统计学中的优化以及R实现

马学俊

献给我的家人、恩师和所有在学术道路上帮助我的人

蓦然回首

谈起优化，统计人无人不知，无人不晓。在经典的统计学中，我们通常力求得到一个“完美”的解。这个解我们当时认为完美无瑕，然而当面对繁杂的世界时，我们依然心惊胆战。因为，我们不知道这样的“完美”会持续多久。刚接触统计，记得那是最小二乘的天下，或者其兄弟加权最小二乘的天下。后来被告知，天下原本不是上述两个兄弟，是优化大帝的，犹如醍醐灌顶，梦中惊醒。遥想原来与君有过一面之缘（研究生必修课）只是未钟情；而今喜相逢，一壶浊酒，笑看秋月春风。缘分呀！说来惭愧，对于大帝不知，犹如有眼不识泰山。谈起优化，其繁体字是優化。看来不是好糊弄之辈，此人“心”在中间，单“人”靠边站，犹如包拯之无私，李逵之耿直。于是，不能再溜须拍马敷衍了事了！吾只能双耳不闻窗外事，上下而求索了。

原打算写一本“老少”皆宜的读物。但优化大帝犹如道，神龙见首不见尾。对于我这种刚修行之人，优化大帝近在咫尺却遥不可及呀。只能怀着忐忑的心情，记录一些拜访优化大帝的感受。书名原定《统计学中的优化之他说》。之所以取名《他说》，是因为这些都是作者的一些“道听途说”。如有不妥，请大家不吝赐教，贵在参与。如果问题，麻烦联系鄙人(yinuoyumi@163.com)，感激涕零。

马学俊

2016年1月

于言蹊堂

前言

随着统计学的发展，统计计算越来越重要，尤其是大数据时代。优化是统计计算的重要组成部分。很多统计方法或模型的实现都依赖它，如分位数回归、复合分位数回归、广义线性回归和众数回归等。

R软件是一个非常强大的统计软件。由于其灵活性和易操作性，它目前几乎超越了SAS和SPSS等统计软件。它不仅仅是统计方法实现的软件，更是学习统计的有利工具。您可以通过“?命令”找到研究的参考文献，如安装quantreg，然后运行?rq；您可以查看分位数回归研究的文献；您可以通过http://cran.stat.sfu.ca 查看最新捐赠的R包，从而学习新的统计方法；您可以查看统计模型或方法实现的R包摘要。如http://cran.at.r-project.org/web/views/Optimization.html总结了R中的优化包；您还可以通过http://www.r-bloggers.com看到R使用的技巧等。

本书主要讲解优化在R中实现，阐述优化在统计学中的应用。书的素材有些来自于R包的帮组文档，有些来自于作者研究中遇到的问题。全书主要解决以下优化问题(暂定，因为内容在更新)：

- 线性规划:

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

  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$\boldsymbol{w} \in \mathbb{R}^{m}$。

- 二次规划:

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

  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$\boldsymbol{b} \in \mathbb{R}^{n}$。

- 非线性方程：

$$f(\boldsymbol{x}) = \mathbf{0}$$

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

- 非线性优化：

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

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。

本书是学习统计中的优化及R实现的入门教程，可以作为统计学专业的课外读物。限于作者水平，不妥或谬误之处在所难免，恳请大家赐教。非常感谢!

马学俊
2016年2月
于言蹊堂

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

第1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。

1.1　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。

1.2　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

1.3　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。

1.3.1 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

$$\sum_i^n \rho_\tau(Y_i - \boldsymbol{X}_i^\top\boldsymbol{\beta}) \tag{1.1}$$

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(1.1)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(1.1)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(1.1)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)。

1.3.2　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：

$$(\hat{b}_1, \ldots, \hat{b}_K, \hat{\boldsymbol{\beta}}) = \arg\min \sum_{k=1}^{K} \Big\{ \sum_{i=1}^{n} \rho_{\tau_k}(Y_i - b_k - \boldsymbol{X}_i^\top \boldsymbol{\beta}) \Big\} \tag{1.2}$$

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1.2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

$$KY = \Delta\boldsymbol{\Theta} \tag{1.3}$$

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。

1.4　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2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。

2.1　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$\boldsymbol{b} \in \mathbb{R}^n$。

2.2　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。

2.3　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。

2.4 应用

2.4.1 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
```

2.5　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3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)。

3.1　一般表达式

解决非线性一般的目的是求解

$$F(\boldsymbol{x}) = \mathbf{0} \tag{3.1}$$

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

3.2　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。

3.2.1 引言

下面从牛顿法开始，简单介绍求解3.1三种简单方法。

- 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。

3.2.2　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：

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k \;\; k = 0, 1, 2, \ldots \tag{3.2}$$

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`。

$$\alpha_k = \frac{\boldsymbol{s}_{k-1}^\top \boldsymbol{s}_{k-1}}{\boldsymbol{s}_{k-1}^\top \boldsymbol{y}_{k-1}} \tag{3.3}$$

  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`。

$$\alpha_k = \frac{\boldsymbol{s}_{k-1}^\top \boldsymbol{y}_{k-1}}{\boldsymbol{y}_{k-1}^\top \boldsymbol{y}_{k-1}} \tag{3.4}$$

- 第三种：简单记为`sane=3`和`dfsane=3`。

$$\alpha_k = sgn(\boldsymbol{s}_{k-1}^\top \boldsymbol{y}_{k-1}) \frac{\|\boldsymbol{s}_{k-1}\|}{\|\boldsymbol{y}_{k-1}\|} \tag{3.5}$$

  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。

3.2.3 SANE 和DF-SANE之全局收敛

为了全局收敛，3.2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：

$$f(\boldsymbol{x}_{k+1}) \leq \max_{0\leq j\leq M} f(\boldsymbol{x}_{k-j}) + \gamma\alpha_k \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{d}_k \tag{3.6}$$

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：

$$f(\boldsymbol{x}_{k+1}) \leq \max_{0\leq j\leq M} f(\boldsymbol{x}_{k-j}) + \eta_k - \gamma\alpha_k^2 f(\boldsymbol{x}_k) \tag{3.7}$$

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。

3.2.4 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3.3式，第二种是`method=2`对应3.4式，第三种是`method=3` 对应3.5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$M > 100$。

3.2.5 *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
```

3.2.6 *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。

3.3 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。

3.3.1 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)。

3.4 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4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。

4.1　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。

4.2　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`。

4.3 应用

4.3.1 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。

$$Q_n(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h_n} K\Big(\frac{Y_i - \boldsymbol{X}_i^\top \boldsymbol{\beta}}{h_n}\Big) \tag{4.1}$$

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

下面我们使用BB包求解4.1，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(4.1)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。

4.4　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5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。

5.1　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。

5.2 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)} )
```

5.2.1 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。

5.2.2 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
```

5.2.3　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)
```

5.2.4　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
```

5.2.5 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,.....)
```

5.2.6 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。

5.2.7　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

5.2.8　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

5.3　另一种表达式

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

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。

5.4 应用

5.4.1 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。

5.4.2　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：

$$l_j(0)=-2\log\{EL_j(0)\}-2n\log(n)=2\sum_{i=1}^{n}\log(1+\lambda X_{ij}Y_i) \tag{5.1}$$

其中$EL_j(0)$是H_0下的经验似然，λ满足下面条件

$$\sum_{i=1}^{n}\frac{X_{ij}Y_i}{1+\lambda X_{ij}Y_i}=0 \tag{5.2}$$

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。

5.5 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`