非线性规划与 R 语言

黄湘云

2016 年 8 月 26 日

目录

1 引言 **3**

2 非线性方程组解法 **3**
2.1 两个几乎错误的解法 3
2.1.1 古典迭代解法 3
2.1.2 不动点迭代 4
2.2 Newton 和拟 Newton 迭代 5
2.3 Barzilai-Borwein 算法 6

3 隐函数极值与非线性方程组 **6**
3.1 rootSolve 包 7
3.2 BB 包 8
3.3 nleqslv 包 8
3.4 定初始值 9

4 隐函数极值与非线性规划 **10**
4.1 alabama 包 10
4.2 Rdonlp2 包 11
4.3 Rsolnp 包 11

5 非线性规划 **12**
5.1 非线性最小二乘与无约束优化 12
5.2 复杂非线性回归 13
5.3 一维优化问题 14
5.4 箱式约束 16
5.4.1 optim 函数 16
5.4.2 nlminb 函数 17
5.4.3 BB 包 17
5.4.4 Rcgmin 包 18
5.4.5 Rvmmin 包 19
5.4.6 dfoptim 包 20
5.5 线性约束 21
5.5.1 constrOptim 函数 21

5.6 非线性约束 22
5.6.1 NlcOptim 包 22
5.6.2 Rnlminb2 包 25
5.7 凸优化 26
5.8 应用 29
5.8.1 正态分布参数估计 29
5.8.2 二项泊松混合分布参数估计 30
5.9 Scilab 接口 36

6 启发式算法 37

6.1 随机数生成 37

7 R 软件信息 39

1 引言

在 R 帮助系统，相关程序手册和 JSS 论文的基础上，我踩了几个 R 包里的坑，分享一些使用经验，对相关的 R 包做了归类整理，供读者学习参考！文章以求解一个隐函数的极值为例，介绍非线性方程组的解法，这里只是冰山一角，相关 R 包的手册有求解更加复杂的非线性方程组案例，更多的内容请读者去探索了！

2 非线性方程组解法

下面考虑如何数值求解含 n 个未知变量、n 个方程的非线性方程组

$$F(\boldsymbol{x}) = \begin{cases} f_1(x_1, x_2, ..., x_n) = 0 \\ f_2(x_1, x_2, ..., x_n) = 0 \\ \quad\cdots\cdots\cdots\cdots\cdots\cdots \\ f_n(x_1, x_2, ..., x_n) = 0 \end{cases}$$

其中 n 大于 1 的正整数，每个 $f_i(x_1, x_2, ..., x_n)(i = 1, 2, \cdots, n)$ 为 $\mathbb{R}^n \to \mathbb{R}$ 的 n 元实值函数，且至少一个是非线性函数。

2.1 两个几乎错误的解法

这两个解法来自文献 [1]，只在极其特殊的情况下才能使用，而合适的例子只有充分发挥数学家的想象力去造了！没有任何实用价值，在很多文献里也看不到此等解法，因此说这是几乎错误的解法！可见看书，也不要被书给骗了！

2.1.1 古典迭代解法

非线性 Jacobi 迭代、Gauss-Seidel 迭代和 SOR 迭代都是从线性方程组迭代解法推广而来的。它们之间很类似，下面以非线性 Jacobi 迭代为例，伪代码如下：

for $k \leftarrow 0, 1, 2, ...$ **do**
    for $i \leftarrow 1, 2, ..., n$ **do**
        用非线性方程的求解器解
        $f_i(x_1^{(k)}, ..., x_{i-1}^{(k)}, u, x_{i+1}^{(k)}, ..., x_n^{(k)})$ 解得 u
        $x_i^{(k+1)} \leftarrow u$
    end for
    如果迭代停止条件满足，则终止循环并输出 $x^{(k+1)}$
end for

用非线性 Jacobi 迭代法求解方程组

$$\begin{cases} (x_1 + 3)(x_2^3 - 7) + 18 = 0 \\ \sin(x_2 e^{x_1} - 1) = 0 \end{cases}$$

取初值 $x_0 = (-0.5, 1.4)'$，已知精确解 $x^\star = (0, 1)'$

```
f1 <- function(x2) {
    temf1 <- function(x) {
        (x + 3) * (x2^3 - 7) + 18
    }
    return(temf1)
}
f2 <- function(x1) {
    temf2 <- function(x) {
        sin(x * exp(x1) - 1)
    }
    return(temf2)
}
Jacobi_iter <- function(x0 = c(-0.5, 1.4), maxiter = 100, eps = 1e-10) {
    N = maxiter
    x2 <- x1 <- rep(0, N)
    x1[1] = x0[1]
    x2[1] = x0[2]
    for (i in seq(N)) {
        x1[i + 1] <- uniroot(f1(x2[i]), interval = c(-1, 1))$root
        x2[i + 1] <- uniroot(f2(x1[i]), interval = c(0, 2))$root
        err <- sqrt((x1[i + 1] - x1[i])^2 + (x2[i + 1] - x2[i])^2)
        if (err <= eps)
            return(list(vec = cbind(x1, x2)[1:(i + 1), ], eqs.root = c(x1[i +
                1], x2[i + 1]), n.iter = i + 1))
    }
}
solution <- Jacobi_iter(x0 = c(-0.5, 1.4), maxiter = 10, eps = 1e-05)
```

这个方法看起来可行，实际却难以操作！因为不能确定两组点列是否收敛，迭代过程中 x_1 和 x_2 的有根区间和初始值 x_0。

2.1.2 不动点迭代

不动点迭代是从非线性方程求根推广而来的解法。试用此法求解下列非线性方程组

$$\begin{cases} x_1 = -\dfrac{\cos x_1}{81} + \dfrac{x_2^2}{9} + \dfrac{\sin x_3}{3} \\ x_2 = \dfrac{\sin x_1}{3} + \dfrac{\cos x_3}{3} \\ x_3 = -\dfrac{\cos x_1}{9} + \dfrac{x_2}{3} + \dfrac{\sin x_3}{6} \end{cases}$$

精确解为 $(0, \frac{1}{3}, 0)$

```
fixed_iter <- function(x0 = rep(0, 3), maxiter = 100, eps = 1e-10) {
    N = maxiter
```

```r
    x2 <- x3 <- x1 <- rep(0, N)
    x1[1] = x0[1]
    x2[1] = x0[2]
    x3[1] = x0[3]
    for (i in seq(N)) {
        x1[i + 1] <- -cos(x1[i])/81 + x2[i]^2/9 + sin(x3[i])/3
        x2[i + 1] <- sin(x1[i])/3 + cos(x3[i])/3
        x3[i + 1] <- -cos(x1[i])/9 + x2[i]/3 + sin(x3[i])/6
        err <- sqrt((x1[i + 1] - x1[i])^2 + (x2[i + 1] - x2[i])^2 + (x3[i +
            1] - x3[i])^2)
        if (err <= eps)
            return(list(vec = cbind(x1, x2, x3)[1:(i + 1), ], eqs.root = c(x1[i +
                1], x2[i + 1], x3[i + 1]), n.iter = i + 1))
    }
}
(solution <- fixed_iter(x0 = rep(1, 3), maxiter = 50, eps = 1e-05))
```

方程组的数值解为 $(3.3122945 \times 10^{-6}, 0.333336, 3.5400238 \times 10^{-6})$，与精确解已经相差无几！这是很凑巧的情况，实际上确定不动点迭代格式是很难的一件事（要保证迭代点列收敛）。

2.2 Newton 和拟 Newton 迭代

Newton 迭代 (又称 Newton-Raphson 迭代) 是从非线性方程求根的 Newton 迭代推广过来的！即

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, i = 0, 1, \cdots$$

推广至

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^i - \boldsymbol{A}_i^{-1} \boldsymbol{F}(x^i), i = 0, 1, \cdots$$

其中

$$\boldsymbol{A}_i = \boldsymbol{F}'(\boldsymbol{x}^i) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1^i} & \frac{\partial f_1}{\partial x_2^i} & \cdots & \frac{\partial f_1}{\partial x_n^i} \\ \frac{\partial f_2}{\partial x_1^i} & \frac{\partial f_2}{\partial x_2^i} & \cdots & \frac{\partial f_2}{\partial x_n^i} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1^i} & \frac{\partial f_n}{\partial x_2^i} & \cdots & \frac{\partial f_n}{\partial x_n^i} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

牛顿迭代具有二阶局部收敛性，因此是否收敛与初值的选取有关，在初值接近方程组的解时，一般保证收敛。

当方程组规模比较大时，每次重新计算 $\boldsymbol{A}_i$ 是很费劲的一件事！拟 Newton 迭代计算新的 $\boldsymbol{A}_{i+1}$，只需要满足如下方程即可

$$\boldsymbol{A}_{i+1}(\boldsymbol{x}^{i+1} - \boldsymbol{x}^i) = \boldsymbol{F}(\boldsymbol{x}^{i+1}) - \boldsymbol{F}(\boldsymbol{x}^i)$$

但当 $n > 1$ 时，$\boldsymbol{A}_{i+1}$ 并不能由上述方程唯一确定，因此通常做法是

$$\boldsymbol{A}_{i+1} = \boldsymbol{A}_i + \Delta \boldsymbol{A}_i, \mathrm{rank}(\Delta \boldsymbol{A}_i) \geq 1$$

当 $\text{rank}(\Delta \boldsymbol{A}_i) = 1$ 时，称为秩 1 拟 Newton 法，其完整的迭代格式如下：

$$\begin{cases} \boldsymbol{x}^{i+1} = \boldsymbol{x}^i - \boldsymbol{A}_i^{-1}\boldsymbol{F}(x^i) \\ \boldsymbol{A}_{i+1} = \boldsymbol{A}_i + [\boldsymbol{y}^i - \boldsymbol{A}_i\boldsymbol{r}^i]\dfrac{(\boldsymbol{r}^i)^T}{(\boldsymbol{r}^i)^T\boldsymbol{r}^i} \\ \qquad\;\; = \boldsymbol{A}_i + \boldsymbol{F}(\boldsymbol{x}^{i+1})\dfrac{(\boldsymbol{r}^i)^T}{(\boldsymbol{r}^i)^T\boldsymbol{r}^i} \end{cases}$$

其中 $\boldsymbol{r}^i = \boldsymbol{x}^{i+1} - \boldsymbol{x}^i, \boldsymbol{y}^i = \boldsymbol{F}(\boldsymbol{x}^{i+1}) - \boldsymbol{F}(\boldsymbol{x}^i)$，推导细节可以参考文献 [2]。

Newton 法和拟 Newton 法是求解非线性方程组最著名的算法，但是，Newton 法每次迭代都要计算雅可比矩阵 A_i，拟 Newton 法每次间接求解雅可比矩阵，这在方程组规模比较大的时候，实现迭代过程需要的存储空间和计算量是非常可观的，不利于推广！

2.3 Barzilai-Borwein 算法

基于 Barzilai-Borwein 算法 [3] 的高效算法 SANE[4] 和 DF-SANE[5](这里 DF 表示 Derivative-Free) 弥补了经典的 Newton 迭代算法缺陷！这些算法统称为谱方法 (Spectral Methods)，主要思想是构造迭代格式：

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha_k\boldsymbol{d}^k, k = 0, 1, 2, \cdots$$

其中 α_k 是步长，d_k 是搜索方向，定义如下：

$$\boldsymbol{d}^k = \begin{cases} -F(\boldsymbol{x}^k) & \text{for DF-SANE,} \\ \pm F(\boldsymbol{x}^k) & \text{for SANE.} \end{cases}$$

对于 SANE 算法，$F(\boldsymbol{x}^k)$ 的符号与评价函数 (merit function)$||F(\boldsymbol{x}^k)||$ 的下降方向一致。步长 α_k 定义如下：

$$\alpha_{k+1} = \begin{cases} \frac{(\boldsymbol{r}^k)^T\boldsymbol{r}^k}{(\boldsymbol{r}^k)^T\boldsymbol{y}^k}, k = 0, 1, \cdots \\ \frac{(\boldsymbol{r}^k)^T\boldsymbol{y}^k}{(\boldsymbol{y}^k)^T\boldsymbol{y}^k}, k = 0, 1, \cdots \end{cases}$$

第一种步长来自文献 [4] 和 [5]，相应的算法记为 sane-1 和 dfsane-1，BBsolve 函数的参数对应 method=1；第二种步长来自文献 [3]，相应的算法记为 sane-2 和 dfsane-2，BBsolve 函数的参数对应 method=2，这是默认设置；另外第三种步长来自文献 [6]，定义如下

$$\alpha_k = \begin{cases} \alpha_0 = \min(1, \frac{1}{||F(\boldsymbol{x}_0)||}), \\ \text{sgn}((\boldsymbol{r}^{k-1})^T\boldsymbol{y}^{k-1})\frac{||(\boldsymbol{r}^{k-1})^T||}{||\boldsymbol{y}^{k-1}||}. \end{cases}$$

以上方法的数值实验比较详见文献 [7]，这里强调的是使用策略，如果默认设置下，BBsolve 求解失败，请切换参数 method。

3 隐函数极值与非线性方程组

设隐函数

$$z = f(x, y, z)$$

构造函数

$$F = F(x, y, z) = z - f(x, y, z)$$

然后计算偏导数

$$\frac{\partial z}{\partial x} = -\frac{F_x}{F_z} = 0$$

和

$$\frac{\partial z}{\partial y} = -\frac{F_y}{F_z} = 0$$

然后联立方程求解，即得极值。因此，可以说这个问题可以看作求解方程 (组)。

求下列隐函数 z 的极小值[1]

$$z = \sin[(zx - 0.5)^2 + 2xy^2 - \frac{z}{10}]e^{-[(x-0.5-e^{-y+z})^2+y^2-\frac{z}{5}+3]}$$

其中，$x \in [-1, 7], y \in [-2, 2]$

可以求解非线性方程 (组) 的 R 包有 rootSolve[8]，BB[9]，nleqslv[10] 等包

就这个方程组来说，首先要求出另外两个方程：

```
library(Deriv)
expr <- expression(z - sin((z * x - 0.5)^2 + 2 * x * y^2 - z/10) * exp(-((x -
    0.5 - exp(-y + z))^2 + y^2 - z/5 + 3)))
DFun <- sapply(all.vars(expr)[-1], function(v) {
    Simplify(D(expr, v))  # 表达式求偏导
})
# all.vars(expr)[-1] # 适用于方程组有多个未知变量
```

为了避免复制粘贴，利用 eval 函数，让表达式转为函数，得到如下方程组：

```
equations <- function(m) {
    x = m[1]
    y = m[2]
    z = m[3]
    f1 <- eval(DFun$x)
    f2 <- eval(DFun$y)
    f3 <- eval(expr)
    c(f1, f2, f3)
}
```

3.1 rootSolve 包

rootSolve 包实现了 Newton-Raphson 算法，该算法迭代求解方程组，求解结果受初始猜测值影响，且不具有全局搜索能力，因此不能保证一定找着方程组的解。此外，非线性方程组一般有多组解，Newton-Raphson 算法从不同初始值迭代，可能收敛到不同的解。

```
library(rootSolve)
library(numDeriv)
multiroot(f = equations, start = c(0, 0, 0))$root
```

[1] 题目来源于 http://www.7d-soft.com/

```
## [1] -2.615394e+00 -7.948542e-05 -5.038561e-11

multiroot(f = equations, start = c(3, -1, -0.02))$root

## [1]  6.927409e-01 -1.354166e+00 -4.039957e-10
```

从求解的结果看，幸运的是，三个初始猜测都找到了方程的解，但是第一组解不在约束区域内（舍去），第二组是可行解，我们有理由相信还有很多组解，在所有的解中要确定最小的 Z，如果一直猜测下去将是没完没了的。

3.2 BB 包

BB 包实现了基于 Barzilai-Borwein 方法的 SANE 和 DF-SANE 算法，算法内容参见文献 [7]。BB 包能求解非线性方程组 (BBsolve 函数)，搜索目标函数的局部极值 (BBoptim 函数)，极小的内存要求，适合求解成千上万个变量的高维问题。

```
library(BB)
p1 <- c(3, -1, -0.02)
# trace 是否显示迭代过程，tol是迭代的终止条件
BBsolve(par = p1, fn = equations, control = list(trace = FALSE, tol = 1e-15))$par

##   Successful convergence.
## [1]  2.89827024 -0.85731300 -0.02335408
```

3.3 nleqslv 包

nleqslv 包基于 Broyden 秩 1 修正和 Newton 算法，采用全局线搜索 (cubic，quadratic or geometric) 和信赖域方法 (double dogleg，Powell single dogleg or Levenberg-Marquardt type) 求解非线性方程组，雅可比矩阵可以奇异 (病态)。

searchZeros 函数较之前两个包求解方程组，可以同时从一组随机设定的初始值迭代求解。

```
library(nleqslv)
N <- 10
set.seed(1234)
xstart <- cbind(runif(N, -1, 7), runif(N, -2, 2), runif(N, -1, 1))  # N initial guesses
ans <- searchZeros(xstart, equations, method = "Broyden", global = "dbldog")
ans$x

##              [,1]      [,2]          [,3]
##  [1,] -4.36932451 17.914361  2.107657e-10
##  [2,] -1.20729171 -0.948079  3.334697e-11
##  [3,]  0.00871189 -1.495315 -1.122215e-12
##  [4,]  2.67153867 -1.822008  2.196321e-11
##  [5,]  3.55393029  2.924213 -6.862717e-10
##  [6,]  4.03843251 -1.985305 -9.317864e-11
```

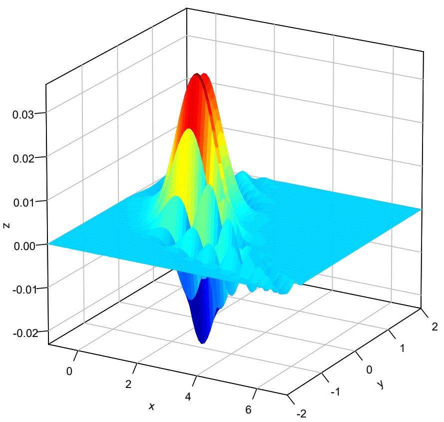
```
## [7,] 4.26680647 -2.256637 -6.395099e-20
## [8,] 4.73051966  1.629647 -1.141917e-10
## [9,] 5.13710699 -2.285852 -8.089367e-09
```

3.4 定初始值

综合以上方法，有一个共同特点就是给出初值，一个好的初值，才有比较不错的结果。因此下面就是如何去寻找一个好的初值。一个简单的办法就是将 x 和 y 构成的约束区域进行网格划分，格点 (x_i, y_i) 代入原隐函数，然后求解一个非线性方程，这样问题就简化下来了！既然是确定初值，网格不必太细，另外，如果维数比较高，可以在网格划分后，再随机选取一定量的格点，以避免陷入过量的计算。

```
x <- seq(-1, 7, length.out = 80)
y <- seq(-2, 2, length.out = 40)
myfun <- function(m) {
    x = m[1]
    y = m[2]
    temFun <- function(z) {
        z - sin((z * x - 0.5)^2 + 2 * x * y^2 - z/10) * exp(-((x - 0.5 - exp(-y +
            z))^2 + y^2 - z/5 + 3))
    }
    return(temFun)
}
xy_mat <- apply(as.matrix(expand.grid(x, y)), 1, function(x) uniroot(myfun(x),
    c(-10, 10))$root)
```

计算网格中每个格点对应的 z 值后，就可以画出三维图像和等高线图。函数图像和代码如下:



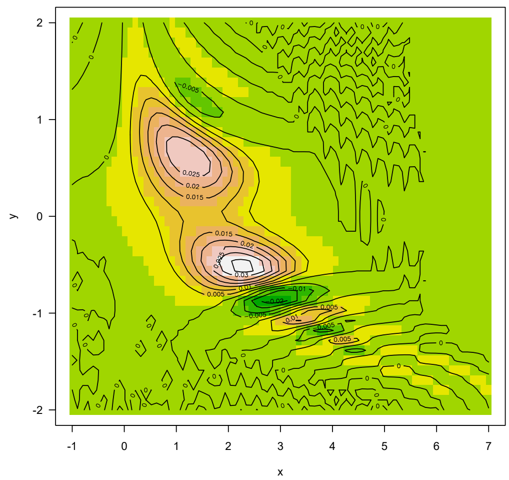


图 1: 隐函数图像

```
library(plot3D)
op <- par(mfrow = c(1, 2), omi = c(0, 0, 0.5, 0), mar = c(4, 4, 1, 1))
z <- matrix(xy_mat, 80, 40)
persp3D(x, y, z, colvar = z, colkey = FALSE, bty = "b2", lighting = F, theta = 30,
    phi = 20, ltheta = 90, lphi = 180, r = 50, d = 0.1, nticks = 5, ticktype = "detailed")
xlabs <- seq(-1, 7, by = 1)
ylabs <- seq(-2, 2, by = 1)
plot(c(-1, 7), c(-2, 2), type = "n", axes = FALSE, xlab = "x", ylab = "y")
axis(1, labels = xlabs, at = xlabs, las = 1)  # x-axis
axis(2, labels = ylabs, at = ylabs, las = 1)  # y-axis
image(x, y, matrix(xy_mat, 80, 40), col = terrain.colors(10), add = TRUE)
contour(x, y, matrix(xy_mat, 80, 40), add = TRUE)
box()
par(op)
```

根据函数图象，不难确定合适的初始值 $(3, -1, -0.02)$，至此，隐函数极值问题的方程组解法就讲到这里！

4 隐函数极值与非线性规划

隐函数极值问题还可以转化为非线性规划问题 (NLPs)，然后用非线性优化方法求解，即求解规划问题[2]

$$
\begin{aligned}
&\min \; z \\
&s.t. \begin{cases} z = \sin[(zx-0.5)^2 + 2xy^2 - \frac{z}{10}]e^{-[(x-0.5-e^{-y+z})^2 + y^2 - \frac{z}{5} + 3]} \\ -1 \leqslant x \leqslant 7 \\ -2 \leqslant y \leqslant 2 \end{cases}
\end{aligned}
$$

```
# 目标函数
fn1 <- function(x) {
    x[3]  # z
}
# 等式约束
eqn1 <- function(x) {
    # x是向量，分量分别对应x,y,z
    x[3] - sin((x[3] * x[1] - 0.5)^2 + 2 * x[1] * x[2]^2 - x[3]/10) * exp(-((x[1] -
        0.5 - exp(-x[2] + x[3]))^2 + x[2]^2 - x[3]/5 + 3))
}
```

4.1 alabama 包

[2] 与商业软件 1stOpt 计算结果一致

```
library(alabama)
x0 <- c(2, -1, 0)
## 不等式约束
hin <- function(x) {
    h <- rep(NA, 4)
    h[1] <- x[1] + 1
    h[2] <- 7 - x[1]
    h[3] <- x[2] + 2
    h[4] <- 2 - x[2]
    h
}
constrOptim.nl(par = x0, fn = fn1, heq = eqn1, hin = hin, control.outer = list(trace = FALSE))$par

## [1]  2.89786606 -0.85740072 -0.02335409
```

一个简单的箱式约束 (box constraints) 都写得这么复杂，这是 alabama 包的缺点。

4.2 Rdonlp2 包

```
library(Rdonlp2)
x0 <- c(2, -1, 0)
donlp2(x0, fn1, par.lower = c(-1, -2, -Inf), par.upper = c(7, 2, Inf), nlin = list(eqn1),
    nlin.l = 0, nlin.u = 0)$par

## [1]  2.89824218 -0.85732232 -0.02335408
```

输出结果太多，篇幅所限，这里仅保留最优解！alabama 和 Rdonlp2 计算结果差不多，初值可以远离局部最优点，但都能找到离初值点最近的局部最优点。

4.3 Rsolnp 包

```
library(Rsolnp)
hin <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    return(c(x1, x2))
}
x0 <- c(3, -1, -0.02)
solnp(x0, fun = fn1, eqfun = eqn1, eqB = 0, ineqfun = hin, ineqLB = c(-1, -2),
    ineqUB = c(7, 2), control = list(trace = FALSE))$pars

## [1]  2.89827001 -0.85731302 -0.02335408
```

对初值要求高，一定要接近方程的解，才能收敛到局部最优解。读者可以尝试其他的初值点，体会一下效果！

5 非线性规划

优化理论的基础最近几十年来没有大的飞跃，能解的问题始终有限，但是作为一个强有力的工具，在各个学科中扮演着重要的角色。数学优化和数学规划、非线性优化和非线性规划、凸优化和凸规划等等一对又一对的名词，本质上没有区别，优化偏数学一点，规划应是运筹学的概念。初学这个领域，发现同一本书里重复出现这些名词，常常被它们绕来绕去！书要是偏理论的一般都是用优化，偏应用都是用规划。由于这个领域还不成熟，会发现书的框架，一会儿是按算法来分的，一会儿是按问题来分的。实际中常常出现一个算法可以解多个问题，一个问题能被多个算法解。基于应用的角度，开源实现的原则，以问题划分章节，这与文献 [11] 和 [12] 中处理方式不同

5.1 非线性最小二乘与无约束优化

设因变量 Y 的期望 $E(Y)$ 与 m 个自变量 $X_1, X_2, ...X_m$ 满足非线性的函数关系：

$$Y = \phi(X_1, X_2, ..., X_m, \beta_1, \beta_2, ..., \beta_r) + \epsilon = \phi(X, \beta) + \epsilon$$

其中 $\beta = (\beta_1, \beta_2, ..., \beta_r)'$ 是未知参数，ϵ 是误差项。

通常我们采用非线性最小二乘准则估计未知参数 β，即求 $\hat{\beta}$，使得

$$F(\beta) = \frac{1}{2}\sum_{i=1}^{n} \epsilon_i^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - \phi(x_i, \beta))^2$$

在 $\beta = \hat{\beta}$ 达到最小值。在这个准则下求 $\hat{\beta}$，就是解一个无约束最优化问题。

我们考虑来自 car 包 [13] 的 USPop 数据集—从 1790 年到 2000 年每十年一次的人口普查数据 (单位: 百万)，一个简单的拟合模型就是 logistic 生长模型

$$y = \frac{\beta_1}{1 + \exp[-(\beta_2 + \beta_3 x)]} + \epsilon$$

nls 函数默认使用 Gauss-Newton 法计算待估系数。非线性回归 (并与线性回归比较) 结果如图所示:

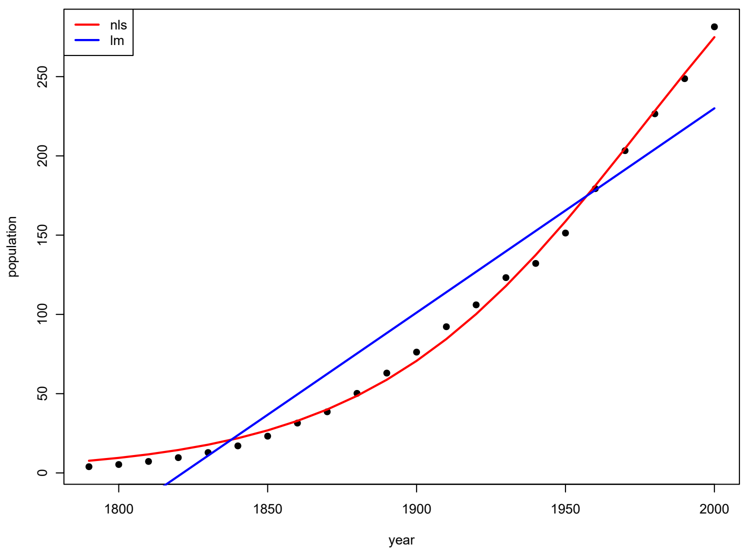


图 2: 非线性回归

```
data(USPop,package = "car")
plot(USPop,pch=16)
##  Gauss-Newton
nls_model <- nls(population ~ beta1/(1+exp(-(beta2+beta3*year))),
                 start = list(beta1=400,beta2=-49,beta3=0.025),
                 data = USPop,trace = TRUE)
# summary(nls_model)
fit_model <- fitted(nls_model)
lines(fit_model~USPop$year,col="red",lwd=2)
## linear regression least-squares
lm_model <-lm(population~year,data = USPop)
fit_lm_model <- fitted(lm_model)
lines(fit_lm_model~USPop$year,col="blue",lwd=2)
legend( "topleft",legend = c("nls","lm"),
        col = c("red","blue"),lwd = 2,cex = 1)
```

5.2 复杂非线性回归

这是一道面试题[3]，给出了问题背景和拟合模型结构，以及求解模型所需的数据。拟合模型结构如下：

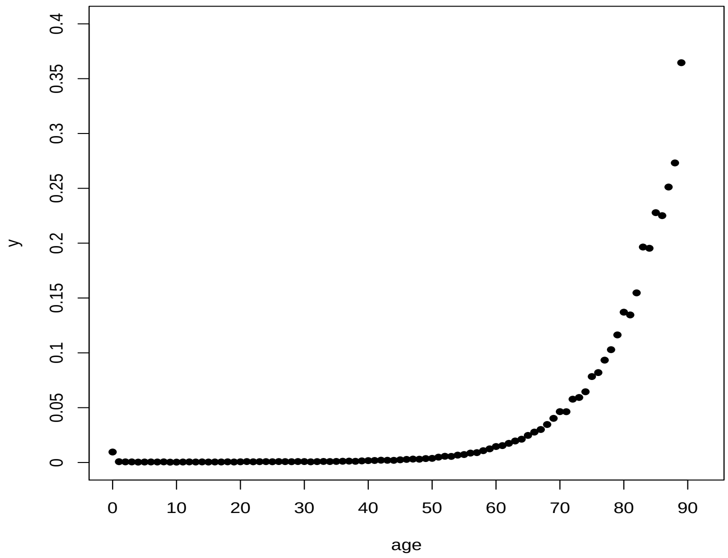
$$y = A^{(x+B)^C} + D\exp(-E(\ln x - \ln F)^2) + GH^x$$

x 代表年龄 age(已知项)，y 是响应变量 (已知项)，$A, B, ..., G$ 是模型 8 个待估参数。

查看数据和散点图如下：

```
## 'data.frame': 91 obs. of  3 variables:
##  $ age: int  0 1 2 3 4 5 6 7 8 9 ...
##  $ q  : num  0.009462 0.000743 0.000577 0.000541 0.000349 ...
##  $ y  : num  0.009553 0.000744 0.000577 0.000542 0.000349 ...
```

[3]https://github.com/Cloud2016/Interview



5.3 一维优化问题

解下列优化问题

$$\min \quad \frac{25R(T)+30[1-R(T)]}{(T+0.7)R(T)+\int_0^T(t+0.9)f(t)dt}$$

$$s.t. \quad \frac{\int_0^T R(T)dt}{(T+0.7)R(T)+\int_0^T(t+0.9)f(t)dt} \geq 0.92, T \geq 0$$

其中:

$$R(t) = \exp[-(\frac{t}{5000})^{2.5}]$$

$$f(t) = \frac{2.5}{5000}(\frac{t}{5000})^{1.5}\exp[-(\frac{t}{5000})^{2.5}]$$

这里 T 为未知数，即求解满足约束条件，同时使目标函数最终取值最小的时间 T。

先写几个函数

```
Rt <- function(t) {
    # R(t)
    exp(-(t/5000)^(2.5))
}

Ft <- function(t) {
    # F(t)
    2.5/5000 * (t/5000)^(1.5) * Rt(t)
}
Ftt <- function(t) {
    (t + 0.9) * Ft(t)
}
```

```
RFt <- function(t) {
    (t + 0.7) * Rt(t) + integrate(Ftt, 0, t)$value
}
```

求解可行域

```
f <- function(t) {
    # s.t.
    integrate(Rt, 0, t)$value/RFt(t) - 0.92
}
uniroot(f, c(0, 100))  # call the Brent Algrithm

## $root
## [1] 8.05
##
## $f.root
## [1] -1.207512e-11
##
## $iter
## [1] 10
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 0.0001075021
```

目标函数

```
objfun <- function(t) {
    (25 * Rt(t) + 30 * (1 - Rt(t)))/RFt(t)
}
```

调 stats 包内 nlminb 函数求解优化问题

```
nlminb(9, objfun, lower = 8.05, upper = Inf)

## $par
## [1] 9706.718
##
## $objective
## [1] 0.006760459
##
## $convergence
## [1] 0
```

```
##
## $iterations
## [1] 38
##
## $evaluations
## function gradient
##       40       42
##
## $message
## [1] "relative convergence (4)"
```

小结：

通常教科书里的例子都是可以直接调用 nlminb 函数，一行代码了事。没有考虑目标函数和约束条件的复杂性，更没有考虑可行域 (通常都是直接给出)。事实上，目标函数的性质，可行域的形状都决定着优化问题的类型，求解算法的选用[4]。

5.4 箱式约束

解非线性优化问题

$$\min \ z = (1-x)^2 + 100(y-x^2)^2$$
$$s.t. \begin{cases} x \leqslant 1.9 \\ y \leqslant 1.5 \end{cases}$$

```r
fr <- function(x) {
    100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2
}
library(numDeriv)
grr <- function(x) {
    grad(fr, c(x[1], x[2]))  # gradient
}
```

5.4.1 optim 函数

```r
optim(c(-1.2, 1), fr, grr, method = "L-BFGS-B", lower = c(-Inf, -Inf), upper = c(1.9,
    1.5))
```

```
## $par
## [1] 1 1
##
## $value
## [1] 2.678097e-17
##
```

[4] 此处只是想强调求解优化问题的实际过程

```
## $counts
## function gradient
##       51       51
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

5.4.2 nlminb 函数

```
nlminb(c(-1.2, 1), fr, grr, lower = c(-Inf, -Inf), upper = c(1.9, 1.5))
```

```
## $par
## [1] 1 1
##
## $objective
## [1] 2.022999e-21
##
## $convergence
## [1] 0
##
## $iterations
## [1] 35
##
## $evaluations
## function gradient
##       44       36
##
## $message
## [1] "X-convergence (3)"
```

5.4.3 BB 包

BB 包里的 BBoptim 函数可解带箱式约束 (box constrains) 的非线性优化问题。

```
Rosenbrock <- function(x) {
    # Extended Rosenbrock function
    n <- length(x)
    j <- 2 * (1:(n/2))
    jm1 <- j - 1
```

```
    sum(100 * (x[j] - x[jm1]^2)^2 + (1 - x[jm1])^2)
}
p0 <- c(1.2, 1.3)
BBoptim(par = p0, fn = Rosenbrock, method = 3, lower = rep(0, 2), upper = rep(2,
    2), control = list(trace = FALSE))
```

```
##   Successful convergence.
## $par
## [1] 0.9996846 0.9993682
##
## $value
## [1] 9.959223e-08
##
## $gradient
## [1] 0.0002283974
##
## $fn.reduction
## [1] 2
##
## $iter
## [1] 47
##
## $feval
## [1] 52
##
## $convergence
## [1] 0
##
## $message
## [1] "Successful convergence"
##
## $cpar
## method      M
##      3     50
```

5.4.4 Rcgmin 包

Rcgmin 包 [14] 共轭梯度算法，非线性函数极小

```
library(Rcgmin)
Rcgmin(fn = fr, gr = grr, par = p0, lower = rep(0, 2), upper = rep(2, 2))
```

```
## $par
## [1] 0.9999999 0.9999999
```

```
##
## $value
## [1] 4.448528e-15
##
## $counts
## [1] 47 22
##
## $convergence
## [1] 0
##
## $message
## [1] "Rcgmin seems to have converged"
##
## $bdmsk
## [1] 1 1
```

5.4.5 Rvmmin 包

Rvmmin 包 [15] 变尺度算法 (Variable Metric)，非线性函数极小

```
library(optextras)
library(Rvmmin)
solution <- Rvmmin(fn = fr, gr = grr, par = c(1.2, 1.2), lower = rep(0, 2),
    upper = rep(2, 2))
print(solution)
```

```
## $par
## [1] 1 1
##
## $value
## [1] 4.14645e-29
##
## $counts
## [1] 30 24
##
## $convergence
## [1] 0
##
## $message
## [1] "Rvmminb appears to have converged"
##
## $bdmsk
## [1] 1 1
```

5.4.6 dfoptim 包

dfoptim 包 [16] 不需要梯度信息，适用不可微的目标函数优化。hjk 函数实现 Hooke-Jeeves 算法，nmk 函数实现 Nelder-Mead 算法

```
library(dfoptim)
hjkb(par = p0, fn = fr, lower = rep(0, 2), upper = rep(2, 2))

## $par
## [1] 0.9999992 0.9999969
##
## $value
## [1] 2.334129e-10
##
## $convergence
## [1] 0
##
## $feval
## [1] 613
##
## $niter
## [1] 19

nmkb(par = p0, fn = fr, lower = rep(0, 2), upper = rep(2, 2))

## $par
## [1] 0.9995533 0.9990923
##
## $value
## [1] 2.203708e-07
##
## $feval
## [1] 65
##
## $restarts
## [1] 0
##
## $convergence
## [1] 0
##
## $message
## [1] "Successful convergence"
```

5.5 线性约束

$$\min\ z=(1-x)^2+100(y-x^2)^2$$
$$s.t.\begin{cases} x\leqslant 1.9\\ y\leqslant 1.5\\ x-y\leqslant -0.1\end{cases}$$

5.5.1 constrOptim 函数

```
# ui %*% theta - ci >= 0
constrOptim(c(0.5, 1), fr, grr, ui = rbind(c(-1, 0), c(0, -1), c(-1, 1)), ci = c(-1.9,
    -1.5, 0.1))
```

```
## $par
## [1] 1.090957 1.190957
##
## $value
## [1] 0.008332439
##
## $counts
## function gradient
##      241       50
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $outer.iterations
## [1] 4
##
## $barrier.value
## [1] -0.0001647574
```

5.6 非线性约束

5.6.1 NlcOptim 包

NlcOptim 包 [17]NlcOptim 函数实现了序列二次规划 SQP 算法，适用于求解非线性目标函数，线性、非线性的等式和不等式约束优化。一般约束优化问题，描述如下：

$$
\min\ f(x)\\
s.t.\begin{cases}
ceq(x)=0, & \text{等式约束}\\
c(x)\leqslant 0, & \text{不等式约束}\\
Ax\leqslant B, & \text{线性不约束}\\
Aeqx\leqslant Beq, & \text{线性等式约束}\\
lb\leqslant x\leqslant ub, & \text{变量边界约束}
\end{cases}
$$

以下列非线性优化问题为例：

$$
\min\ z=(1-x)^2+100(y-x^2)^2\\
s.t.\begin{cases}
x^2+y^2\leqslant 2\\
y-x=0\\
0\leqslant x\leqslant 2\\
0\leqslant y\leqslant 2
\end{cases}
$$

```
library(MASS)
library(NlcOptim)
# 约束条件函数
con <- function(x) {
    f = NULL  # 不等式约束
    f = rbind(f, x[1]^2 + x[2]^2 - 2)
    g = NULL  # 等式约束
    g = rbind(g, x[2] - x[1])
    return(list(ceq = g, c = f))
}
# lb,ub 矩阵类型
NlcOptim(X = c(0, 0), objfun = fr, confun = con, lb = as.matrix(rep(0, 2)),
    ub = as.matrix(rep(2, 2)))

## $p
##              [,1]
## [1,] 0.9999996
## [2,] 0.9999996
##
## $fval
## [1] 1.429828e-11
##
## $lambda
## $lambda$lower
```

```
##       [,1]
## [1,]     0
## [2,]     0
##
## $lambda$upper
##       [,1]
## [1,]     0
## [2,]     0
##
## $lambda$eqnonlin
## [1] 0.0002235099
##
## $lambda$ineqnonlin
## [1] 3.575903e-05
##
##
## $grad
##               [,1]
## [1,] -0.0002965327
## [2,]  0.0001519917
##
## $hessian
##            [,1]       [,2]
## [1,]  2.4995545 -0.4998495
## [2,] -0.4998495  0.5000000
```

再举个例子

$$\max \ z = \frac{\sin(2\pi x)^3 \sin(2\pi y)}{x^3(x+y)}$$
$$s.t. \begin{cases} x^2 - y + 1 \leqslant 0 \\ 1 - x + (y-4)^2 \leqslant 0 \\ 0 \leqslant x, y \leqslant 10 \end{cases}$$

对于这种复杂的例子，初始猜测值需要接近最优解才会有好的效果！

```
fr <- function(x) {
    # NlcOptim函数默认求极小，因此目标函数添加负号
    -sin(2 * pi * x[1])^3 * sin(2 * pi * x[2])/(x[1]^3 * (x[1] + x[2]))
}
con <- function(x) {
    f = NULL  # 不等式约束
    f = rbind(f, x[1]^2 - x[2] + 1)
    f = rbind(f, 1 - x[1] + (x[2] - 4)^2)
    return(list(ceq = NULL, c = f))
}
```

```
NlcOptim(X = c(1.2, 4.2), objfun = fr, confun = con, lb = as.matrix(rep(0, 2)),
    ub = as.matrix(rep(10, 2)))
```

```
## $p
##           [,1]
## [1,] 1.227974
## [2,] 4.245371
##
## $fval
## [1] -0.09582504
##
## $lambda
## $lambda$lower
##      [,1]
## [1,]    0
## [2,]    0
##
## $lambda$upper
##      [,1]
## [1,]    0
## [2,]    0
##
## $lambda$ineqnonlin
## [1] 0 0
##
##
## $grad
##                [,1]
## [1,] -0.0029815984
## [2,] -0.0003950376
##
## $hessian
##              [,1]       [,2]
## [1,] 11.25832393 0.05312406
## [2,]  0.05312406 3.73455233
```

Rdonlp2 包也需要一个好的初始值，才能获得最优解。

```
# library(Rdonlp2)
x0 <- c(1, 4)
con1 <- function(x) {
    x[1]^2 - x[2] + 1
}
```

```
con2 <- function(x) {
    1 - x[1] + (x[2] - 4)^2
}
nlin.l = c(-Inf, -Inf)
nlin.u = c(0, 0)
donlp2(par = c(1.2, 4.1), fr, par.lower = c(0, 0), par.upper = c(10, 10), nlin = list(con1,
    con2), nlin.l = nlin.l, nlin.u = nlin.u)$par

## [1] 1.227971 4.245373
```

5.6.2 Rnlminb2 包

Rnlminb2 包 [18] 内点法求解非线性约束优化，是 nlminb 的升级版，因为添加了非线性约束。

```
library(Rnlminb2)
nlminb2NLP(start = c(1.2, 4.1), fun = fr, eqFun = list(con1, con2), par.lower = c(0,
    0), par.upper = c(10, 10))

## $opt
## $opt$par
## [1] 1.227971 4.245373
##
## $opt$objective
## [1] -0.09582504
##
## $opt$convergence
## [1] 0
##
## $opt$iterations
## [1] 8
##
## $opt$evaluations
## function gradient
##       32       24
##
## $opt$message
## [1] "both X-convergence and relative convergence (5)"
##
##
## $solution
## [1] 1.227971 4.245373
##
## $objective
## [1] -0.09582504
```

```
##
## $status
## [1] 0
##
## $message
## [1] "both X-convergence and relative convergence (5)"
##
## $solver
## [1] "nlminb2NLP"
##
## $elapsed
## Time difference of 0.01563096 secs
##
## $version
## [1] "Rnlminb2 3002.10 2009-04-12"
##
## attr(,"class")
## [1] "solver" "list"
```

以上出现的求解非线性规划的 R 包和函数都只能获得局部最优解！

5.7 凸优化

一般非线性优化问题：

$$
\begin{aligned}
\min \quad & f(x) \\
s.t. \quad & g(x) = 0 \\
& l_h \leq h(x) \leq u_h \\
& l_x \leq x \leq u_x
\end{aligned}
$$

其中 $f(x)$，$g(x)$ 和 $h(x)$ 是光滑的

求解下列优化问题

$$
\begin{aligned}
\min \quad & f(x) = -2x_1^2 - x_2^2 \\
s.t. \quad & x_1^2 + x_2^2 - 2 = 0 \\
& x_2 - e^{x_1} \geq 0 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

可行域是一段弧，这里加载 Rsolnp 包，调用 solnp 函数求解，基于增广拉格朗日乘子法和 SQP 内点算法

```r
library(Rsolnp)
fn1 <- function(x) {
    -2 * x[1]^2 - x[2]^2
}
```

```r
eqn1 <- function(x) {
    x[1]^2 + x[2]^2
}
ineqn1 <- function(x) {
    exp(x[1]) - x[2]
}
x0 = c(1/2, sqrt(3)/2)
solnp(x0, fun = fn1, eqfun = eqn1, eqB = 2, ineqfun = ineqn1, ineqLB = -Inf,
    ineqUB = 0, LB = c(0, 0), UB = c(Inf, Inf))
```

```
##
## Iter: 1 fn: -2.6015  Pars:  0.54216 1.41903
## Iter: 2 fn: -2.1598  Pars:  0.34753 1.38501
## Iter: 3 fn: -2.1036  Pars:  0.32074 1.37764
## Iter: 4 fn: -2.1026  Pars:  0.32025 1.37748
## Iter: 5 fn: -2.1026  Pars:  0.32025 1.37748
## Iter: 6 fn: -2.1026  Pars:  0.32025 1.37748
## solnp--> Completed in 6 iterations
## $pars
## [1] 0.3202524 1.3774754
##
## $convergence
## [1] 0
##
## $values
## [1] -1.250000 -2.601534 -2.159816 -2.103646 -2.102562 -2.102562 -2.102562
##
## $lagrange
##              [,1]
## [1,] -1.1444080
## [2,] -0.3978371
##
## $hessian
##              [,1]       [,2]       [,3]
## [1,]  0.2120107 -0.3672395  0.2050988
## [2,] -0.3672395  1.5596659 -0.2063528
## [3,]  0.2050988 -0.2063528  0.9911041
##
## $ineqx0
## [1] -2.682989e-10
##
## $nfuneval
## [1] 130
```

```
##
## $outer.iter
## [1] 6
##
## $elapsed
## Time difference of 0.2031269 secs
##
## $vscale
## [1] 2.10256159 0.00000001 1.00000000 1.00000000 1.00000000
```

$$
\begin{aligned}
\min \quad & f(x) = -2x_1^2 - x_2^2 \\
s.t. \quad & x_1^2 + x_2^2 - 2 \le 0 \\
& -x_1 + x_2 \ge 0 \\
& x_1, x_2 \ge 0
\end{aligned}
$$

```
ineqn2 <- function(x) {
    c(x[1]^2 + x[2]^2, x[1] - x[2])
}
solnp(c(1/2, 3/4), fun = fn1, eqfun = eqn1, eqB = 2, ineqfun = ineqn2, ineqLB = c(-Inf,
    -Inf), ineqUB = c(2, 0), LB = c(0, 0), UB = c(Inf, Inf))

##
## Iter: 1 fn: -3.3421  Pars:  0.94784 1.24310
## solnp--> Solution not reliable....Problem Inverting Hessian.
## $pars
## [1] 0.9478435 1.2431043
##
## $convergence
## [1] 2
##
## $values
## [1] -1.062500 -3.342123
##
## $lagrange
## [1] 0
##
## $hessian
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
```

```
## [4,]    0    0    0    1
##
## $ineqx0
## [1]  2.0000000 -0.2952608
##
## $nfuneval
## [1] 7
##
## $outer.iter
## [1] 1
##
## $elapsed
## Time difference of 0 secs
##
## $vscale
## [1] 1.0625 1.1875 1.0000 1.0000 1.0000 1.0000
```

5.8 应用

5.8.1 正态分布参数估计

估计正态分布的参数

```
set.seed(123)
n <- 1000  # 产生n个正态随机数
mu <- 4   # 预先设定的均值和方差
Var <- 4
x <- rnorm(n,mean=mu,sd= Var)
# 矩估计
theta <- c(mu=mean(x),sigma=sqrt((n-1)*var(x)/n))
# 存储数据
# library(RevoScaleR)
# rxDataStep(inData = data.frame(x), outFile = "optim-work.xdf",
#        overwrite = TRUE,rowsPerRead = 2000, reportProgress = 0)
# 对数似然函数
logLikFun <- function(param, mleData) {
  # param 待估参数
                mu <- param[1]  # 均值
                sigma <- param[2] # 方差
                -sum(dnorm(mleData, mean = mu, sd = sigma, log = TRUE))
}
# 二元无约束优化 optim默认选择方法: Nelder and Mead
mle <- optim(par = c(mu = 0, sigma = 1),fn = logLikFun,mleData = x)
```

```
# 样本量很大的时候，最大似然估计没有矩估计好
rbind(mle = mle$par, standard = theta)
```

```
##                  mu    sigma
## mle      4.063699 3.965281
## standard 4.064511 3.964796
```

改变样本量计算均值和方差，比较两种估计随样本量的变化！
样本量小 (1000) 的时候，均值，似然估计比矩估计差，方差，似然估计比矩估计好
样本量大 (10000) 的时候，矩估计均值方差一致比似然估计好

5.8.2 二项泊松混合分布参数估计

Nelder-Mead 数值方法

```
set.seed(1234)   # 设置随机数种子,用于可重复实现
# 产生二项泊松混合分布随机数
bipossion <- function(n,p,theta){
        # n 随机数个数
        # p 比列 向量 sum(p)=1
        # theta 泊松分布参数 length(theta) 子泊松分布个数
        # length(p)=length(theta)=2
        x <- rep(NA,n)
        temp <- rbinom(n,1,p)
# x <- ifelse(temp==1,rpois(1,theta[1]),rpois(1,theta[2]))  # 为何不对
        for(i in seq(n)){
            if(temp[i]==1) x[i] <- rpois(1,theta[1])
            if(temp[i]==0) x[i] <- rpois(1,theta[2])
        }
        return(x)
}

# 似然函数
fun <- function(mleData){
  logLikFun <- function(param){
    p <- param[1];
    theta1 <- param[2];
    theta2 <- param[3];
    -sum(log(p*dpois(mleData,theta1)+(1-p)*dpois(mleData,theta2)))
  }
  return(logLikFun)
}
# 矩估计和最大似然估计
ME_MLE <- function(n,p,theta){
```

```
  x <- bipossion(n,p,theta) ## 二项泊松混合分布
  x1 <- rpois(n,theta[1])
  x2 <- rpois(n,theta[2])
  ME <- c(mean(x1),mean(x2)) # ME矩估计 moment estimation
  MLE <- optim(par = c(0.06,mean(x1),mean(x2)),fn = fun(mleData=x))$par
  # MLE估计  maximum likelihood estimation
  list(me=ME,mle=MLE)
}
system.time(res<- ME_MLE(n=10000,p=0.6,theta = c(2,4)))
```

```
##    user  system elapsed
##    1.95    0.00    1.97
```

```
res
```

```
## $me
## [1] 2.0265 3.9505
##
## $mle
## [1] 0.5939049 2.0316159 3.9724824
```

SANN 模拟退火方法

```
n = 10000
p = 0.6
theta = c(2, 4)
x <- bipossion(n, p, theta)
x1 <- rpois(n, theta[1])
x2 <- rpois(n, theta[2])
optim(par = c(0.06, mean(x1), mean(x2)), fn = fun(mleData = x), method = "SANN")
```

```
## $par
## [1] 0.5757513 1.9557854 3.9377440
##
## $value
## [1] 20014.17
##
## $counts
## function gradient
##    10000       NA
##
## $convergence
## [1] 0
##
## $message
```

```
## NULL
```

牛顿-拉弗森算法

```
library(Deriv)
expr <- expression(p * theta1^x/exp(theta1) + (1 - p) * theta2^x/exp(theta2))
DFun <- sapply(all.vars(expr)[-3], function(v) {
    Simplify(D(expr, v))  # 表达式求偏导
})
tempFun <- function(x) {
    equations <- function(m) {
        p = m[1]
        theta1 = m[2]
        theta2 = m[3]
        f1 <- sum((dpois(x, theta1) - dpois(x, theta2))/(p * dpois(x, theta1) +
            (1 - p) * dpois(x, theta2)))
        f2 <- sum((p * eval(DFun$theta1)/factorial(x))/(p * dpois(x, theta1) +
            (1 - p) * dpois(x, theta2)))
        f3 <- sum(((1 - p) * eval(DFun$theta2)/factorial(x))/(p * dpois(x, theta1) +
            (1 - p) * dpois(x, theta2)))
        c(f1, f2, f3)
    }
    return(equations)
}
multiroot(f = tempFun(x), start = c(0.06, mean(x1), mean(x2)))$root

## [1] -0.2144497 -4.3908603  2.3360875
```

BB 算法

```
# 比rootSolve效果好 ,能收敛到真解, 只是误差较大
BBsolve(par = c(0.06, mean(x1), mean(x2)), fn = tempFun(x), method = 2, control = list(trace = FALSE,
    NM = TRUE, tol = 1e-15))$par

##   Unsuccessful convergence.
## [1] 1.158441 2.795900 2.795900
```

调 multiStart 函数

```
p0 <- matrix(runif(900), 300, 3)  # 300组初始值
ans <- multiStart(par = p0, fn = tempFun(x), action = "solve")
sum(ans$conv)  # 收敛的情况数目
pmat <- ans$par[ans$conv, ]
ord1 <- order(pmat[, 1])
ans <- round(pmat[ord1, ], 4)
ans[!duplicated(ans), ]
```

nls 非线性回归

```
Poisson <- data.frame(x = seq(from = 0, to = 13, by = 1), y = c(0.0862, 0.1862,
    0.2257, 0.1831, 0.1375, 0.0853, 0.0508, 0.0244, 0.0124, 0.0046, 0.0024,
    8e-04, 4e-04, 2e-04))
nls(y ~ p * dpois(x, theta1) + (1 - p) * dpois(x, theta2), start = list(p = 0.06,
    theta1 = mean(x1), theta2 = mean(x2)), data = Poisson, trace = FALSE)

## Nonlinear regression model
##   model: y ~ p * dpois(x, theta1) + (1 - p) * dpois(x, theta2)
##    data: Poisson
##      p theta1 theta2
## 0.5965 2.0343 3.9882
##  residual sum-of-squares: 7.268e-05
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 1.517e-06
```

nlm 无约束极小

```
fun <- function(mleData) {
    logLikFun <- function(param) {
        p <- param[1]
        theta1 <- param[2]
        theta2 <- param[3]
        -sum(log(p * dpois(mleData, theta1) + (1 - p) * dpois(mleData, theta2)))
    }
    return(logLikFun)
}
nlm(fun(x), c(0.06, mean(x1), mean(x2)), hessian = FALSE)

## $minimum
## [1] 20014.16
##
## $estimate
## [1] 0.5780794 1.9590208 3.9425252
##
## $gradient
## [1] -0.008752977  0.001662050  0.004829692
##
## $code
## [1] 1
##
## $iterations
## [1] 18
```

Rdonlp2 大规模无约束极小

```
donlp2(c(0.06, mean(x1), mean(x2)), fun(x))$par

## [1] 0.5780896 1.9590380 3.9425451
```

BFGS 算法

```
grr <- function(param) {
    grad(fun(x), c(param[1], param[2], param[3]))  # gradient
}
optim(c(0.06, mean(x1), mean(x2)), fun(x), grr, method = "BFGS")$par

## [1] 0.5783057 1.9593788 3.9431136
```

BBoptim

```
# Log-likelihood for a binary Poisson mixture distribution
poissmix.loglik <- function(p, y) {
    # p 参数: 比例 theta1 theta2 y 频率数据
    i <- 0:(length(y) - 1)
    loglik <- y * log(p[1] * exp(-p[2]) * p[2]^i/exp(lgamma(i + 1)) + (1 - p[1]) *
        exp(-p[3]) * p[3]^i/exp(lgamma(i + 1)))
    return(sum(loglik))
}
lo <- c(0, 0, 0)  # lower limits for parameters
hi <- c(1, Inf, Inf)  # upper limits for parameters
p0 <- c(0.06, mean(x1), mean(x2))  # a randomly generated vector of length 3
y <- c(862, 1862, 2257, 1831, 1375, 853, 508, 244, 124, 46, 24, 8, 4, 2)
BBoptim(par = p0, fn = poissmix.loglik, y = y, lower = lo, upper = hi, control = list(maximize = TRUE))$

## iter:  0  f-value:  -21693.49  pgrad:  4.02
## iter:  10  f-value:  -20014.44  pgrad:  21.54491
## iter:  20  f-value:  -20007.85  pgrad:  12.0492
## iter:  30  f-value:  -20005.31  pgrad:  0.7950439
## iter:  40  f-value:  -20005.3  pgrad:  0.06672053
## iter:  50  f-value:  -20005.3  pgrad:  3.637979e-05
##   Successful convergence.
## [1] 0.5939202 2.0314715 3.9727857
```

GenSA

```
library(GenSA)
poissmix.loglik <- function(p,y) {
        i <- 0:(length(y)-1)
        loglik <- y * log(p[1] * exp(-p[2]) * p[2]^i / exp(lgamma(i+1)) +
```

```
                          (1 - p[1]) * exp(-p[3]) * p[3]^i / exp(lgamma(i+1)))
        return (-sum(loglik) )
}
lower <- c(0,0,0)
upper <- c(1,Inf,Inf)
tol     <- 1e-13
ctrl <- list(threshold.stop=tol,verbose =TRUE)
SA1 <- GenSA(par=c(0.06,mean(x1),mean(x2)),lower = lower,upper = upper,
                                y=y,fn=poissmix.loglik,control = ctrl)
```

```
## It: 1, obj value: 20005.29951
## ..........................................................................................
## It: 800, obj value: 20005.29951
## ..........................................................................................
```

```
print(SA1[c("value","par","counts")])
```

```
## $value
## [1] 20005.3
##
## $par
## [1] 0.5939219 2.0314741 3.9727899
##
## $counts
## [1] 60630
```

DEoptim 差分进化

```
library(DEoptim)
loglik <- function(y) {
    # p 参数: 比例 theta1 theta2 y 频率数据
    fun <- function(p) {
        i <- 0:(length(y) - 1)
        loglik <- y * log(p[1] * exp(-p[2]) * p[2]^i/exp(lgamma(i + 1)) + (1 -
            p[1]) * exp(-p[3]) * p[3]^i/exp(lgamma(i + 1)))
        return(-sum(loglik))
    }
    return(fun)
}
lower <- c(0, 0, 0)
upper <- c(1, 10, 10)
DEopt <- DEoptim(lower = lower, upper = upper, fn = loglik(y), control = list(trace = FALSE))
print(DEopt$optim)
```

```
## $bestmem
```

```
##      par1      par2      par3
## 0.5950872 2.0336796 3.9757643
##
## $bestval
## [1] 20005.3
##
## $nfeval
## [1] 402
##
## $iter
## [1] 200
```

A simple Monte Carlo optimizer using adaptive coordinate sampling

```
library(smco)

## This is smco package 0.1

asmcol <- smco(par = c(0.06, mean(x1), mean(x2)), LB = lower, UB = upper, fn = loglik(y),
    maxiter = 1e+05, trc = FALSE)
print(asmcol[c("par", "value")])

## $par
## [1] 0.5994617 2.0403952 3.9871445
##
## $value
## [1] 20005.31
```

5.9 Scilab 接口

Scilab[5]功能类似 MATLAB，同样的开源软件还有 Octave[6]。optimbase 包 [19]、optimsimplex 包 [20]、neldermead 包 [21] 和 Matrix 包 [22]，合一起实现开源软件 Scilab 内的基本命令和优化求解器 fminsearch。

```
library(Matrix)
library(optimbase)
library(optimsimplex)
library(neldermead)
zeros(3)

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

[5] http://www.scilab.org/

[6] http://www.gnu.org/software/octave/

```
ones(4, 5)
```

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

```
# function evaluations
banana <- function(x) {
    y <- 100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2
}
sol <- fminsearch(banana, c(-1.2, 1))
sol

opt <- optimset(TolX = 0.01)
# The absolute tolerance on simplex size. The default value is 1.e-4.
sol <- fminsearch(banana, c(-1.2, 1), opt)
sol

outfun <- function(x, optimValues, state) {
    plot(x[1], x[2], xlim = c(-1.5, 1.5), ylim = c(-1.5, 1.5))
    par(new = TRUE)
}
opt <- optimset(OutputFcn = outfun)
sol <- fminsearch(banana, c(-1.2, 1), opt)
sol

opt <- optimset(MaxIter = 10)
sol <- fminsearch(banana, c(-1.2, 1), opt)

opt <- optimset(Display = "iter")
sol <- fminsearch(banana, c(-1.2, 1), opt)
sol
```

6 启发式算法

6.1 随机数生成

随机数发生器[7] (Random Number Generator) 是进行随机模拟的基础，如果没有可靠的随机数发生器，一切计算结果都没有说服力。一个好的随机数发生器，必是在效率和质量之间取得很好的平衡！效率就是短时间内产生足够多的随机数，质量就是通过一系列测试，以满足计算需要。

[7] 确切地说是伪随机数发生器 Pseudo-Random Number Generator

如果有一天模拟的结果不是预想的那样，程序也没有出错 (事实上，十有八九就是程序出问题)，就该考虑随机数发生器了。

如果不是涉及到超大规模的模拟计算，一般计算软件内置的随机数发生器就能满足计算需要了。

下面以 R 软件平台为例，就 Mersenne-Twister 随机数发生器做 Run-Length 测试，

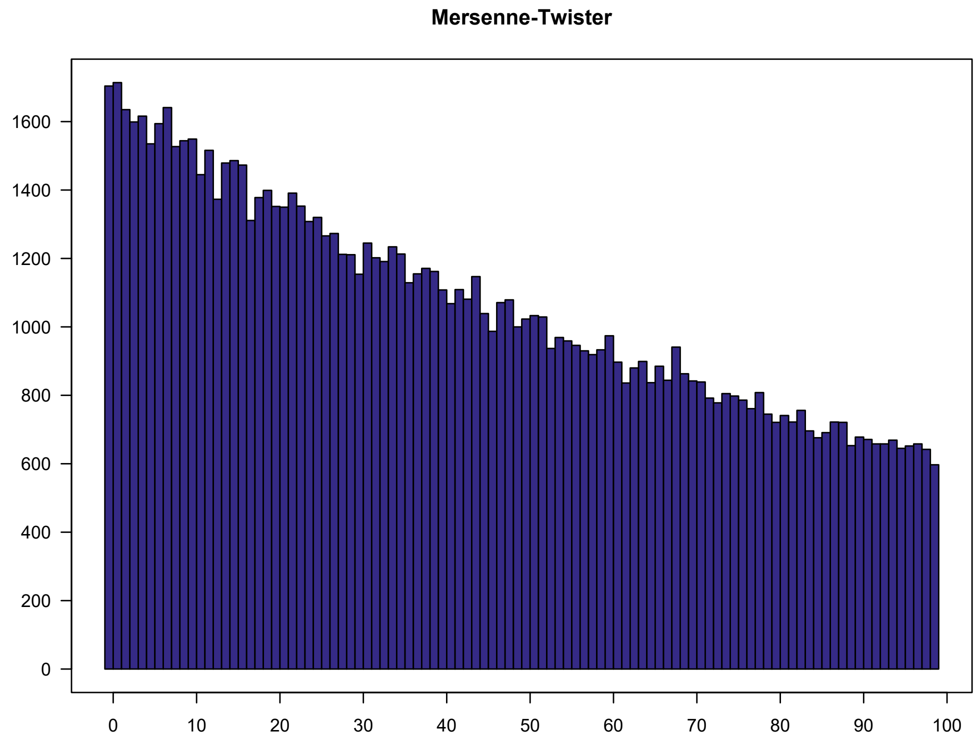


图 3: 游程检验

```
set.seed(1234, kind = "Mersenne-Twister")
n = 2^24
x <- runif(n)
delta = 0.01
len <- diff(c(0, which(x < delta), n + 1)) - 1
dat <- table(len[len < 100])
ylabs <- seq(0, 1800, by = 200)
xlabs <- seq(0, 100, by = 10)

hist(len[len < 100], breaks = -1:99, col = "#352A86", border = "black", axes = FALSE,
    ann = FALSE)
title(main = "Mersenne-Twister")
axis(1, labels = xlabs, at = xlabs, las = 1)  # x-axis
axis(2, labels = ylabs, at = ylabs, las = 1)  # y-axis
box()
```

7 R 软件信息

```r
sessionInfo()
```

```
## R version 3.2.5 (2016-04-14)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 14393)
##
## locale:
## [1] LC_COLLATE=Chinese (Simplified)_China.936
## [2] LC_CTYPE=Chinese (Simplified)_China.936
## [3] LC_MONETARY=Chinese (Simplified)_China.936
## [4] LC_NUMERIC=C
## [5] LC_TIME=Chinese (Simplified)_China.936
##
## attached base packages:
## [1] graphics  grDevices datasets  stats     utils     methods   base
##
## other attached packages:
##  [1] neldermead_1.0-10   optimsimplex_1.0-5 optimbase_1.0-9
##  [4] Matrix_1.2-6        smco_0.1           DEoptim_2.2-3
##  [7] GenSA_1.1.6         Rnlminb2_3002.10   NlcOptim_0.4
## [10] MASS_7.3-45         dfoptim_2016.7-1   Rvmmin_2013-11.12
## [13] optextras_2016-8.8 Rcgmin_2013-2.21   Rsolnp_1.16
## [16] Rdonlp2_3002.10     alabama_2015.3-1   nleqslv_3.0.3
## [19] BB_2014.10-1        numDeriv_2014.2-1  rootSolve_1.6.6
## [22] Deriv_3.7.0         knitr_1.13         Rmpi_0.6-5
##
## loaded via a namespace (and not attached):
##  [1] magrittr_1.5    lattice_0.20-33 quadprog_1.5-5  stringr_1.0.0
##  [5] highr_0.6       tools_3.2.5     grid_3.2.5      parallel_3.2.5
##  [9] formatR_1.4     evaluate_0.9    stringi_1.1.1   truncnorm_1.0-7
```

```r
devtools::session_info()
```

```
##  setting  value
##  version  R version 3.2.5 (2016-04-14)
##  system   x86_64, mingw32
##  ui       RTerm
##  language en
##  collate  Chinese (Simplified)_China.936
##  tz       Asia/Taipei
##  date     2016-08-26
##
```

```
## package      * version    date       source
## alabama      * 2015.3-1   2015-03-06 CRAN (R 3.2.2)
## BB           * 2014.10-1  2014-11-07 CRAN (R 3.2.2)
## DEoptim      * 2.2-3      2015-01-09 CRAN (R 3.2.2)
## Deriv        * 3.7.0      2016-04-05 CRAN (R 3.2.5)
## devtools       1.12.0     2016-06-24 CRAN (R 3.2.5)
## dfoptim      * 2016.7-1   2016-07-10 CRAN (R 3.2.5)
## digest         0.6.10     2016-08-02 CRAN (R 3.2.5)
## evaluate       0.9        2016-04-29 CRAN (R 3.2.3)
## formatR        1.4        2016-05-09 CRAN (R 3.2.5)
## GenSA        * 1.1.6      2016-02-15 CRAN (R 3.2.3)
## highr          0.6        2016-05-09 CRAN (R 3.2.5)
## knitr        * 1.13       2016-05-09 CRAN (R 3.2.3)
## lattice        0.20-33    2015-07-14 CRAN (R 3.2.5)
## magrittr       1.5        2014-11-22 CRAN (R 3.2.2)
## MASS         * 7.3-45     2015-11-10 CRAN (R 3.2.5)
## Matrix       * 1.2-6      2016-05-02 CRAN (R 3.2.5)
## memoise        1.0.0      2016-01-29 CRAN (R 3.2.5)
## neldermead   * 1.0-10     2015-01-11 CRAN (R 3.2.2)
## NlcOptim     * 0.4        2016-03-15 CRAN (R 3.2.5)
## nleqslv      * 3.0.3      2016-08-08 CRAN (R 3.2.5)
## numDeriv     * 2014.2-1   2015-05-04 CRAN (R 3.2.2)
## optextras    * 2016-8.8   2016-08-08 CRAN (R 3.2.5)
## optimbase    * 1.0-9      2014-03-02 CRAN (R 3.2.2)
## optimsimplex * 1.0-5      2014-02-02 CRAN (R 3.2.2)
## quadprog       1.5-5      2013-04-17 CRAN (R 3.2.2)
## Rcgmin       * 2013-2.21  2014-12-06 CRAN (R 3.2.2)
## Rdonlp2      * 3002.10    2014-11-11 R-Forge (R 3.2.5)
## Rmpi         * 0.6-5      2014-05-21 CRAN (R 3.2.3)
## Rnlminb2     * 3002.10    2014-11-11 R-Forge (R 3.2.5)
## rootSolve    * 1.6.6      2015-10-05 CRAN (R 3.2.3)
## Rsolnp       * 1.16       2015-12-28 CRAN (R 3.2.3)
## Rvmmin       * 2013-11.12 2014-12-06 CRAN (R 3.2.2)
## smco         * 0.1        2012-10-29 CRAN (R 3.2.3)
## stringi        1.1.1      2016-05-27 CRAN (R 3.2.5)
## stringr        1.0.0      2015-04-30 CRAN (R 3.2.2)
## truncnorm      1.0-7      2014-01-21 CRAN (R 3.2.2)
## withr          1.0.2      2016-06-20 CRAN (R 3.2.5)
```

参考文献

[1] 张平文、李铁军. 数值分析. 北京大学出版社, 2007.

[2] 吴勃英、王德明、丁效华、李道华. 数值分析原理. 科学出版社, 2003.

[3] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[4] W. La Cruz and M. Raydan. Nonmonotone spectral methods for large-scale nonlinear systems. *Optimization Methods and Software*, 18(5):583–599, 2003.

[5] W. La Cruz, J. M. Martínez, and M. Raydan. Spectral residual method without gradient information for solving large-scale nonlinear systems of equations. *Mathematics of Computation*, 75(255):1429, 2006.

[6] Ravi Varadhan and Christophe Roland. Simple and globally convergent methods for accelerating the convergence of any em algorithm. *Scandinavian Journal of Statistics*, 35(2):335–353, 2008.

[7] Ravi Varadhan and Paul Gilbert. BB: An R package for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function. *Journal of Statistical Software*, 32(4):1–26, 2009.

[8] Karline Soetaert. *rootSolve: Nonlinear Root Finding, Equilibrium and Steady-State Analysis of Ordinary Differential Equations*, 2015. R package version 1.6.6.

[9] Ravi Varadhan and Paul Gilbert. *BB: Solving and Optimizing Large-Scale Nonlinear Systems*, 2014. R package version 2014.10-1.

[10] Berend Hasselman. *nleqslv: Solve Systems of Nonlinear Equations*, 2016. R package version 3.0.2.

[11] Paulo Cortez. *Modern Optimization with R*. Springer International Publishing, 2014.

[12] John C. Nash. *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons, 2014.

[13] John Fox and Sanford Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, second edition, 2011.

[14] John C. Nash. *Rcgmin: Conjugate Gradient Minimization of Nonlinear Functions*, 2014. R package version 2013-2.21.

[15] John C. Nash. *Rvmmin: Variable Metric Nonlinear Function Minimization*, 2014. R package version 2013-11.12.

[16] Ravi Varadhan, Johns Hopkins University, Hans W. Borchers, and ABB Corporate Research. *dfoptim: Derivative-Free Optimization*, 2016. R package version 2016.7-1.

[17] Xianyan Chen and Xiangrong Yin. *NlcOptim: Solve Nonlinear Optimization with Nonlinear Constraints*, 2016. R package version 0.4.

[18] Douglas Bates and Deepayan Sarkar. *Rnlminb2: An R extension library for constrained optimization with nlminb.*, 2014. R package version 3002.10/r5962.

[19] Sebastien Bihorel and Michael Baudin. *optimbase: R port of the Scilab optimbase module*, 2014. R package version 1.0-9.

[20] Sebastien Bihorel and Michael Baudin. *optimsimplex: R port of the Scilab optimsimplex module*, 2014. R package version 1.0-5.

[21] Sebastien Bihorel and Michael Baudin. *neldermead: R port of the Scilab neldermead module*, 2015. R package version 1.0-10.

[22] Douglas Bates and Martin Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2016. R package version 1.2-6.