.22600003  1.62160937
> fit.rq$residuals##rq得到的残差
          1           2           3           4           5
 0.23533171 -0.18420287 -0.41254109 -0.60389082 -1.02674518
          6           7           8           9          10
-0.50596114 -2.28958999  1.38345430  1.35550268  0.07200396
         11          12          13          14          15
-0.14516840  0.00000000  0.42308688 -0.36143752  0.34093697
         16          17          18          19          20
 0.00000000 -1.47848291  1.83394280  0.22600003  1.62160937
> #检验残差与fit.rq$residuals一致
> sum(abs(u - v - fit.rq$residuals))
[1] 1.387217e-11
```

输出结果显示：我们的编程与quantreg中的rq结果几乎一样(原因，大家可以用?rq和?lpSolve对比文档看看)。

注：(1)分位数回归的结果(参数估计值)与均值回归的结果没有可比性。因为分位数回归得到的(条件)分位数回归线，而均值回归得到(条件)均值回归线。一般来说，如果误差项的分布中位数和均值相等(如标准正态分布)时，中位数回归与均值回归几乎一样；所以有时文章中会将中位数回归和均值回归相比较。这可以说明数据是否含有异常值。如果两个回归线差距比较大，那么数据有可能存在异常值。(2) 分位数回归是求解(10.1)式。读者注意$\boldsymbol{\beta}$实际与τ有关。不同的τ，得到的不同$\boldsymbol{\beta}$。高分位点的分位数回归线理论上应该高于低分位点的分位数回归线，但利用(10.1)式却不能保证。关于如何解决分位数回归相交的方法，详见Yu和Jones(1998)。

10.3.2　复合分位数回归

复合分位数回归(Composite Quantile Regression)是Zou和Yuan(2008)年提出，它可以克服均值回归的局限性。均值回归对异常值比较敏感。即数据存在异常值时，其估计量会失效。而复合分位数回归通过复合多个分位数回归实现，对于异常值具有抵抗力。计算表明复合分位数的相对效率下界为70%。在误差服从正态分布、双指数分布和混合正态分布情形下，其估计量的相对效率均在95% 以上。我们考虑线性模型：

$$Y_i = \boldsymbol{X}_i^\top \boldsymbol{\beta} + \varepsilon_i$$

其中$\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)^\top$是$d$维未知参数向量，$\varepsilon_i$是随机误差项，且满足$P(\varepsilon_i \leq b_m|\boldsymbol{X}_i) = \tau_m$。复合分位数回归是通过复合多个分位数回归实现的，即求解下面目标函数：

$$(\hat{b}_1, \ldots, \hat{b}_K, \hat{\boldsymbol{\beta}}) = \arg\min \sum_{k=1}^{K} \Big\{ \sum_{i=1}^{n} \rho_{\tau_k}(Y_i - b_k - \boldsymbol{X}_i^\top \boldsymbol{\beta}) \Big\} \tag{10.2}$$

其中$0 < \tau_1 < \tau_2 < \cdots < \tau_M < 1$。

与1.3.1一样，我们引入松弛因子$\boldsymbol{u}$和$\boldsymbol{v}$，将10.2式转化为线性规划问题。

第一步：约束条件

令

$$y_i - b_k - \boldsymbol{x}_i^\top \boldsymbol{\beta} = u_{ik} - v_{ik}$$

其中$u_{ik} = \max(0, y_i - b_k - \boldsymbol{x}_i^\top \boldsymbol{\beta})$表示正部，$v_i = \max(0, -(y_i - b_k - \boldsymbol{x}_i^\top \boldsymbol{\beta}))$表示负部。所以

$$Y_1 - b_{\mathbf{1}} - \boldsymbol{X}_1^\top \boldsymbol{\beta} = u_{1\mathbf{1}} - v_{1\mathbf{1}}$$

$$\cdots$$

$$Y_n - b_{\mathbf{1}} - \boldsymbol{X}_n^\top \boldsymbol{\beta} = u_{n\mathbf{1}} - v_{n\mathbf{1}}$$

$$\cdots$$

$$Y_1 - b_{\boldsymbol{K}} - \boldsymbol{X}_1^\top \boldsymbol{\beta} = u_{1\boldsymbol{K}} - v_{1\boldsymbol{K}}$$

$$\cdots$$

$$Y_n - b_{\boldsymbol{K}} - \boldsymbol{X}_n^\top \boldsymbol{\beta} = u_{n\boldsymbol{K}} - v_{n\boldsymbol{K}}$$

则

$$\begin{bmatrix} Y_1 \\ \vdots \\ Y_n \\ \cdots \\ Y_1 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1\ 0\ \ldots\ 0\ \boldsymbol{X}_1^\top & 1\ 0\ \ldots & 0 & \ldots\ 0\ 0\ \ldots & 0 & \ldots & -1 & 0\ \ldots & 0 & \ldots & 0 & 0\ \ldots & 0 \\ 1\ 0\ \ldots\ 0\ \boldsymbol{X}_2^\top & 0\ 1\ \ldots & 0 \ldots & 0\ 0\ \ldots & 0 & \ldots & 0 & -1\ \ldots & 0 & \ldots & 0 & 0\ \ldots & 0 \\ & & & \cdots & & & & & & & & & \\ 1\ 0\ \ldots\ 0\ \boldsymbol{X}_n^\top & 0\ 0\ \ldots & 1 \ldots & 0\ 0\ \ldots & 0 & \ldots & 0 & 0\ \ldots & -1 & \ldots & 0 & 0\ \ldots & 0 \\ & & & \cdots & & & & & & & & & \\ & & & \cdots & & & & & & & & & \\ 0\ 0\ \ldots\ 1\ \boldsymbol{X}_1^\top & 0\ 0\ \ldots & 0 \ldots & 1\ 0\ \ldots & 0 & \ldots & 0 & 0\ \ldots & 0 & \ldots & -1 & 0\ \ldots & 0 \\ 0\ 0\ \ldots\ 1\ \boldsymbol{X}_2^\top & 0\ 0\ \ldots & 0 \ldots & 0\ 1\ \ldots & 0 & \ldots & 0 & 0\ \ldots & 0 & \ldots & 0 & -1\ \ldots & 0 \\ & & & \cdots & & & & & & & & & \\ 0\ 0\ \ldots\ 1\ \boldsymbol{X}_n^\top & 0\ 0\ \ldots & 0 \ldots & 0\ 0\ \ldots & 1 & \ldots & 0 & 0\ \ldots & 0 & \ldots & 0 & 0\ \ldots & -1 \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_K \\ \boldsymbol{\beta} \\ u_{1\mathbf{1}} \\ u_{2\mathbf{1}} \\ \vdots \\ u_{n\mathbf{1}} \\ \vdots \\ u_{1\boldsymbol{K}} \\ u_{2\boldsymbol{K}} \\ \vdots \\ u_{n\boldsymbol{K}} \\ v_{1\mathbf{1}} \\ v_{2\mathbf{1}} \\ \vdots \\ v_{n\mathbf{1}} \\ \vdots \\ v_{1\boldsymbol{K}} \\ v_{2\boldsymbol{K}} \\ \vdots \\ v_{n\boldsymbol{K}} \end{bmatrix}$$

所以得到

$$KY = \Delta\boldsymbol{\Theta} \tag{10.3}$$

其中将$\boldsymbol{\Theta} = (\boldsymbol{b}^\top, \boldsymbol{\beta}^\top, \boldsymbol{u}^\top, \boldsymbol{v}^\top)^\top$，$\Delta = [\Delta_1, \Delta_2, \Delta_3, \Delta_4]$。$\Delta_1$关于$\boldsymbol{b}$，$\Delta_2$关于$\boldsymbol{\beta}$，$\Delta_3$关于$\boldsymbol{u}$, 以及$\Delta_4$关于$\boldsymbol{v}$.

第二步：目标函数

$$
\begin{aligned}
&\sum_{k=1}^{K}\Big\{\sum_{i=1}^{n}\rho_{\tau_k}(Y_i - b_k - \boldsymbol{X}_i^\top\boldsymbol{\beta})\Big\} \\
&= \sum_{k=1}^{K}\sum_{i=1}^{n}(\tau u_{ik} - (\tau-1)v_{ik}) \\
&= \tau_1 I_n^\top u_1 + (1-\tau_1)\boldsymbol{I}_n^\top \boldsymbol{v}_1 + \cdots + \tau_K \boldsymbol{I}_n^\top u_K + (1-\tau_K)\boldsymbol{I}_n^\top v_K \\
&= (\mathbf{0}_{K\times 1}^\top \boldsymbol{b} + \mathbf{0}_{p\times 1}^\top \boldsymbol{\beta} + \tau_1 \boldsymbol{I}_n^\top u_1 + \cdots + \tau_K \boldsymbol{I}_n^\top u_K + (1-\tau_1)\boldsymbol{I}_n^\top v_1 + \cdots + (1-\tau_K)\boldsymbol{I}_n^\top v_K \\
&= (\mathbf{0}_{K\times 1}^\top, \mathbf{0}_{p\times 1}^\top, \tau_1\boldsymbol{I}_n^\top, \ldots, \tau_K\boldsymbol{I}_n^\top, (1-\tau_1)\boldsymbol{I}_n^\top, \ldots, (1-\tau_K)\boldsymbol{I}_n^\top))(\boldsymbol{b}^\top, \boldsymbol{\beta}^\top, \boldsymbol{u}^\top, \boldsymbol{v}^\top)^\top \\
&\doteq \mathbf{A}^\top\boldsymbol{\Theta}
\end{aligned}
$$

下面我们编写复合分位数回归函数cqr，并且用模拟例子验证代码的有效性。

R 代码

```
library(lpSolve)
cqr <- function(x,y,tau)
{
  p <-  dim(x)[2]
  n <-  dim(x)[1]
  k <- length(tau)
  ##产生\Delta_{1}即b
  bmatrix <- matrix(c(rep(1,n), rep(0,n*k)),n*k,k+1)[,-(k+1)]
  ##产生\Delat_{1}和\Delta_{2}
  #\Delata_{2} <- t(matrix(t(x),p,n*k))
  xx<-  cbind(bmatrix,t(matrix(t(x), p,n*k)))
  #KY
  yy <- rep(y,k)
  #产生\Delta_{3}和\Delta_{4}
  onematrix <- cbind(diag(1,n*k), - diag(1,n*k))
```

```
  #产生B
  B <- cbind(xx,onematrix)

  #产生u和v对应的A的部分
  tauv <- c(t(matrix(tau,k,n)))
  tauv <- c(tauv,1-tauv)
  A <- c(rep(0,k+p),tauv)

  signe <- rep("=",n*k)
  fit <- lp(objective.in=A,const.mat=B,
              const.dir=signe,const.rhs=yy)
  beta <- fit$solution[(k+1):(p+k)]
  b <- fit$solution[1:k]
  return(list(beta=beta,b=b))
}

n <- 100
p <- 2
a <- 2*rnorm(n*2*p, mean = 1, sd =1)
x <- matrix(a,n,2*p)
beta <- 2*rnorm(p,1,1)
beta <- rbind(matrix(beta,p,1), matrix(0,p,1))
beta
y <- x %*% beta - matrix(rnorm(n,0.1,1),n,1)
tau <- 1:5/6

cqr(x,y,tau)
```

输出结果

```
> beta <- rbind(matrix(beta,p,1), matrix(0,p,1))
> beta
          [,1]
[1,] 5.521803
[2,] 2.369093
[3,] 0.000000
[4,] 0.000000
> cqr(x,y,tau)
$beta
[1] 5.378556 2.328535 0.000000 0.000000
$b
[1] 0.0000000 0.0000000 0.2390532 0.7334682 1.3397366
```

输出结果显示cqr的结果与真实beta差距比较较小，这印证了编程的有效性。读者可以修改程序中的A和B编写AdaptiveLASSO复合分位数回归的表达式(Zou和Yuan 2008)：

$$(\hat{b}_1,\ldots,\hat{b}_K,\hat{\boldsymbol{\beta}}) = \arg\min \sum_{k=1}^{K}\Big\{\sum_{i=1}^{n}\rho_{\tau_k}(Y_i - b_k - \boldsymbol{X}_i^{\top}\boldsymbol{\beta})\Big\} + \lambda\sum_{j=1}^{p}\frac{|\beta_j|}{|\hat{\beta}_j|^2}$$

其中λ是调节参数，可以利用交叉核实的方法选择。

10.3.3　AdaptiveLASSO复合分位数回归

AdaptiveLASSO复合分位数回归的表达式(Zou和Yuan 2008)

$$(\hat{b}_1,\ldots,\hat{b}_K,\hat{\boldsymbol{\beta}}) = \arg\min \sum_{k=1}^{K}\Big\{\sum_{i=1}^{n}\rho_{\tau_k}(Y_i - b_k - \boldsymbol{X}_i^{\top}\boldsymbol{\beta})\Big\} + \lambda\sum_{j=1}^{p}\frac{|\beta_j|}{|\hat{\beta}_j|^2}$$

只需要将10.3式中

$$KY = \Delta\boldsymbol{\Theta} \tag{1}$$

Δ修改为$\Delta = [\Delta_1, \Delta_2, -\Delta_2, \Delta_3, \Delta_4]$，其中其中将$\boldsymbol{\Theta} = (\boldsymbol{b}^\top, \boldsymbol{\beta}^{+T}, \boldsymbol{\beta}^{-T}, \boldsymbol{u}^\top, \boldsymbol{v}^\top)^\top$
另外，将A修改为

$$(0^\top_{K\times 1}, \frac{1}{\lambda}\Lambda^\top_{p\times 1}, -\frac{1}{\lambda}\Lambda^\top_{p\times 1}, \tau I^\top_{nK}, (1-\tau)I^\top_{nK})$$

其中$\Lambda = \{\frac{1}{|\hat{\beta}_j|^2}, \ldots, \frac{1}{|\hat{\beta}_p|^2}\}^\top$

λ选择可以采用类AIC准则：

$$LAIC = 2p - log\Big[\sum_{k=1}^{K}\Big\{\sum_{i=1}^{n}\rho_{\tau_k}(Y_i - \hat{b}_k - \boldsymbol{X}_i^\top\hat{\boldsymbol{\beta}})\Big\}\Big]$$

理论上需要λ的界限，但一般比较难以得到。

我们编写函数`cqr.lasso.ip`实现AdaptiveLASSO复合分位数回归。也许由于选择λ方法的原因，效果并不理想。大家可以试试其它方法。

R 代码

```
cqr.lasso <- function(x, y, tau, lambda)
{
   if(missing(lambda)){
    lambda <- 1
  }
  p <- dim(x)[2]
  n <-  dim(x)[1]
  k <- length(tau)
  betaold <- cqr(x,y,tau)$beta
  #将betaold中为零的赋值为0.0001
  betaold[betaold==0] <- 0.0001
  ######产生B开始
  #b
  bmatrix <- matrix(c(rep(1,n),rep (0,n*k)),n*k,k+1)[,-(k+1)]
  ##\Delta_{1},\Delta_{2},-\Delta_{2},
  xx <- cbind(bmatrix,t(matrix(t (x),p,n*k)),
              -t(matrix(t (x),p,n*k)))
  yy <- rep(y,k)
```

```
  #\Delta_{4}, \Delta_{4}
  onematrix <- cbind(diag(1,n*k),-diag(1,n*k))
  ##\Delta
  B <- cbind(xx,onematrix)
  #####产生B结束

  #####产生A开始
  ##u和v对应的A部分
  tauv <- c(t(matrix(tau,k,n)))
  tauv <- c(tauv,1-tauv)
  A <- c(rep(0,k),c(lambda/betaold^2),-c(lambda/betaold^2),tauv)
  #####产生A结束

  signe <- rep("=",n*k)
  fit <- lp(objective.in=A,const.mat=B, const.dir=signe,const.rhs=yy)
  beta <- fit$solution[(k+1):(p+k)]-fit$solution[(k+1+p):(p+k+p)]
  b <- fit$solution[1:k]
  return(list(beta=beta,b=b,object=fit$objval))
  #object目标函数的值
}


n <- 100
p <- 2
a <- 2 * rnorm(n * 2 * p, mean = 1, sd =1)
x <- matrix(a, n, 2 * p)
beta <- 2 * rnorm(p, 1, 1)
beta <- rbind(matrix(beta,p,1), matrix(0,p,1))
beta
y <- x %*% beta-matrix(rnorm(n,0.1,1),n,1)
tau <- 1: 5/6
```

```
cqr(x,y,tau)
###怎么选择\lambda
lambda <- 10^(-6)
fitt <- cqr.lasso(x,y,tau,lambda)
fitt
2*p-log(fitt$object)
###利用AIC准则选择\lambda
lambda <- seq(0,10^(-6),by=10^(-7))
nn <- length(lambda)
LAIC <- NULL
for(i in 1: nn){
  fitt <- cqr.lasso(x,y,tau,lambda[i])
  LAIC <- 2*p-log(fitt$object)
}

index <- which(LAIC==min(LAIC))
index
LAIC[index]
fitt.best <- cqr.lasso(x,y,tau,lambda[index])
fitt.best
```

输出结果

```
> beta
         [,1]
[1,] 5.263735
[2,] 1.867684
[3,] 0.000000
[4,] 0.000000
> y <- x% * %beta-matrix(rnorm(n,0.1,1), n, 1)
> tau <- 1: 5/6
```

```
>
> cqr(x,y,tau)
$beta
[1] 5.159860 1.870044 0.000000 0.000000
$b
[1] 0.0000000 0.0000000 0.3184685 0.7138606 1.2719789
> fitt <- cqr.lasso(x,y,tau,lambda)
> fitt
$beta
[1]  5.1690092  1.9200948 -0.1755381 -0.2303625
$b
[1] 0.0000000 0.3399789 0.8506159 1.3134055 1.9973771
$object
[1] 131.3473
> 2*p-log(fitt$object)
[1] -0.8778447
> index
[1] 1
> LAIC[index]
[1] -0.8778447
> fitt.best <- cqr.lasso(x,y,tau,lambda[index])
> fitt.best
$beta
[1]  5.16124087  1.89313983 -0.03127703 -0.07522966
$b
[1] 0.0000000 0.0000000 0.4936240 0.8802032 1.3124110
$object
[1] 154.8611
```

10.4 参考文献

1. Bendtsen C. (2012). lpSolve. R package version 5.6.13, `http://CRAN.R-project.org/package=lpSolve`.

2. Koenker R., Bassett G. (1978). Regression quantiles. Econometrica, 46, 33 - 50.

3. Koenker R.等(2015). quantreg. R package version 5.19, `http://CRAN.R-project.org/package=quantreg`.

4. Yu, K., Jones, M. (1998). Local linear quantile regression. Journal of the American statistical Association, 93(441), 228-237.

5. Yu K., Lu Z., Stander J. (2003). Quantile regression: applications and current research areas. Journal of the Royal Statistical Society: Series D (The Statistician), 52, 331–350.

6. Zou H., Yuan M.(2008). Composite quantile regression and the oracle model selection theory. The Annals of Statistics,36(3): 1108 - 1126.

第11章　二次规划

上一章，我们介绍了线性规划，这一章将介绍二次规划。顾名思义，二次规划将出现参数的二次项。统计中的约束最小二乘可以利用二次规划求解。本章安排如下：2.1 给出二次规划的一般表达式；2.2 介绍R 的软件实现；2.3 给出另一种表达以及R 实现的命令；2.4 阐述二次规划在统计学中的应用。

11.1　一般表达式

二次规划(Quadratic Programming)的一般表达式是

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^\top \mathbf{D}\boldsymbol{x} - \boldsymbol{d}^\top \boldsymbol{x}$$
$$s.t.\ \mathbf{A}^\top \boldsymbol{x} \geq \boldsymbol{b}$$

其中$\mathbf{D} \in \mathbb{R}^{p\times p}$，$\boldsymbol{x} \in \mathbb{R}^p$，$\mathbf{A} \in \mathbb{R}^{p\times n}$和$\boldsymbol{b} \in \mathbb{R}^n$。

11.2　R的实现

`quadprog`包(Berwin等2013)中的`solve.QR` 函数可以实现，其一般用法是：

```
solve.QP(Dmat,dvec,Amat,bvec,meq=0)
```

其中：

`Dmat`: D

`dvec`: d

`Amat`: A

`bvec`: b

`meq`: 等号的个数，默认为0。需要注意的是等号应该放到大于号的上面，也就是写约束条件时，等式应该放到不等式的上面。

11.3　另一种表达式

二次规划的另一般表达是

$$\min_{\boldsymbol{\beta}} \boldsymbol{c}^\top \boldsymbol{\beta} + \frac{1}{2}\boldsymbol{\beta}^\top \mathbf{H}\boldsymbol{\beta}$$

$$
\begin{aligned}
&s.t.\\
&\boldsymbol{b} \leq \mathbf{A}\boldsymbol{\beta} \leq \boldsymbol{b} + \boldsymbol{r}\\
&\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}
\end{aligned}
$$

kernlab包(Karatzoglou等2015)中ipop可以实现，其用法是

```
ipop(c, H, A, b, l, u, r, sigf = 7, maxiter = 40, margin = 0.05,
     bound = 10, verb = 0)
```

两种表达形式可以互相转化，我们使用前面一个表达形式。

11.4 应用

11.4.1 约束最小二乘

$$\min \sum_{i=1}^{n} (Y_i - \boldsymbol{X}_i^T \boldsymbol{\beta})^2$$

其中$\mathbf{A}\boldsymbol{\beta} > \boldsymbol{b}$。由于

$$\sum_{i=1}^{n} (Y_i - \boldsymbol{X}_i^\top \boldsymbol{\beta})^2 = \boldsymbol{\beta}^\top \mathbf{X}\mathbf{X}^\top \boldsymbol{\beta} - 2\boldsymbol{Y}^\top \mathbf{X}\boldsymbol{\beta}$$

如果令$\mathbf{D} = \mathbf{X}\mathbf{X}^\top$，$\boldsymbol{d} = \mathbf{X}^\top \mathbf{Y}$，则约束最小二乘变转化为二次规划。

R 代码

```
library(quadprog)
#产生数据
n <- 100
x <- rnorm(n)
y <- 2 + 3 * x + rnorm(n)
#转化为二次规划
XX <- cbind(1, x)
D <- t(XX) %*% XX
d <- t(XX) %*% y
```

```
A <- matrix(c(0,1),ncol=1)
b <- 2
solve.QP(Dmat=D,dvec=d,Amat=A,bvec=b)
```

输出结果

```
$solution
[1] 2.018696 2.928100
$value
[1] -588.5004
$unconstrained.solution
[1] 2.018696 2.928100
$iterations
[1] 1 0
```

11.5　参考文献

1. Berwin A. (2013). quadprog. R package version 1.5–5, http://CRAN.R-project.org/package=quadprog.

2. Karatzoglou A., Smola A., Hornik K. kernlab. (2015). R package version 0.9–22, http://CRAN.R-project.org/package=kernlab.

第12章　非线性方程

这一章主要介绍非线性方程(Nonlinear Equation) 的求解。之所以将它放到非线性优化(Nonlinear Optimization)前面介绍，主要因为BB包(Varadhan等2014)。BB 包主要解决在尺度(Large-scale) 的非线性问题，主要包括6个函数，3个水平：

- 第一水平包括3个函数：sane 和dfsane解决非线性方程；spg解决带有球约束(Box Constraint) 的非线性目标函数。
- 第二水平包括2个函数：BBsolve是dfsane的包裹(Wrapper)；BBoptim是sane 的包裹(Wrapper)。
- 最后水平包含1个函数：multiStart解决多个初始值的计算，可以看出算法的收敛情况。

BB包的优点：不需要Hessian矩阵，使用Barazilai-Borwein梯度(Raydan 1997)。

本章安排如下：3.1给出非线性方程的一般表达式；3.2 详细介绍R的软件实现；3.3阐述非线性方程在统计学中的应用：带常数项的泊松回归。该章主要参考BB包的使用文档(Varadhan 和Gilbert 2014)。

12.1　一般表达式

解决非线性一般的目的是求解

$$F(\boldsymbol{x}) = \mathbf{0} \tag{12.1}$$

其中$F : \mathbb{R}^p \to \mathbb{R}^p$，$\boldsymbol{x} \in \mathbb{R}^p$ 。

12.2　R的实现

BB中的sane、dfsane BBsolve可以实现，sane需要导数，dfsane中的df是导数释放(Derivative-free) 的意思。BBsolve是自动调节方法和参数。有时

候sane和dfsane 不收敛时，BBsolve可能收敛。详见下文论述。它们是使用方法是：

```
sane(par, fn, method=2, control=list(),  ...)
dfsane(par, fn, method=2, control=list(),  ...)
BBsolve(par, fn, method=c(2,3,1), control=list(), ...)
```

其中

```
par:  初始值
fn:  非线性函数
method : 谱步长，1，2和3对应3种方法，下文会详细论述。
          默认第2种方法。
control : 参数控制，下文将将介绍。
```

下面我们将详细论述函数使用。主要参考BB包的文档。

12.2.1 引言

下面从牛顿法开始，简单介绍求解12.1三种简单方法。

- 牛顿法(Newtion):它实施的是$F(\boldsymbol{x})$的线性逼近，其迭代表达式是：

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J(\boldsymbol{x}_k)^{-1}F(\boldsymbol{x})k)$$

其中$J : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p$ 是F在$\boldsymbol{x}_k$的雅可比(Jacobian)矩阵。

- 拟牛顿法(Quasi-Newton)避免求J，而是用方法逼近J，其迭代式是：

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - B_k^{-1}F(\boldsymbol{x}_k) \\ B_{k+1} &= B_k - \frac{F(\boldsymbol{x}_{k+1})(\boldsymbol{x}_{k=1} - \boldsymbol{x}_k)^\top}{(\boldsymbol{x}_{k=1} - \boldsymbol{x}_k)^\top(\boldsymbol{x}_{k=1} - \boldsymbol{x}_k)} \end{aligned}$$

其中B_0是单位矩阵。

注：上面两种方法用雅可比行列式或逼近雅可比矩阵解一个线性方程，不适合高维的p。

- 直接的方法是定义一个价值函数(Merit Function)$\phi : \mathbb{R}^p \to \mathbb{R}$，其在$\boldsymbol{u} = \boldsymbol{0}$存在全局极小值。任意$F(\boldsymbol{x}) = \boldsymbol{0}$的解都是$\phi(F(\boldsymbol{x}))$的解，但反过来不一定成立。一个充分的条件是$F$的雅可比矩阵在最小化$\phi(\boldsymbol{u})$是可逆。通常使用的$\phi$是$F$的$L_2$范数，即$\phi(F(\boldsymbol{x})) = \|F(x)\|$。

  注：这种方法实际中不一定很好，但可以作为初始值的产生方法。

12.2.2　SANE 和DF-SANE之局部收敛

SANE (La Cruz and Raydan 2003) 和DF-SANE (La Cruz, Martinez, Raydan 2006)由Raydan和他的同事提出，可以解决大刻度非线性方程的问题。他们是Barzilai-Borwei方法的推广，可以找到局部极小值。它们利用$\pm F(\boldsymbol{x})$作为收索方向(Search Direction)，其迭代式为：

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k \;\; k = 0, 1, 2, \ldots \tag{12.2}$$

其中α_k是谱步长(Spectral Steplength)。$\boldsymbol{d}_k$是收索方法，其定义为：

$$\boldsymbol{d}_k = \begin{cases} -F(\boldsymbol{x}_k) & \text{DF-SANE} \\ \pm F(\boldsymbol{x}_k) & \text{SANE} \end{cases}$$

其中α_k 主要有3种方法

- 第一种：简单记为`sane=1`和`dfsane=1`。

$$\alpha_k = \frac{\boldsymbol{s}_{k-1}^\top \boldsymbol{s}_{k-1}}{\boldsymbol{s}_{k-1}^\top \boldsymbol{y}_{k-1}} \tag{12.3}$$

  其中$\boldsymbol{s}_{k-1} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}$，$\boldsymbol{y}_{k-1} = F(\boldsymbol{x}_k) - F(\boldsymbol{x}_{k-1})$。

- 第二种：简单记为`sane=2`和`dfsane=2`。

$$\alpha_k = \frac{\boldsymbol{s}_{k-1}^\top \boldsymbol{y}_{k-1}}{\boldsymbol{y}_{k-1}^\top \boldsymbol{y}_{k-1}} \tag{12.4}$$

- 第三种：简单记为`sane=3`和`dfsane=3`。

$$\alpha_k = sgn(\boldsymbol{s}_{k-1}^\top \boldsymbol{y}_{k-1}) \frac{\|\boldsymbol{s}_{k-1}\|}{\|\boldsymbol{y}_{k-1}\|} \tag{12.5}$$

  其中当$v \neq 0$时$sgn(v) = v/|v|$，其它为0。

注：对于这三种方法，$\alpha_0 = \min(1.1/\|F(\boldsymbol{x}_0)\|)$。$\alpha_k$之所以有效因为它们与$J_k$的条件数有密切关系。

12.2.3 SANE 和DF-SANE之全局收敛

为了全局收敛，12.2式需要适当线性收索技术(Suitable Line Search Technique)。

对于SANE，La Cruz和Raydan (2003)考虑一个非增的线性收索技术(Grippo, Lampariello和Lucidi 1986, GLL条件)，其可以表示为：

$$f(\boldsymbol{x}_{k+1}) \leq \max_{0\leq j\leq M} f(\boldsymbol{x}_{k-j}) + \gamma\alpha_k \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{d}_k \tag{12.6}$$

其中:

1. 价值函数$f(\boldsymbol{x}) = F(\boldsymbol{x})^\top F(\boldsymbol{x})$。
2. γ是一个非常小的正数，一般设置为10^{-4}。
3. M 是一个正数，它决定f函数非增的自由度。当$M = 0$，严格递减，但Barzilai-Borwein 表现很差。一般M 在5和20之间。
4. 在SANE中，$\nabla f(\boldsymbol{x}_k)^\top \boldsymbol{d}_k$等于$\pm F_k^\top J_k F_k$，其中$F_k = F(\boldsymbol{x}_k)$，$J_k$是$F$在$\boldsymbol{x}_k$的雅可比矩阵。为了避免求$J_k$，也可以用逼近方法得到，即

$$F_k^\top J_k F_k \approx F_k^\top \Big[\frac{F(\boldsymbol{x}_k + hF_k) - F_k}{h}\Big]$$

其中$h = 10^{-7}$。

对于DF-SANE(导数释放的SANE，derivative-free SANE)，La Cruz等(2006)提出一种的全局线性收索技术：

$$f(\boldsymbol{x}_{k+1}) \leq \max_{0\leq j\leq M} f(\boldsymbol{x}_{k-j}) + \eta_k - \gamma\alpha_k^2 f(\boldsymbol{x}_k) \tag{12.7}$$

其中:

1. $\gamma = 10^{-4}$
2. $\eta_k > 0$ 随着k递减，且满足$\sum_{k=0}^{\infty}\eta_k = \eta < \infty$。注：添加它是为了避免计算雅可比矩阵。这也是DF-SANE被称为导数释放的原因。另外，它可保证了满足不增的GLL条件。它在每一步迭代中加入一个附加函数的取值，这有效避免了计算涉及到雅可比矩阵的二次内积(Quadratic Product)。因此，DF-SANE 相比SANE在计算F的个数上更加经济。$\eta_k > 0$保证每次迭代更好的执行(Well-defined)。

3. $-\gamma\alpha_k^2 f(\boldsymbol{x}_k)$ 是为了建立全局的收敛的理论条件。

12.2.4　SANE和DF-SANE在BB中的实现

SANE和DF-SANE主要使用BB中的sane和dfsane函数实现，需要注意的是：

1. 三种谱步长利用method可以设置。第一种是method=1对应12.3式，第二种是method=2对应12.4式，第三种是method=3 对应12.5式。默认是第二种，因为它一般优于其它两种。

2. BB的谱步长初始值是$\alpha_0 = \min\{1, 1/\|F(\boldsymbol{x}_0)\|\}$，否则原始的实施$\alpha_0 = 1$。

3. 参数初始值的设定。如果用户不设置。我们将价值函数$f(\boldsymbol{x})$作为目标函数，使用Nelder-Mead Nonlinear Simplex 算法(Nelder和Mead 1965，optim函数可以实现)得到。

4. 提高收敛速度。如果sane和dfsane不收敛时，我们将价值函数$f(\boldsymbol{x})$作为目标函数，利用BFGS算法(optim中设置method="L-BFGS-B")。如sane中设置convergence = 4 或5 ，以及dfsane中设置convergence = 2 或5。注：这不一定保证收敛。

5. 如果非常接近于解时，如$f(\boldsymbol{x}_k) < 10 - 4$时，我们使用动态延迟策略(Dynamic Retard Strategy, Luengo和Raydan, 2003):

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_{k-1}\boldsymbol{d}_k$$

6. 停止迭代。在noimp迭代次数中$f(\boldsymbol{x})$不降时停止。模拟noimp=100。当M很大时，这个准则非常有用，如$M > 100$。

12.2.5　*BBsolve*的选择

当sane和dfsane不收敛时，BBsolve函数将是一个不多的选择。一般采用的策略是：

1. 尝试改变M的取值：一般默认$M = 10$，可以尝试$M = 50$。

2. 尝试改变谱步长方法：一般默认method=2，可以尝试1和3.

3. 尝试Nelder-Mead(NM) 初始值：一般默认NM = FALSE，可以尝试 NM = TRUE。

下面使用Freudenstein-Roth函数演示方法的结果：

$$f(\mathbf{x}) = \Big\{x_1 - 13 + \big[(5 - x_2)x_2 - 2\big]x_2\Big\}^2 + \Big\{x_1 - 29 + \big[(x_2 + 1)x_2 - 14\big]x_2\Big\}^2$$

其中$\boldsymbol{x} = (x_1, x_2)^\top$。

R 代码

```
library(BB)
#Freudenstein-Roth 函数
froth <- function(p){
   r <- rep(NA, length(p))
   r[1] <- -13 + p[1] + (p[2] * (5 - p[2]) - 2) * p[2]
   r[2] <- -29 + p[1] + (p[2] * (1 + p[2]) - 14) * p[2]
  r
}
#初始值
p0 <- rep(0, 2)
dfsane(par = p0, fn = froth, control = list(trace = FALSE))
#control = list(trace = FALSE))不显示计算过程
sane(par = p0, fn = froth, control = list(trace = FALSE))
BBsolve(par = p0, fn = froth)
```

输出结果

```
> dfsane(par = p0, fn = froth, control = list(trace = FALSE))
$par
[1] -4.990589 -1.448398
$residual
```

```
[1] 10.42073
$fn.reduction
[1] 17.04337
$feval
[1] 131
$iter
[1] 106
$convergence  #说明sane不收敛
[1] 5
$message
[1] "Lack of improvement in objective function"
Warning message:
In dfsane(par = p0, fn = froth, control = list(trace = FALSE)) :
  Unsuccessful convergence.
> sane(par = p0, fn = froth, control = list(trace = FALSE))
$par
[1] -5.729871 -1.702569
$residual
[1] 9.592763
$fn.reduction
[1] 18.21428
$feval
[1] 417
$iter
[1] 127
$convergence #说明dfsane不收敛
[1] 5
$message
[1] "Lack of improvement in objective function"
Warning message:
In sane(par = p0, fn = froth, control = list(trace = FALSE)) :
  Unsuccessful convergence.
```

```
> BBsolve(par = p0, fn = froth)
  Successful convergence.
$par
[1] 5 4
$residual
[1] 2.012453e-09
$fn.reduction
[1] 6.998875
$feval
[1] 1109
$iter
[1] 233
$convergence
[1] 0
$message
[1] "Successful convergence"
$cpar #说明使用的谱步长，M的取值和初始值使用的方法。
method      M     NM
     1     50      1
```

12.2.6 *multiStart*的使用

上面一节，我们已经见识了BBsolve的“神奇”之处。如果存在多个解怎么办？multiStart可以有效解决这一个问题。它采用的策略是赋不同的初始值看起结果或收敛效果。它使用时将BBsolve作为自己的一个包裹，即每一个初始值都使用BBsolve计算，然后看结果。下面我们将一个例子说明。这个例子有12个实数根，126个复数根。

R 代码

```
hdp <- function(x) {
  f <- rep(NA, length(x))
```

```
  f[1] <- 5 * x[1]^9 - 6 * x[1]^5 * x[2]^2 + x[1] * x[2]^4
          + 2 * x[1] * x[3]
  f[2] <- -2 * x[1]^6 * x[2] + 2 * x[1]^2 * x[2]^3
          + 2 * x[2] * x[3]
  f[3] <- x[1]^2 + x[2]^2 - 0.265625
  f
}
# 长度为3的200个初始值
p0 <- matrix(runif(600), 200, 3)
ans <- multiStart(par=p0, fn=hdp, action="solve")
#计算收敛的次数 因为收敛ans$conv=0
sum(ans$conv)
#输出收敛的解
pmat <- ans$par[ans$conv, ]
#显示唯一的实数解
fit <- round(pmat, 4)
fit[!duplicated(fit), ]
pc <- princomp(pmat)
biplot(pc)#图省略 从图可以看出实数根收敛到一个参数矩阵
```

输出结果

```
> fit[!duplicated(fit), ]
         [,1]    [,2]    [,3]
 [1,]  0.5154  0.0000 -0.0124
 [2,]  0.4670  0.2181  0.0000
 [3,]  0.2799  0.4328 -0.0142
 [4,] -0.2799  0.4328 -0.0142
 [5,] -0.5154  0.0000 -0.0124
 [6,] -0.2799 -0.4328 -0.0142
```

```
 [7,]  0.4670 -0.2181  0.0000
 [8,]  0.2799 -0.4328 -0.0142
 [9,]  0.0000  0.5154  0.0000
[10,]  0.0000 -0.5154  0.0000
[11,] -0.4670 -0.2181  0.0000
[12,] -0.4670  0.2181  0.0000
```

注：虽然初始值不一样，但实数根时一样的。12个实数根。

12.3 应用

非线性方程在统计学中应用非常广泛，如估计方程的求解。

12.3.1 带常数项的泊松回归

这个例子是BB包文档自带的例子。带常数的泊松回归(Poisson Regression With Offset)是常见的回归分析，其主要解决因变量是可数的类型(Count)，如一段时间内事件发生的个数。Y_i是在时间t_i的观察事件出现的个数，$\boldsymbol{X}_i$是对应的自变量的观测值，则估计方程是：

$$\sum_{i=1}^{n} \boldsymbol{X}_i^\top \left[Y_i - t_i \exp(\boldsymbol{X}_i^\top \boldsymbol{\beta}) \right] = 0$$

我们模拟一组数据。设$\boldsymbol{\beta} = (-5, 0.04, 0.3, 0.05, 0.3, -0.005, 0.1, -0.4)^\top$。$Y_i|t_i \sim poission(t_i) * \boldsymbol{X}_i^\top \boldsymbol{\beta}$。$t_i \sim N(\mu = 100, \sigma = 10)$。$n = 500$，$p = 8$。$\boldsymbol{X}$的产生详见下面$R$代码。与$glm$函数相比，结果几乎一样。

R 代码

```
library(BB)
#估计方程
U.eqn <- function(beta) {
  Xb <- c(X %*% beta)
  c(crossprod(X, Y - (obs.period * exp(Xb))))
}
```

```
#数据的产生机制
poisson.sim <- function(beta, X, obs.period) {
  Xb <- c(X %*% beta)
  mea <- exp(Xb) * obs.period
  rpois(nrow(X), lambda = mea)
}

n <- 500
X <- matrix(NA, n, 8)
X[,1] <- rep(1, n)
X[,3] <- rbinom(n, 1, prob=0.5)
X[,5] <- rbinom(n, 1, prob=0.4)
X[,7] <- rbinom(n, 1, prob=0.4)
X[,8] <- rbinom(n, 1, prob=0.2)
X[,2] <- rexp(n, rate = 1/10)
X[,4] <- rexp(n, rate = 1/10)
X[,6] <- rnorm(n, mean = 10, sd = 2)
obs.period <- rnorm(n, mean = 100, sd = 10)
beta <- c(-5, 0.04, 0.3, 0.05, 0.3, -0.005, 0.1, -0.4)
Y <- poisson.sim(beta, X, obs.period)
#命令中没有附件X和Y。原因估计方程就没有这些参数。
res <- dfsane(par = rep(0,8), fn = U.eqn,
              control = list(NM = TRUE, M = 100, trace = FALSE))
res
#glm 命令结果
glm(Y ~ X[,-1], offset = log(obs.period),
    family = poisson(link = "log"))
```

输出结果

```
> res <- dfsane(par = rep(0,8), fn = U.eqn,
                 control = list(NM = TRUE, M = 100, trace = FALSE))
> res
$par
[1] -4.71563077  0.03498321  0.42322030  0.05216986
[2]0.24161694 -0.02897904  0.02020744 -0.45617348
$residual
[1] 9.937054e-08
$fn.reduction
[1] 9867.023
$feval
[1] 1143
$iter
[1] 986
$convergence
[1] 0
$message
[1] "Successful convergence"
> #glm 命令结果
> glm(Y ~ X[,-1], offset = log(obs.period),
        family = poisson(link = "log"))
Coefficients:
(Intercept)     X[, -1]1     X[, -1]2     X[, -1]3
   -4.71563      0.03498      0.42322      0.05217
   X[, -1]4     X[, -1]5     X[, -1]6     X[, -1]7
   0.24162     -0.02898      0.02021     -0.45617
Degrees of Freedom: 499 Total (i.e. Null);  492 Residual
Null Deviance:     1725
Residual Deviance: 538  AIC: 1680
```

从输出结果可以看出自编函数结果和glm的结果相差比较小，这也印证了自编函数的有效性。其中glm利用加权最小二乘法求解(详见?glm)。

12.4　参考文献

1. La C., Martinnez J., Raydan M (2006). Residual method without gradient information for solving large-scale nonlinear systems of equations. Mathematics of Computation, 75(255), 1429.

2. La C., Raydan M. (2003). Spectral methods for large-scale nonlinear systems. Optimization Methods and Software, 18(5), 583–599.

3. Luengo F., Raydan M. (2003). Gradient method with dynamical retards for large-scale optimization problems. Electronic Transactions on Numerical Analysis, 16, 186-193.

4. Nelder, J., Mead, R. (1965). A simplex method for function minimization. The computer journal, 7(4), 308-313.

5. Raydan M. (1997). The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. SIAM Journal on Optimization, 7(1), 26-33.

6. Grippo L., Lampariello F., Lucidi S. (1986). A nonmonotone line search technique for Newton's method. SIAM Journal on Numerical Analysis, 23(4), 707-716.

7. Varadhan R., Paul Gilbert P., Raydan M. BB. R package version 2014.10–1, http://CRAN.R-project.org/package=BB.

8. Varadhan R., Gilbert P. (2014). BB: An R ppackage for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function. https://cran.r-project.org/web/packages/BB/vignettes/BBvignetteJSS.pdf

第13章　参数约束的非线性优化

非线性优化(Nonlinear Optimization)可以分为参数约束、线性约束和非线性约束的非线性优化。这一章，我们主要介绍带有参数约束的非线性优化。本章安排如下：4.1给出参数约束非线性优化的一般表达式；4.2 详细介绍R的软件实现；4.3阐述参数约束非线性优化在统计学中的应用：众数回归。

13.1　一般表达式

参数约束的非线性优化的一般表达式为：

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t.\ \boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u$$

其中$\boldsymbol{x} \in \mathbb{R}^p$，$f(\boldsymbol{x})$是$\mathbb{R}^p \to \mathbb{R}$连续的函数，$\boldsymbol{x}_l$和$\boldsymbol{x}_u$是参数$\boldsymbol{x}$的上下界。

13.2　R的实现

BB包基本的优化函数是spg，高级版本是是BBoptim。它们可以解决带有球约束(Box-constraints)或其它利用投映的约束(Constraints Using Projection)。

```
spg(par, fn, gr=NULL, method=3, lower=-Inf, upper=Inf,
       project=NULL, projectArgs=NULL,  ...)
BBoptim(par, fn, gr=NULL, method=c(2,3,1), lower=-Inf, upper=Inf,
     project=NULL, projectArgs=NULL,   ...)
```

其中

par:　初始值
fn:　非线性函数
method : 谱步长，1，2和3对应3种方法，下文会详细论述。

默认第2种方法。

`lower` : 参数的下界`xl`

`upper` : 参数的上界`xu`

`project`和`projectArgs`: 参数投影设置，详见`?spg`。

13.3 应用

13.3.1 众数回归

相比均值回归的均值估计和分位数回归的分位数估计，众数回归是寻找最大可能的估计，即众数的估计。众数回归最早可以追朔到1989年(Lee 1989)。我们考虑如下模型：

$$Y_i = \boldsymbol{X}_i^\top \boldsymbol{\beta}_0 + \varepsilon_i, i = 1, 2, \ldots, n$$

其中$\boldsymbol{X}_i \in \mathbb{R}^p$，$\boldsymbol{\beta}_0$是属于参数空间$\mathbb{B} \subset \mathbb{R}^p$位置的参数。给定$\boldsymbol{x}_i$的$\varepsilon_i$ 条件密度函数在$\varepsilon_i = 0$有严格全局最大值，也即是$Mode(Y|X = x) = \boldsymbol{x}^\top \boldsymbol{\beta}_0$。

众数回归是求下面目标函数的最大值。

$$Q_n(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h_n} K\Big(\frac{Y_i - \boldsymbol{X}_i^\top \boldsymbol{\beta}}{h_n}\Big) \tag{13.1}$$

其中$K(\bullet)$是核函数，通常取高斯核函数。h_n是带宽(Bandwidth)，与n有关。关于h_n的选择，我们采用Kemp等(2012)提出的$h_n = k * MAD * n^{-0.143}$，其中$MAD = med\{|(Y_i - \boldsymbol{X}_i^\top \widehat{\boldsymbol{\beta}}_{ols}) - med(Y_i - \boldsymbol{X}_i^\top \widehat{\boldsymbol{\beta}}_{ols})|\}$，$k = 0.8$。从而

$$\hat{\boldsymbol{\beta}} = \arg\max_{\beta} Q_n(\boldsymbol{\beta})$$

关于13.1的求解，Yao和Li(2014)提出的MEM (Modal Expectation-maximization)算法。这种算法主要通过两步实施，类似于EM(Expectation-maximization)算法。

E步 ： 求

$$\pi(i|\boldsymbol{\beta}^k) = \frac{K_{h_n}(Y_i - \boldsymbol{X}_i^\top \boldsymbol{\beta}^k)}{\sum_{i=1}^{n} K_{h_n}(Y_i - \boldsymbol{X}_i^\top \boldsymbol{\beta}^{k+1})}$$

M步

$$\begin{aligned}\boldsymbol{\beta}^{k+1} &= \arg\max_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left[\pi(j|\boldsymbol{\beta}^k) \log K_{h_n}(Y_i - \boldsymbol{X}_i^\top \boldsymbol{\beta})\right] \\ &= (\boldsymbol{X}^\top \boldsymbol{W}_k \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W}_k \boldsymbol{y}\end{aligned}$$

其中$W = \text{diag}(\pi(1|\boldsymbol{\beta}^k), \pi(2|\boldsymbol{\beta}^k), \ldots, \pi(n|\boldsymbol{\beta}^k))$。

下面我们使用BB包求解13.1，并且与Yao和Li(2014)提出的方法进行比较。我们使用的模型是Yao和Li(2014)文章中模拟的例子：

$$Y = 1 + 3X + \sigma(X)\varepsilon$$

其中X来自$[0,1]$的均匀分布，$\varepsilon \sim 0.5N(-1, 2.5^2) + 0.5N(1, 0.5^2)$，$\sigma = 1 + 2X$。我们可以得到$E(\varepsilon) = 0$，$Mode(\varepsilon) = 1$，所以$Model(Y|X) = 2 + 5X$。

我们在编写代码时，有时不可能一次完工，可能需要很多次。下面是我在比较这两种方法编程的一个“完整”流程：首先先运行一次，然后在运行多次。代码先运行一次保证没有错误的情况下，才可以运行多次。不然很难发现错误。

R 代码

```
library(BB)
data1 <- function(n){
  x <- cbind(1,runif(n))
  ##genered error begin
  comp <- sample(c(0, 1), size = n, prob = c(0.5, 0.5),
                 replace = T)
  e <- rnorm(n, mean = ifelse(comp == 0, -1, 1),
             sd = ifelse(comp == 0, 2.5, 0.5))
  #plot(density(e))#看e的核密度估计

  sigmax <- 1 + 2 * x[, 2]
  beta <- c(1, 3)
  y <- x %*% beta + sigmax * e
```

```
  dat = list(y=y, x=x)
  return(dat)
}

n <- 1000
dat <- data1(n)
x <- dat$x
y <- dat$y
fit <- lm(y~x-1)
res <- fit$residuals
MAD <- quantile(abs(res-quantile(res, probs=0.5)),probs=0.5)
h <- as.numeric(0.8 * MAD* n^(-0.143)) #对结果影响比较大
theta0 <- fit$coefficients
theta <- theta0

library(BB)
################################## 自编函数开始
mrmy <- function(theta,y,x,h){
  kcol <-  dnorm( (y - x %*% theta) / h)
  return(sum(kcol))
}

lo <- c(-Inf, -Inf)
hi <- c(Inf, Inf)
po <- theta0
ans.my <- BBoptim(par=po, fn=mrmy, y=y, x=x, h=h,lower=lo,
                  upper=hi, control=list(maximize=T, trace=F))
ans.my
#赋不同初始值
p0 <- matrix(runif(20),10,2)
ans <- multiStart(par=p0, fn=mrmy, y=y, x=x, h=h,
                  action="optimize",lower=lo, upper=hi,
```

```
                    control=list(maximize=T) )
pmat <- round(cbind(ans$fvalue[ans$conv], ans$par[ans$conv, ]), 3)
dimnames(pmat) <- list(NULL, c("fvalue","parameter 1","parameter 2"))
pmat[!duplicated(pmat), ]
################################## 自编函数结束

################################## MEM开始
mrMEM <- function(theta, y, x, h){
  n <- dim(x)[1]
  p <- dim(x)[2]
  juli <- 1
  iter <- 0
  ##循环开始
  while(juli>10^(-3)){##juli<=0.1停止
    w <- dnorm((y- x %*% theta) / h)
    ww <- w/sum(w)
    # 第二步
    W <- matrix(0, n, n)
    diag(W) <- ww
    thetanew <- solve(t(x) %*% W %*% x) %*% t(x) %*% W %*% y
    cha <- abs(thetanew - theta)
    juli <- sum(cha ^ 2)
    theta <- thetanew
    iter<- iter + 1
  }
  ##循环结束
  return(list(iter=iter, theta= theta, juli=juli))
}

ans.mem <- mrMEM(theta=theta0, y=y, x=x, h=h)
ans.mem
################################## MEM结束
```

```
##两种方法比较
ans.my$par
ans.mem$theta
```

输出结果

```
> ans.my <- BBoptim(par=po, fn=mrmy, y=y, x=x, h=h,lower=lo,
                    upper=hi, control=list(maximize=T, trace=F))
  Successful convergence.
> ans.my
$par
      x1       x2
2.012864 5.314904
$value
[1] 109.0143
$gradient
[1] 1.421085e-07
$fn.reduction
[1] -71.81809
$iter
[1] 30
$feval
[1] 240
$convergence
[1] 0
$message
[1] "Successful convergence"
$cpar
method      M
     2     50
> pmat[!duplicated(pmat), ]#识别不相同的解
```

```
      fvalue parameter 1 parameter 2
[1,] 109.014       2.013       5.315
[2,]  74.254       3.422       1.172
> ans.mem
$iter
[1] 33
$theta
         [,1]
[1,] 2.041570
[2,] 5.134812
$juli
[1] 0.0009450327
> ans.my$par
      x1       x2
2.012864 5.314904
> ans.mem$theta
         [,1]
[1,] 2.041570
[2,] 5.134812
```

注：结果显示(13.1)可能存在多解。这也是众数回归的缺点。Chen等(2016)提出的方法可以借鉴。

R 代码

重复100次比较2两种方法：

```
beta.ture <- c(2,5)
Bias.my <- NULL
Bias.mem <- NULL
for(i in 1:100){
  n <- 500
  dat <- data1(n)
```

```
  x <- dat$x
  y <- dat$y
  fit <- lm(y~x-1)
  res <- fit$residuals
  MAD <- quantile(abs(res-quantile(res, probs=0.5)),probs=0.5)
  h <- as.numeric(0.8 * MAD* n^(-0.143)) #对结果影响比较大
  theta0 <- fit$coefficients
  ans.my <- BBoptim(par=po, fn=mrmy, y=y, x=x, h=h,lower=lo,
                    upper=hi, control=list(maximize=T, trace=F))
  ans.mem <- mrMEM(theta=theta0, y=y, x=x, h=h)
  Bias.my[i] <- sum(abs(ans.my$par-beta.ture))
  Bias.mem[i] <- sum(abs(ans.mem$theta-beta.ture))
}

mean(Bias.my)
mean(Bias.mem)
```

输出结果

```
> mean( Bias.my)
[1] 0.4928513
> mean( Bias.mem)
[1] 0.5373698
```

从输出结果可以看出，BB包算法得到的结果更好些。因为是偏差更小一些。

13.4　参考文献

1. Chen Y., Genovese C., Tibshirani R., Wasserman L. (2016). Nonparametric modal regression. The Annals of Statistics, 在线.

2. Kemp G., Silva, J. (2012). Regression towards the mode. Journal of Econometrics, 170(1), 92-101.

3. Lee M. (1989). Mode regression. Journal of Econometrics, 42, 337 - 349.

4. Yao W., Li, L. (2014). A new regression model: modal linear regression. Scandinavian Journal of Statistics, 41(3), 656-671.

第14章　线性和非线性约束的非线性优化

上一章，我们介绍了线性约束的非线性优化，这一章将介绍非线性约束的非线性优化。非线性优化可以由R包`Rdonlp2`可以实现。在安装这个包时，可以运行`install.packages("Rdonlp2", repos="http://R-Forge.R-project.org")`安装。如果上面命令失效，我们可以先`https://r-forge.r-project.org/R/?group_id=156` 下载安装包，然后利用本地加载安装。本章主要参考`Rdonlp2`的帮组文档(Tamura 2007)。除了`Rdonlp2`包外，`NlcOptim`包(Chen 和Yin 2015)也可以实现非线性约束的非线性方程。

本章安排如下：5.1给出非线性优化的一般表达式；5.2 介绍R的软件实现；5.3 给出另一种表达以及R实现的命令；5.4阐述非线性优化在统计学中的应用：经验似然超高维变量筛选。

14.1　一般表达式

带有线性或非线性约束的非线性优化的一般表达式为：

$$\begin{aligned}
&\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\
&s.t.\ \boldsymbol{x}_l \le \boldsymbol{x} \le \boldsymbol{x}_u, \\
&\quad \boldsymbol{b}_l \le \mathbf{A}\boldsymbol{x} \le \boldsymbol{b}_u, \\
&\quad \boldsymbol{c}_l \le c(\boldsymbol{x}) \le \boldsymbol{c}_u
\end{aligned}$$

其中$\boldsymbol{x} \in \mathbb{R}^p$，$f(\boldsymbol{x})$：$\mathbb{R}^p \to \mathbb{R}$，$\mathbf{A} \in \mathbb{R}^{n\times p}$。$\boldsymbol{x}_l$和$\boldsymbol{x}_u$ 是参数$\boldsymbol{x}$的上下界，$\boldsymbol{b}_l$和$\boldsymbol{b}_u$是$\boldsymbol{x}$的线性组合$\mathbf{A}\boldsymbol{x}$的上下界（线性约束），$\boldsymbol{c}_l$和$\boldsymbol{c}_u$是$\boldsymbol{x}$的非线性约束$c(\boldsymbol{x})$的上下界。

14.2 R的实现

```
donlp2 <- function(par, fn,
                   par.upper=rep(+Inf, length(par)),#xu
                   par.lower=rep(-Inf, length(par)),#xl

                   A = NULL,
                   lin.upper=rep(+Inf, length(par)),#bu
                   lin.lower=rep(-Inf, length(par)),#bl

                   nlin = list(),
                   nlin.upper=rep(+Inf, length(nlin)),#cu
                   nlin.lower=rep(-Inf, length(nlin)),#cl

                   control=donlp2.control(),
                   control.fun=function(lst){return(TRUE)} )
```

14.2.1 初始值

初始值需要用户给出，但Rdonlp2可以根据约束条件修订初始值。

14.2.2 目标函数和梯度

目标函数的值必须返回值必须是数值的。如：

```
objective.fun <- function(par){
    # 计算par并且返回值存储在ans中
    :
    ans # 返回值
}
:
ret <- donlp2(par=par, fn=objective.fun, ....)
```

给出梯度函数可以提高计算的精度和效率。梯度函数非常有用。一般利用函数的特性(Attibute)“gr”实现：

```
# par是长度为n的向量
grad.fun <- function(par){
c(v1, v2, ..., vn)
}
#提供目标函数的梯度函数
attr(objective.fn, "gr") <- grad.fun
```

14.2.3　约束

参数约束利用par.upper、par.lower和lin.upper、lin.lower以及nlin.upper、nlin.lower控制参数、线性和非线性约束。如

```
# par[1]<0, 0<par[2]<1, par[3]>1
par.lower <- c(-Inf, 0, 1)
par.upper <- c( 0, 1, +Inf)

# 两个参数的约束
# (1) par[1]+par[2]=0
# (2) par[1]-2*par[2]+10>0
lin.lower <- c(0, -10)
lin.upper <- c(0, +Inf)
```

14.2.4　线性约束

A中的每一行都代表参数的一个线性组合。如

```
# 两个参数的线性约束
# (1) par[1]+par[2]=0
# (2) par[1]-2*par[2]+10>0
lin.lower <- c(0, -10)
lin.upper <- c(0, +Inf)
A = rbind( c(1, 1), #第一个线性组合
           c(1,-2) ) # 第二个线性组合
```

14.2.5 非线性约束和其梯度

非线性约束中，用户也定义非线性约束函数的梯度。其梯度定义与目标函数梯度相似。如

```
# 2个参数1个非线性约束
# par[1]*par[2] = 1
nlcon1 <- function(par){
  par[1]*par[2]
}
nlcon1.gr <- function(par){
  c(par[2], par[1])
}
attr(nlcon1, "gr")<-nlcon1.gr
nlin.upper = c(1)
nlin.lower = c(1)
:
ret <- donlp2(par, fn,
              nlin=list(nlcon1),#所有非线性约束函数
              nlin.upper=nlin.upper,nlin.lower=nlin.lower,.....)
```

14.2.6 数值梯度

如果用户不定义梯度，donlp2使用下面三种方法计算数值梯度。数值梯度显然没有“真实”的梯度有效。一般情况，用户最好定义梯度。假设有n个参数。

1. 向前差分(Forward Difference)：需要n个函数的附加值(difftype=1)

2. 中心差分(Central Difference)：需要2n个函数的附加值(difftype=2)

3. 利用6阶逼近计算Richardson Extrapolation ：需要6n个函数的附加值(difftype=3)。默认的实施方法，精确但代价比较高。

14.2.7　一些设置

`iterma (4000)`：迭代次数。`difftype(3)`：数值差分类型`hessian(FALSE)`：Hessian矩阵`te0(TRUE)`：输出计算的每一步骤

14.2.8　输出结果

- `par`：参数估计值
- `gradf`：梯度
- `fx`：目标函数值
- `hesstype`：Hessian矩阵。默认`hessian=FALSE(default)`，需要修改为`hessian=TRUE`
- `xnorm`：参数估计值的L2范数

14.3　另一种表达式

$$\begin{aligned}\min_{\boldsymbol{x}} f(\boldsymbol{x})\ \ s.t.&\boldsymbol{x}\in\mathcal{S}\\ \mathcal{S}\in\{&\boldsymbol{x}\in R^n,\\ &ceq(\boldsymbol{x})=0\\ &\boldsymbol{x}\le\boldsymbol{x}_u,\\ &\mathbf{A}\boldsymbol{x}\le\boldsymbol{B},\\ &\mathbf{Aeq}\boldsymbol{x}\le\boldsymbol{Beq},\\ &\boldsymbol{lb}\le c(\boldsymbol{x})\le\boldsymbol{ub}\}\end{aligned}$$

其中$f(\boldsymbol{x})$是一个连续的函数，$\boldsymbol{lb}$和$\boldsymbol{ub}$是参数$\boldsymbol{x}$的上下界，$\boldsymbol{B}$是$\boldsymbol{x}$的线性组合$\mathbf{A}\boldsymbol{x}$的上界，$\boldsymbol{Beq}$是等号的约束（线性约束），$ceq(\boldsymbol{x})$的非线性约束$c(\boldsymbol{x})$的等号约束。

`NlcOptim`包(Chen 和Xin 2015)中`NlcOptim`可以实现。

```
NlcOptim(X = NULL, objfun = NULL,
```

```
        confun = NULL,
         A = NULL, B = NULL,
         Aeq = NULL, Beq = NULL,
         lb = NULL, ub = NULL,
```

需要注意约束函数confun的格式，如

```
confun=function(x){
f=NULL
f=rbind(f,x[1]^2+x[2]^2+x[3]^2+x[4]^2+x[5]^2-10)
f=rbind(f,x[2]*x[3]-5*x[4]*x[5])
f=rbind(f,x[1]^3+x[2]^3+1)
return(list(ceq=f,c=NULL))##ceq是等号
}
```

如果只含有非线性约束，它是一个不错的选择。在该章中，我们使用donlp2函数。

14.4 应用

14.4.1 复杂的例子

这个例子包括了参数、线性和非线性约束的所有类型。

$$\min_{x_i.i=1...10} 5.04x_1 + 0.035x_2 + 10x_3 + 3.36x_4 - 0.0063x_4x_7$$

$$s.t.$$

$$h_1(x) = 1.22x_4 - x_1 - x_5 = 0$$

$$h_2(x) = 98000x_3/(x_4x_9 + 1000x_3) - x_6 = 0$$

$$h_3(x) = (x_2 + x_5)/x_1 - x_8 = 0$$

$$g_1(x) = 35.82 - 0.222x_{10} - bx_9 \geq 0, b = 0.9$$

$$g_2(x) = -133 + 3x_7 - ax_{10} \geq 0, a = 0.99$$

$$g_3(x) = -g_1(x) + x_9(1/b - b) \geq 0$$

$$g_4(x) = -g_2(x) + (1/a - a)x_{10} \geq 0$$

$$g_5(x) = 1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - ax_4 \geq 0$$
$$g_6(x) = 57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x_6 - ax_7 \geq 0$$
$$g_7(x) = -g_5(x) + (1/a - a)x_4 \geq 0$$
$$g_8(x) = -g_6(x) + (1/a - a)x_7 \geq 0$$
$$0.00001 \leq x_1 \leq 2000$$
$$0.00001 \leq x_2 \leq 16000$$
$$0.00001 \leq x_3 \leq 120$$
$$0.00001 \leq x_4 \leq 5000$$
$$0.00001 \leq x_5 \leq 2000$$
$$85 \leq x_6 \leq 93$$
$$90 \leq x_7 \leq 95$$
$$3 \leq x_8 \leq 12$$
$$1.2 \leq x_9 \leq 4$$
$$145 \leq x_{10} \leq 162$$

经整理后，我们可以得到10个参数约束，5个线性约束（2个等式，3个不等式）：

$$h_1 \to 1.22x_4 - x_1 - x_5 = 0$$
$$g_1 \to -0.222x_{10} - bx_9 \geq -35.82, b = 0.9$$
$$g_2 \to 3x_7 - ax_{10} \geq 133, a = 0.99$$
$$g_3 \to x_9(1/b - b + b) + 0.222x_{10} \geq 35.82$$
$$g_4 \to -3x_7 + (1/a - a + a)x_{10} \geq -133$$

6个非线性约束（2个等式，4个不等式）：

$$h_2 \to 98000x_3/(x_4x_9 + 1000x_3) - x_6 = 0$$
$$h_3 \to (x_2 + x_5)/x_1 - x_8 = 0$$
$$g_5 \to 1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - ax_4 \geq 0$$
$$g_6 \to 57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x6 - ax7 \geq 0$$

$$g_7 \to -g5(x) + (1/a - a)x_4 \geq 0$$
$$g_8 \to -g6(x) + (1/a - a)x_7 \geq 0$$

R 代码

```
library(Rdonlp2)#加载Rdonlp2包
#复杂的例子
a <- 0.99; b <- 0.9

# 目标函数
fn <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4];
  x5 <- par[5]; x6 <- par[6]; x7 <- par[7]; x8 <- par[8];
  x9 <- par[9]; x10 <- par[10]
  5.04*x1 + 0.035*x2 + 10*x3 +3.36*x5 - 0.063*x4*x7
}

# 参数约束
par.l <- c(rep(1e-5, 5), 85, 90, 3, 1.2, 145)
par.u <- c(2000, 16000, 120, 5000, 2000, 93, 95, 12, 4, 162)

#线性和非线性约束
linbd <- matrix(0, nr=5, nc=2)#设置线性约束个数
nlinbd <- matrix(0, nr=6, nc=2)#设置非线性约束个数

## 线性约束
linbd[1,] <- c(0,0) # h1
linbd[2,] <- c(-35.82, Inf) # g1
linbd[3,] <- c(133, Inf) # g2
linbd[4,] <- c(35.82,Inf) # g3
linbd[5,] <- c(-133, Inf) # g4
```

```
# 非线性约束1
h2 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4];
  x5 <- par[5]; x6 <- par[6]; x7 <- par[7]; x8 <- par[8];
  x9 <- par[9]; x10 <- par[10]
  98000*x3/(x4*x9+1000*x3)-x6
}
nlinbd[1,] <- c(0,0)
# 非线性约束2
h3 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4];
  x5 <- par[5]; x6 <- par[6]; x7 <- par[7]; x8 <- par[8];
  x9 <- par[9]; x10 <- par[10]
  (x2+x5)/x1 - x8
}
nlinbd[2,] <- c(0,0)

# 非线性约束3
g5 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4];
  x5 <- par[5]; x6 <- par[6]; x7 <- par[7]; x8 <- par[8];
  x9 <- par[9]; x10 <- par[10]
  1.12*x1 + 0.13167*x1*x8 - 0.00667*x1*x8^2 - a*x4
}
nlinbd[3,] <- c(0,Inf)
# 非线性约束4
g6 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4];
  x5 <- par[5]; x6 <- par[6]; x7 <- par[7]; x8 <- par[8];
  x9 <- par[9]; x10 <- par[10]
  57.425 + 1.098*x8 - 0.038*x8^2 + 0.325*x6 - a*x7
}
```

```
nlinbd[4,] <- c(0,Inf)
# 非线性约束5
g7 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4];
  x5 <- par[5]; x6 <- par[6]; x7 <- par[7]; x8 <- par[8];
  x9 <- par[9]; x10 <- par[10]
  -g5(par) + (1/a-a)*x4
}
nlinbd[5,] <- c(0,Inf)
# 非线性约束6
g8 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4];
  x5 <- par[5]; x6 <- par[6]; x7 <- par[7]; x8 <- par[8];
  x9 <- par[9]; x10 <- par[10]
  -g6(par) + (1/a-a)*x7
}
nlinbd[6,] <- c(0,Inf)

#设置A, 5个线性约束 X 10个参数, 所以A是5X10矩阵
A <- rbind(c(-1, 0, 0, 1.22,-1, 0, 0, 0, 0, 0), #h1
           c( 0, 0, 0, 0, 0, 0, 0, 0, -b, -0.222), #g1
           c( 0, 0, 0, 0, 0, 0, 3, 0, 0, -a), #g2
           c( 0, 0, 0, 0, 0, 0, 0, 0,(1/b-b)+b, 0.222), #g3
           c( 0, 0, 0, 0, 0, 0, -3, 0, 0,(1/a-a)+a)) #g4

#初始值
p0 <- c(1745, 12e3, 11e1, 3048, 1974, 89.2, 92.8, 8, 3.6, 145)




ret <- donlp2(par=p0, fn=fn,
```

```
                par.u=par.u, par.l=par.l,
                A=A,
                lin.u=linbd[,2], lin.l=linbd[,1],
                nlin=list(h2,h3,g5,g6,g7,g8),
                nlin.upper=nlinbd[,2], nlin.lower=nlinbd[,1])
ret$par
```

输出结果

```
>ret$par
 [1]  1698.094765 15818.614889    54.102682
 [4]  3031.225217  2000.000000    90.115422
 [7]    95.000000    10.493298     1.561636
[10]   153.535354
```

编程时需要注意的是：编写目标函数和约束函数时，每一个函数内的参数都进行了赋值，如fn和h2时，x1...x10都进行了赋值处理。

上面例子是文档自带的例子。下面我们将非线性优化应用到统计模型中。

14.4.2　经验似然筛选方法

随着科学技术的发展，超高维数据越来越多出现在遗传、基因芯片、磁共振成像等领域。一般来说，超高维数据是指变量个数远远大于样本量的数据。例如，我们研究基因对白血病的影响。由于受到经费、被试者人数等影响，样本量往往很小，而变量个数非常大，如200个被试者，2000个基因。通常假设$p = O(\exp(n^{\kappa})), 0 < \kappa < 1/2$。处理此类问题时，我们通常进行“稀疏性”假定，即假定只有很少的自变量对于因变量产生影响，也就说自变量系数为零的很多，非零的很少。这种假定具有一定的合理性，影响某一个事物的因素也许有很多个，但是起主要作用也许只有少数几个或很少的几个因素。稀疏性假定是我们处理超高维（高维）问题的基本假定。由于计算成本的原因，当前比较流行的处理高维数据方法已不太适合处理超高维的数

据，如LASSO、SCAD和MCP等方法。为此，Fan 和Lv (2008) 基于Pearson 相关系数提出SIS (Sure Independent Screening)。SIS 虽然不像LASSO、SCAD和MCP 那样利用罚函数一次性选择变量，但它可以方便快捷的筛选变量，即通过简单排序筛选变量。SIS筛选出来的变量比较多，它可以确保那些对因变量有影响的自变量全被选出，这也是被称为确保(Sure) 的原因。超高维变量选择主要是两步法思想：第一步是通过某一个规则初步筛选变量；第二步是利用已有的方法(LASSO、SCAD 和MCP 等)对第一步筛选出来的变量再进行变量选择。对于超高维变量选择，第一步非常重要，第二步只是利用已有的方法。SIS自2008年提出，目前已从线性模型推广到广义线性模型、可加模型和变系数模型等。R包主要有`SIS`包(Fan等2015)和`QCSIS`包(Ma等2015)可以实现线性模型、广义线性模型和模型释放的超高维变量筛选方法。

Chang等(2013)基于经验似然提出一种超高维数据的变量筛选方法，即经验似然筛选方法(Empirical Likelihood-Sure Independence Screening)，简称EL-SIS。设$\boldsymbol{X}=(X_1,X_2,\ldots,X_p)^\top$ 是p 维自变量向量，$\boldsymbol{\beta}=(\beta_1,\beta_2,\ldots,\beta_p)^\top$，$Y$是因变量。$\{(\boldsymbol{X}_i,Y_i)\}_{i=1}^n$是随机样本。假设$\mathrm{E}(X_j)=0$和$\mathrm{E}(X_j^2)=1,j=1,2,\ldots,p$。原假设$H_0:\beta_j=0$，则似然比的定义是：

$$l_j(0)=-2\log\{EL_j(0)\}-2n\log(n)=2\sum_{i=1}^{n}\log(1+\lambda X_{ij}Y_i) \tag{14.1}$$

其中$EL_j(0)$是H_0下的经验似然，λ满足下面条件

$$\sum_{i=1}^{n}\frac{X_{ij}Y_i}{1+\lambda X_{ij}Y_i}=0 \tag{14.2}$$

EL-SIS是取似然比取值最靠前的d 个，即

$$\widehat{\mathcal{M}}=\{1\le j\le p:\widehat{l_j(0)}\text{ 排在最靠前的}d\}$$

d 经常取$[n/\log(n)]$ 或$n-1$。

下面我们将EL-SIS转化为非线性约束的非线性优化：经验似然$EL_j(0)$是取最大值，也就是说求$l_j(0)$的最小值

$$\min_{\lambda}2\sum_{i=1}^{n}\log(1+\lambda X_{ij}Y_i)$$

$$s.t. \sum_{i=1}^{n} \frac{X_{ij}Y_i}{1+\lambda X_{ij}Y_i} = 0$$

下面以超高维文章中经常使用的一个例子阐述EL-SIS的编程。其中我们在编写ELSIS函数时，我们先编写了一个“草稿”，然后在编写完整的函数。详见“运行一次”。

R 代码

```
rm(list=ls(all=TRUE))#清空所有对象
library(QCSIS)#加载QCSIS包
library(Rdonlp2)#加载Rdonlp2包
#产生数据开始
n <- 50
p <- 500
x <- matrix(rnorm(n * p), n, p)
e <- rnorm(n, 0, 1)
y <- 5 * x[, 1] + 5 * x[, 2] + 5 * x[, 3] + e
#产生数据结束
d <- floor(n/log(n))
#QC-SIS
fit.QCSIS <- QCSIS(x = x, y = y, d = d)
fit.QCSIS$M


#运行一次开始
nlin.l <- 0
nlin.u <- 0
par.l <- 0
par.u <- Inf
j=5

XjY <- x[,j] * y
fn <- function(lambda){
```

```
  sum(2*(1 + lambda*XjY))
}

#constraint function
nlcon=function(lambda){
  sum(XjY/(1+lambda*XjY))
}
lambda0 <- n^(-0.5)
fn(lambda0)
nlcon(lambda0)
ret <- donlp2(par=lambda0, fn =fn, par.u=par.u,par.l=par.l,
              nlin=list(nlcon), nlin.u=nlin.u, nlin.l=nlin.l)
ret$par
ret$fx
#运行一次结束

#EL-SIS 开始
ELSIS <- function(x, y, d){
  p <- dim(x)[2]
  n <- dim(x)[1]
  x <- scale(x)#标准化
  EL <- NULL
  nlin.l <- 0
  nlin.u <- 0
  par.l <- 0
  par.u <- Inf
  lambda0 <- n^(-0.5)#根据Chen和Van(2009)
  for(j in 1: p ){
    XjY <- x[,j] * y
    fn <- function(lambda){
      sum(2*(1 + lambda*XjY))
    }
```

```
    nlcon=function(lambda){
      sum(XjY/(1+lambda*XjY))
    }
    #求解
    ret <- donlp2(par=lambda0, fn =fn, par.u=par.u,par.l=par.l,
                  nlin=list(nlcon), nlin.u=nlin.u, nlin.l=nlin.l)
    EL[j] <-  ret$fx #掉lj(0)

  }
  w.el <- EL
  M.el <- order(w.el, decreasing = T)[1:d]
  return(list(w = w.el, M = M.el))

}
#EL-SIS结束

fit.ELSIS <- ELSIS(x = x, y = y, d = d)
fit.ELSIS$M

#比较两种结果
fit.QCSIS$M
fit.ELSIS$M
```

输出结果

```
> #比较两种结果
> fit.QCSIS$M
 [1]   2   1   3 340 131 190 283 250 159  82 443 411
> fit.ELSIS$M
 [1]   1   2 190 443 283  87 307   3 206 458 159 246
```

从输出结果，我们可以看出编写的ELSIS函数的有效性(注意：由于数据产生不同，所以读者运行的结果和上面的结果会不同)。其实EL-SIS中有一个假设：如果0 属于$\{X_{ij}Y_i\}_{i=1}^n$ 的凸包(Convex Hull)，令$\widehat{l_j(0)} = \infty$。这个问题可以转化为判断一个非线性方程有没有解：即$\sum_{i=1}^n t_i X_{ij} Y_i = 0, 0 \leq t_i \leq 1$ 是否存在解t_i。另外，Chang 等(2016) 提出的局部经验似然比筛选方法。该方法也可以转化为非线性约束的非线性优化。读者可以尝试编写它们的代码。

14.5 参考文献

1. Chang J., Tang C., Wu Y. (2013). Marginal empirical likelihood and sure independence feature screening. Annals of Statistics, 41(4), 2123–2148.

2. Chang, J., Tang, C. , Wu, Y. (2016). Local independence feature screening for nonparametric and semiparametric models by marginal empirical likelihood. Annals of Statistics, 在线.

3. Chen X., Yin X. (2015). NlcOptim. R package version 0.3, `http://CRAN.R-project.org/package=NlcOptim`.

4. Chen, S. , Van K. , (2009). A review on empirical likelihood methods for regression. Test, 18(3), 415-447.

5. Fan J., Lv J. (2008). Sure independence screening for ultrahigh dimensional feature space. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 70(5), 849-911.

6. Fan J.等(2015). SIS. R package version 0.7–6, `http://CRAN.R-project.org/package=SIS`.

7. Ma X., Zhang J., Zhou J. (2015). QCSIS. R package version 0.1, `http://CRAN.R-project.org/package=QCSIS`.

8. Tamura R. (2007). Rdonlp2 - an R interface to DONLP2. `http://svitsrv25.epfl.ch/R-doc/library/Rdonlp2/doc/tutorial.pdf`

第四部分

常见算法

第15章　非线性规划

这一部分介绍梯度下降法及其衍生的方法。主要参考的是非线性规划(第二版, Dimitri P. Bertsekas)第一章的内容。

15.1　基本的概念

最优化问题数学模型一般可以用约束集X 和目标函数f表示。集合X包含对于所有可能的x的集合，函数f将X的元素映射到实数集上。假设$x = (x_1, \ldots, x_n)^\top$是未知参数向量，约束集$X$是$n$维欧氏空间$R^n$的子集。这一章考虑是无约束的非线性规划，即

$$\begin{aligned} &\min f(x) \\ &s.t.\ \ x \in R^n \end{aligned} \tag{15.1}$$

局部最小值点和全局最大值点：

- x^*是局部最小值，如果存在$\varepsilon > 0$，

$$f(x^*) \leq f(x)\ \ \forall x \text{ 使得 } \|x - x^*\| < \varepsilon$$

其中$\|\cdot\|$是欧式范数。

- x^*是全局最小值，如果

$$f(x^*) \leq f(x)\ \ \forall x \in R^n$$

如果上面=不成立，x^*称为严格的。

定义15.1.1 (凸集和凸函数)

- R^n的子集C被称作是凸的*(Convex)*，如果

$$\alpha x + (1 - \alpha y) \in C\ \ \forall x, y \in C\ \ \forall \alpha \in [0, 1]$$

- C是凸集，函数$f : C \to R$称为凸的，如果

$$f(\alpha x + (1 - \alpha y)) \leq \alpha f(x) + (1 - \alpha) f(y)\ \ \forall x, y \in C\ \ \forall \alpha \in [0, 1]$$

命题15.1.2 (凸函数判断) *(a)* 线性函数是凸的

(b) 任意的向量范数是凸的

(c) 凸函数的加权和(权重为正数)是凸的

(d) 如果I是一个指数集合，C是R^n的凸子集，$f_i : C \to R$对于每一个$i \in I$都是凸函数，则定义在$C \to (\infty, \infty]$

$$h(x) = \sup_{i \in I} f_i(x)$$

是凸函数。

命题15.1.3 (最优性必要条件) 令x^*为$f : R^n \to R$的无约束局部最小值，并且假设f在开集S上连续可微，$x^* \in S$，则有

$$\nabla f(x^*) = 0 \quad \text{(一阶必要条件)}$$

如果f在S中是二阶连续可微的，则有

$$\nabla^2 f(x^*) \text{ 半正定} \quad \text{(一阶必要条件)}$$

命题15.1.4 (二阶最优性充分条件) 令$f : X \to R$为开集X上二阶可微函数，假设向量$x^* \in S$，满足

$$\nabla f(x^*) = 0 \quad \nabla^2 f(x^*) \text{ 正定}$$

那么x^*为f的严格无约束局部最小值。

命题15.1.5 (凸目标函数) 令$f : X \to R$为开集X上一个凸函数，则

(a) 在X上的f的局部最小值也是X的全局最小值。如果f是严格凸的，那么至少存在一个全局最小值。

(b) 如果X是开集，那么x^*是f在X的全局最优解的充要条件是$\nabla f(x^*) = 0$.

$$\nabla f(x^*) = 0 \quad \nabla^2 f(x^*) \text{ 正定}$$

那么x^*为f的严格无约束局部最小值。

灵敏度：分析当一个参数向量变化是最优解如何变化。

15.2　梯度法

给定一个向量$x \in \Re^n$，满足$\nabla f \neq 0$，简单梯度法的形式是：

$$x_\alpha = x - \alpha \nabla f(x), \quad \forall \alpha \geq 0 \tag{15.2}$$

下面讨论称为下降法的原因，即想法说明$f(x_\alpha) \leq f(x)$. 利用Taloy展开式和上面等式，

$$\begin{aligned} f(x_\alpha) &= f(x) + \nabla f^\top(x)(x_\alpha - x) + o(\|x_\alpha - x\|) \\ &= f(x) - \alpha \|\nabla f(x)\|^2 + o(\alpha \|\nabla f(x)\|) \\ &= f(x) - \alpha \|\nabla f(x)\|^2 + o(\alpha) \end{aligned}$$

当α接近零时，$\alpha\|\nabla f(x)\|^2$远大于$o(\alpha)$，所以对于一个充分小的正数α，$f(x_\alpha) \leq f(x)$.

进一步讨论：

$$x_\alpha = x + \alpha d \quad \forall \alpha \geq 0 \tag{15.3}$$

其中$d \in \Re^n$是一个n维向量。

我们使用泰勒展开

$$\begin{aligned} f(x_\alpha) &= f(x) + \nabla f^\top(x)(x_\alpha - x) + o(\|x_\alpha - x\|) \\ &= f(x) + \alpha \nabla f^\top(x) d + o(\alpha) \end{aligned}$$

要保证$f(x_\alpha) \leq f(x)$，$\nabla f(x)^\top d < 0$。从而得到方向d必须满足

$$\nabla f(x)^\top d \tag{15.4}$$

下面进一步讨论一般的迭代：

$$x^{k+1} = x^k + \alpha^k d^k, \ \ k = 0, 1, \ldots, \tag{15.5}$$

如果$\nabla f(x^k) \neq 0$,选择方向d^k，满足

$$\nabla f(x^k)^\top d^k < 0. \tag{15.6}$$

并且选择步长α^k为正数。

- $\nabla f(x^k) = 0$,迭代停止，即$x^{k+1} = x^k$
- $\nabla f(x^k) \neq 0$,迭代停止，即x^{k+1}按照d^k方向搜索，考虑到d^k和$\nabla f(x^k)$的关系，我们将这一类方法成为梯度下降法

15.2.1 梯度法的一般形式

下面，我们将(15.5)推广，其一般的表达式：

$$x^{k+1} = x^k - \alpha^k D^k \nabla f(x^k) \tag{15.7}$$

其中D^k是正定对称矩阵，这是因为下降条件是

$$\nabla f(x^k)^\top D^k \nabla f(x^k) > 0$$

这要求D^k是正定。

15.2.2 几种D^k

- **最速下降法**

$$D^k = I$$

其中I是$n \times n$的单位矩阵。缺点：收敛速度减慢。这个名称由于

$$d^k = -\frac{\nabla f(x^k)}{\|\nabla f(x^k)\|}$$

- **牛顿法**

$$D^k = (\nabla^2 f(x^k))^{-1}$$

其中$\nabla^2 f(x^k)$是正定，如果不是正定，使用$\nabla^2 f(x^k) + \Delta^k$，$\Delta^k$是一个对角矩阵。

15.2.3 α^k的选择

α^k的选择没有最优的准则，下面给出几种常见的准则：

- **最小化准则**

$$f(x^k + \alpha^k d^k) = \min_{\alpha \geq 0} f(x^k + \alpha^k d^k)$$

缺点：α没有范围，实际计算不易操作。

- **有限最小化准则**

$$f(x^k + \alpha^k d^k) = \min_{\alpha \in [0,s]} f(x^k + \alpha^k d^k)$$

前面这个准则需要一维搜索。

- **固定步长准则**

$$\alpha^k = s$$

简单但如果步长过大会出现$\{x^k\}$发散，太短，收敛慢。

15.2.4　终止条件

- $\|\nabla f(x^k)\| \leq \epsilon$，其中$\epsilon$是一个充分小的数。缺点：有量纲。
- $$\frac{\|\nabla f(x^k)\|}{\|\nabla f(x^0)\|} \leq \epsilon$$
- $$\|x^{k+1} - x^k\| \leq \epsilon \ \ (or)\epsilon\|x^k\|$$
- $$f(x^k) - f(x^{k+1}) \leq \epsilon$$

15.2.5　收敛速率

常用的三种方式：

- **计算复杂度方法**：尝试估计给定算法达到最优解所需要的初等运算次数。缺点：这种方法给出是”最坏情况“下的估计，其结果和初始值到最优解的距离有关。
- **信息复杂度方法**：计算单个函数值或者其梯度所需要的次数比较困难，可以通过精确最优解或者近似最优解随需要函数值或者梯度值的计算次数。
- **局部芬妮下方法**：关注算法在最优解领域内的局部表现，通过Taloy展开局部分析接近最优解时算法的性能。缺点：没有考虑离最优解比较远时算法的性能。

前两种方法都是体现最差情况的估计值讨论，导致它们理论模型与实际之间存在很大差异。比如单纯性在最坏的情况很差，而这种情况几乎极少出现。另外，椭圆法在最坏的时候比单纯性，但这个方法实际表现很不理想。

15.3 增量梯度法

假设$f(x)$可以分成m个数据块，即

$$\min f(x)=\frac{1}{2}\|g(x)\|^2=\frac{1}{2}\sum_{i=1}^{m}\|g_i(x)\|^2 \tag{15.8}$$

$$s.t.\ x\in\Re^n \tag{15.9}$$

其中$g=(g_1,\ldots,g_m)^\top$.

前面的方法使得迭代成本比较高，我们希望更新时不需要处理全部的数据集合。增量梯度法处理x^k到x^{k+1}分成m步，假设

$$\psi_0=x^k$$

其包括如下m步

$$\begin{aligned}
\psi_1&=\psi_0-\alpha^k\nabla g_1(\psi_0)g_1(\psi_0)\\
\psi_2&=\psi_1-\alpha^k\nabla g_2(\psi_1)g_2(\psi_1)\\
&\cdots\\
\psi_i&=\psi_{i-1}-\alpha^k\nabla g_i(\psi_{i-1})g_i(\psi_{i-1})\\
&\cdots\\
\psi_m&=\psi_{m-1}-\alpha^k\nabla g_m(\psi_{m-1})g_m(\psi_{m-1})
\end{aligned}$$

其中步长是α^k,方向采用的是第i个数据块的梯度。$\nabla g_i(\cdot)$是$n\times 1$向量，$g_i(\cdot)$是一个数。

$$\nabla(\frac{1}{2}\|g_i(x)\|^2)|_{x=\psi_{i-1}}=\nabla g_i(\psi_{i-1})g_i(\psi_{i-1})$$

这种方法可以写成

$$x^{k+1}=x^k-\alpha^k\sum_{i=1}^{m}\nabla g_i(\psi_{i-1})g_i(\psi_{i-1}) \tag{15.10}$$

我们可以看出此时使用的方向是

$$\sum_{i=1}^{m}\nabla g_i(\psi_{i-1})g_i(\psi_{i-1}).$$

下面我举一个例子：最小二乘问题，假设$x \in \Re$

$$\min f(x) = \frac{1}{2}\sum_{i=1}^{n}(a_i x - b_i)^2$$

其中a_i和b_i是已知的数据。每一块$f_i = \frac{1}{2}(a_i x - b_i)^2$，则

$$\nabla f_i(x) = a_i(a_i x - b_i)$$

进而得到迭代

$$\psi_i = \psi_{i-1} - \alpha^k \sum_{i=1}^{n} a_i(a_i \psi_{i-1} - b_i)$$

15.4　共轭方向法

一开始，共轭方向法提出的目的希望加快最速下降法的收敛速度，避免牛顿法大的计算量，最初是为了解决二次问题，即$\min f(x) = \frac{1}{2}x^\top Q x - b^\top x$.下面讨论非线性的问题。

$$x^{k+1} = x^k + \alpha^k d^k$$

其中α^k采用最小化准则得到

$$f(x^k + \alpha^k d^k) = \min_{\alpha} f(x^k + \alpha^k d^k)$$

且

$$d^k = -\nabla f(x^k) + \beta^k d^{k-1}$$

常见计算β^k方式是：

$$\beta^k = \frac{\nabla f(x^k)^\top (\nabla f(x^k) - \nabla f(x^{k-1}))}{\nabla f(x^k)^\top \nabla f(x^{k-1})}$$

或者

$$\beta^k = \frac{\nabla f(x^k)^\top \nabla f(x^k)}{\nabla f(x^k)^\top \nabla f(x^{k-1})}$$

共轭梯度法经常用于参数很大的问题。这种方法经常开始几次迭代之后生成一些没有意义且效率低下的搜素方向。所以，我们经常采用循环的方法使用共轭方向。在循环第一步使用最速下降法进行搜索。

15.5 坐标下降法

如果目标函数不可导，我们可以使用坐标下降法。该方法每次迭代沿着一个坐标分量方向最小。这不仅简化了搜索方向的计算，而且使得步长选择简单一些，因为沿着分量方向的最小化相对容易求解。

给定x^k，x^{k+1}的第i个分量有下面式子得到

$$x_i^{k+1} \in \mathrm{a}rgmin_{\xi \in R} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, \xi, x_{i+1}^k, \ldots, x_n^k)$$

这种方法适合x^i有上下界的f的最小化。这种方法可以进行并行计算。实际问题中，这种方法比较高效。