统计算法基础 Lab4

姓名：王凯栋　　学号：PB20071441　　日期：2023/5/15

目录

1.《统计计算》习题 6/22：对数似然函数为：

$$l(\theta_1,\theta_2)=\sum_{j=1}^{4} n_j ln\pi_j(\theta_1,\theta_2)$$

其中 $\pi_1(\theta_1,\theta_2) = 2\theta_1\theta_2, \pi_2(\theta_1,\theta_2) = \theta_1(2-\theta_1-2\theta_2), \pi_3(\theta_1,\theta_2) = \theta_2(2-\theta_2-2\theta_1), \pi_4(\theta_1,\theta_2) = (1-\theta_1-\theta_2)^2$ $n_1 = 17, n_2 = 182, n_3 = 60, n_4 = 176$

编写 R 程序，分别用 Newton 法，阻尼 Newton 法，BFGS 法，Fisher 得分法求极大似然估计。比较这几种方法的收敛速度。

优化问题为

$$argmax_{\theta_1,\theta_2}\ l(\theta_1,\theta_2)$$

$$\frac{\partial l(\theta_1,\theta_2)}{\partial\theta_1}=\frac{n_1}{\theta_1}+\frac{2n_2(1-\theta_1-\theta_2)}{\theta_1(2-\theta_1-2\theta_2)}-\frac{2n_3}{2-\theta_2-2\theta_1}-\frac{2n_4}{1-\theta_1-\theta_2}$$

$$\frac{\partial l(\theta_1,\theta_2)}{\partial\theta_2}=\frac{n_1}{\theta_2}-\frac{2n_2}{(2-\theta_1-2\theta_2)}+\frac{2n_3(1-\theta_1-\theta_2)}{(2-\theta_2-2\theta_1)\theta_2}-\frac{2n_4}{1-\theta_1-\theta_2}$$

$$\nabla^2 l(\theta_1,\theta_2)=\begin{pmatrix} \frac{-(n_1+n_2)}{\theta_1^2}-\frac{n_2}{(2-\theta_1-2\theta_2)^2}-\frac{4n_3}{(2-\theta_2-2\theta_1)^2}-\frac{2n_4}{(1-\theta_1-\theta_2)^2} & -\frac{2n_2}{(2-\theta_1-2\theta_2)^2}-\frac{2n_3}{(2-\theta_2-2\theta_1)^2}-\frac{2n_4}{(1-\theta_1-} \\ -\frac{2n_2}{(2-\theta_1-2\theta_2)^2}-\frac{2n_3}{(2-\theta_2-2\theta_1)^2}-\frac{2n_4}{(1-\theta_1-\theta_2)^2} & \frac{-(n_1+n_3)}{\theta_1^2}-\frac{n_3}{(2-\theta_2-2\theta_1)^2}-\frac{4n_2}{(2-\theta_1-2\theta_2)^2}- \end{pmatrix}$$

- Newton 法

```
n1 <- 17
n2 <- 182
n3 <- 60
n4 <- 176
iter <- 0# 初始化
theta <- c(0.1,0.1)# 初始值
track11 <- NULL # 用来记录迭代的轨迹
track12 <- NULL
track11[1]<- theta[1]
track12[2]<- theta[2]
theta_diff <- 1
# 梯度表示
gradient <- function(theta){
  partial1 <- n1/theta[1]+n2*(2-2*theta[1]-2*theta[2])/(theta[1]*(2-theta[1]-2*theta[2]
  partial2 <- n1/theta[2]+n3*(2-2*theta[1]-2*theta[2])/(theta[2]*(2-theta[2]-2*theta[1]
  return(c(partial1,partial2))
}



#Hessian 矩阵表示
Hessian <- function(theta){
  m11 <- n1/(-theta[1]^2)+n2*(-2*theta[1]*(2-theta[1]-2*theta[2])-4*(1-theta[1]-theta[2
  m22 <- n1/(-theta[2]^2)+n3*(-2*theta[2]*(2-theta[2]-2*theta[1])-4*(1-theta[2]-theta[1
  m12 <- n2*(-2)/(2-theta[1]-2*theta[2])^2+n3*(-2)/(2-theta[2]-2*theta[1])^2+n4*(-2)/(t
  m21 <- m12
  return(matrix(c(m11,m12,m21,m22),2,2))
}

#Newton 法迭代
while(theta_diff > 1e-3 & iter < 100){
  iter=iter+1
  option <- theta
```

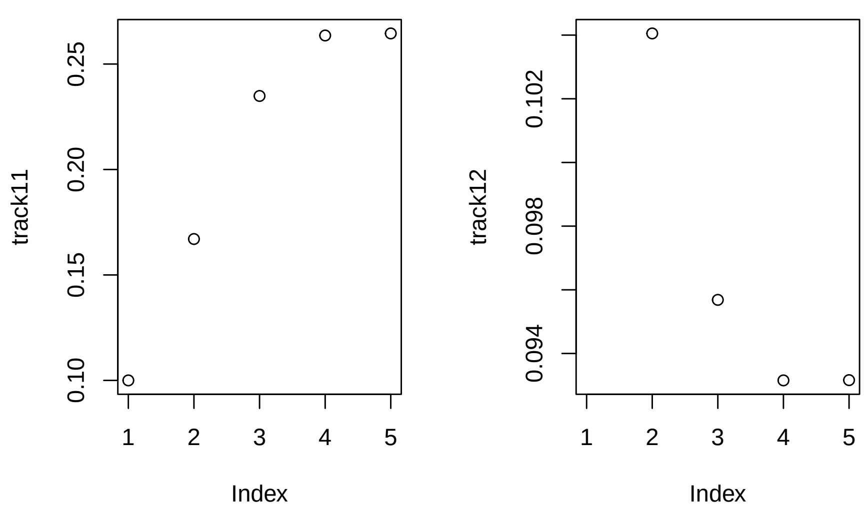
```
  theta = theta-solve(Hessian(theta))%*%gradient(theta)
  theta_diff = sum(abs(option-theta))
  track11[iter+1] <- theta[1]
  track12[iter+1] <- theta[2]
}
print(iter)# 打印迭代次数
```

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

```
print(theta)
```

```
##             [,1]
## [1,] 0.26450666
## [2,] 0.09316282
```

```
# 画出迭代轨迹
par(mfrow=c(1,2))
plot(track11)
plot(track12)
```



- 阻尼 Newton 法

```
theta <- c(0.1,0.1)
track21 <- NULL # 用来记录迭代的轨迹
track22 <- NULL
iter = 0
track21[1]<- theta[1]
track22[2]<- theta[2]
theta_diff <- 1
f <- function(theta){
  pi1=2*theta[1]*theta[2]
  pi2=theta[1]*(2-theta[1]-2*theta[2])
  pi3=theta[2]*(2-theta[2]-2*theta[1])
  pi4=(1-theta[1]-theta[2])^2
  value <- n1*log(pi1)+n2*log(pi2)+n3*log(pi3)+n4*log(pi4)
  return(value)
}


# 回溯直线法求最优步长
opt_step<- function(theta){
  alpha=0.5
  gama=0.8
  s=1
  h <- solve(Hessian(theta))%*%gradient(theta)
  while(f(theta-s*h)<f(theta)-alpha*s*t(gradient(theta))%*%h){
    s <- s*gama
  }
  return(s)
}

# 阻尼 Newton 法求解
while(theta_diff>1e-3 & iter < 100){
  option <- theta
```

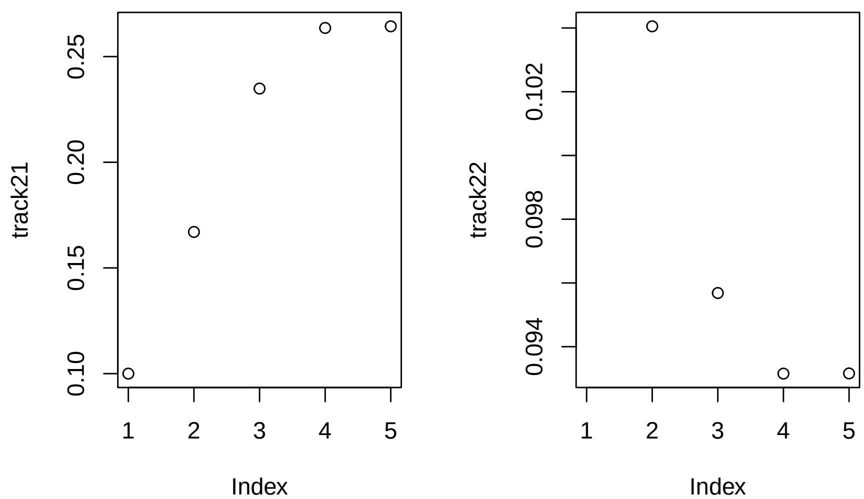
```
  iter=iter+1
  s <- opt_step(theta)# 求解最优步长
  theta = theta-s*solve(Hessian(theta))%*%gradient(theta)# 迭代
  theta_diff <- sum(abs(theta-option))
  track21[iter+1] <- theta[1]
  track22[iter+1] <- theta[2]
}
print(iter)# 打印迭代次数
```

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

```
print(theta)
```

```
##           [,1]
## [1,] 0.2643119
## [2,] 0.0931612
```

```
# 画出迭代轨迹
par(mfrow=c(1,2))
plot(track21)
plot(track22)
```



- BFGS 法

```
theta <- c(0.1,0.1)
V <- solve(Hessian(theta))# 只计算一遍 Hessian 矩阵即可
track31 <- NULL # 用来记录迭代的轨迹
track32 <- NULL
iter = 0
track31[1]<- theta[1]
track32[2]<- theta[2]
theta_diff <- 1


# 回溯直线法求最优步长
opt_step1<- function(theta,V){
  alpha=0.5
  gama=0.8
  s=1
  h <- V %*% gradient(theta)
```

```r
  while(f(theta-s*h)<f(theta)-alpha*s*t(gradient(theta))%*%h){
    s <- s*gama
  }
  return(s)
}




#BFGS 求解
while(theta_diff>1e-3 & iter < 100){
  iter=iter+1
  s <- opt_step1(theta,V)# 求解最优步长
  option <- theta
  theta = theta-s*V%*%gradient(theta)# 迭代
  delta= theta-option
  esi <- gradient(theta)-gradient(option)
  esi <- matrix(esi,2,1)
  part1 <- (as.numeric(t(esi)%*%delta+t(esi)%*%V%*%esi))*(delta%*%t(delta))/as.numeric(
  part2 <- ((V%*%esi%*%t(delta))+t(V%*%esi%*%t(delta)))/as.numeric(t(esi)%*%delta)
  V=V+part1-part2
  theta_diff <- sum(abs(theta-option))
  track31[iter+1] <- theta[1]
  track32[iter+1] <- theta[2]
}
print(iter)# 打印迭代次数
```

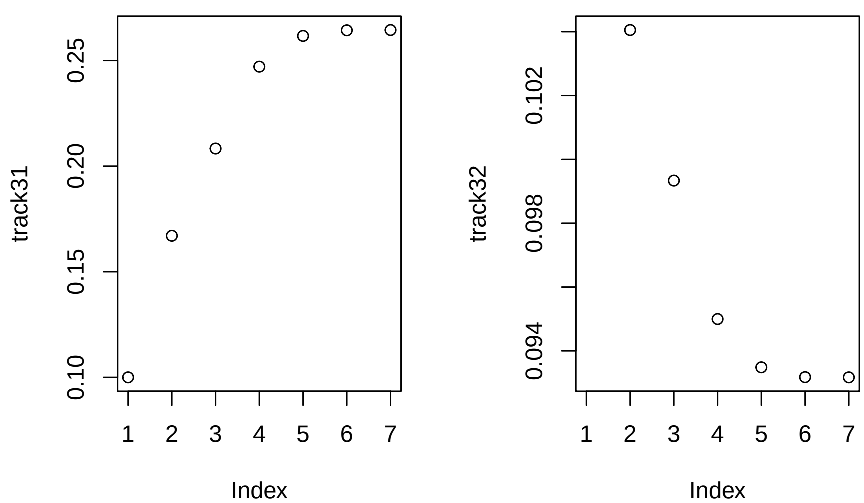
```
## [1] 6
```

```r
print(theta)
```

```
##            [,1]
## [1,] 0.26444347
```

```
## [2,] 0.09317058
```

```
# 画出迭代轨迹
par(mfrow=c(1,2))
plot(track31)
plot(track32)
```



上面已经算出了 Hessian 矩阵的表达式，对其求期望可得：

$$\nabla^2 l(\theta_1,\theta_2) = \begin{pmatrix} \frac{-(n_1+n_2)}{\theta_1^2} - \frac{n_2}{(2-\theta_1-2\theta_2)^2} - \frac{4n_3}{(2-\theta_2-2\theta_1)^2} - \frac{2n_4}{(1-\theta_1-\theta_2)^2} & -\frac{2n_2}{(2-\theta_1-2\theta_2)^2} - \frac{2n_3}{(2-\theta_2-2\theta_1)^2} - \frac{2n_4}{(1-\theta_1-} \\ -\frac{2n_2}{(2-\theta_1-2\theta_2)^2} - \frac{2n_3}{(2-\theta_2-2\theta_1)^2} - \frac{2n_4}{(1-\theta_1-\theta_2)^2} & \frac{-(n_1+n_3)}{\theta_1^2} - \frac{n_3}{(2-\theta_2-2\theta_1)^2} - \frac{4n_2}{(2-\theta_1-2\theta_2)^2} - (1 \end{pmatrix}$$

即对上面 Hessian 矩阵中元素，令 $n_1 = n\pi_1, n_2 = n\pi_2, n_3 = n\pi_3, n_4 = n\pi_4$

- Fisher 得分法

```
n <- 435
iter <- 0# 初始化
```

```
theta <- c(0.1,0.1)# 初始值
track41 <- NULL # 用来记录迭代的轨迹
track42 <- NULL
track41[1]<- theta[1]
track42[2]<- theta[2]
theta_diff <- 1
s=0.8

# 求 Fisher 信息阵
Fisher <- function(theta){
  Fish <- matrix(0,2,2)
  pi1=2*theta[1]*theta[2]
  pi2=theta[1]*(2-theta[1]-2*theta[2])
  pi3=theta[2]*(2-theta[2]-2*theta[1])
  pi4=(1-theta[1]-theta[2])^2
  Fish[1,1] <- -n*(pi1+pi2)/(theta[1]^2)-n*pi2/(2-theta[1]-2*theta[2])^2
  -4*n*pi3/(2-theta[2]-2*theta[1])^2-2*n*pi4/(1-theta[1]-theta[2])^2
  Fish[1,2] <- -2*n*pi2/(2-theta[1]-2*theta[2])^2-2*n*pi3/(2-theta[2]-2*theta[1])^2
  -2*n*pi4/(1-theta[1]-theta[2])^2
  Fish[2,1] <- Fish[1,2]
  Fish[2,2] <- -n*(pi1+pi3)/theta[2]^2-4*n*pi2/(2-theta[1]-2*theta[2])^2-n*pi3/(2-theta
  return(-Fish)
}



#Fisher 得分法法迭代
while(theta_diff > 1e-3 & iter < 1000){
  iter=iter+1
  option <- theta
  theta = theta+s*solve(Fisher(theta))%*%gradient(theta)
  theta_diff = sum(abs(option-theta))
  track41[iter+1] <- theta[1]
```

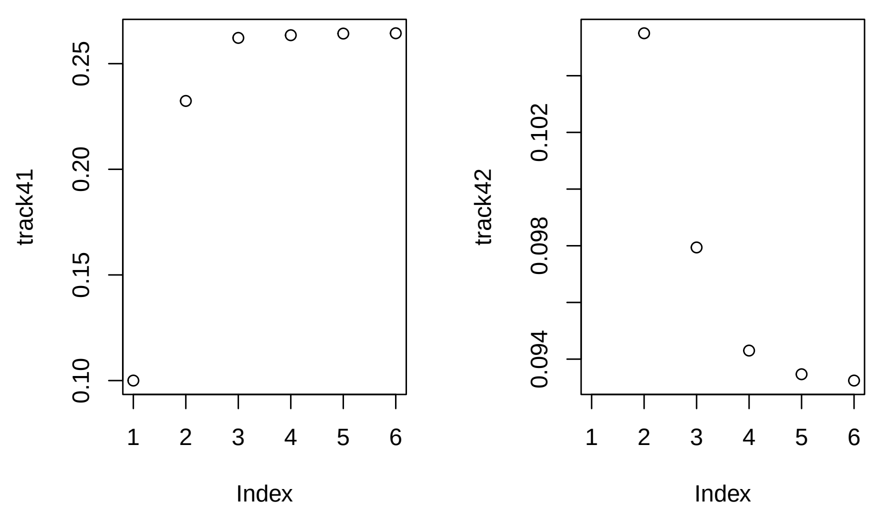
```
  track42[iter+1] <- theta[2]
}
print(iter)# 打印迭代次数
```

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

```
print(theta)
```

```
##            [,1]
## [1,] 0.26438681
## [2,] 0.09324377
```

```
# 画出迭代轨迹
par(mfrow=c(1,2))
plot(track41)
plot(track42)
```



2.《统计计算》习题 6/23:，这里

$$logit^{-1}(x) = \frac{exp(x)}{1+exp(x)}$$

解：$Y_i \sim B(m_i, logit^{-1}(\beta_0+\beta_1 x_i))$, 所以

$$P(Y_i = y_i) = \binom{m_i}{y_i} logit^{-1}(\beta_0+\beta_1 x_i)^{y_i}(1-logit^{-1}(\beta_0+\beta_1 x_i))^{m_i - y_i}$$

由 $Y_i, i = 1, 2, \cdots, 4$ 之间的独立性，所以对数似然函数为：

$$l(\beta, \beta_0) = \sum_{i=1}^{4} ln(P(Y_i = y_i))$$

去掉与参数无关的常数项后，对数似然函数化为:

$$l(\beta_0, \beta_1) = \sum_{i=1}^{4} y_i ln \frac{logit^{-1}(\beta_0+\beta_1 x_i)}{1-logit^{-1}(\beta_0+\beta_1 x_i)} + m_i ln(1-logit^{-1}(\beta_0+\beta_1 x_i))$$

由 $logit^{-1}(x) = \frac{exp(x)}{1+exp(x)}$，上式可进一步化为：

$$l(\beta_0, \beta_1) = \sum_{i=1}^{4} y_i(\beta_0+\beta_1 x_i) - m_i ln(1+exp(\beta_0+\beta_1 x_i))$$

所以，求其梯度，得：

$$\nabla l(\beta_0, \beta_1) = (\sum_{i=1}^{4} y_i - m_i logit^{-1}(\beta_0+\beta x_i), \sum_{i=1}^{4} x_i y_i - m_i x_i logit^{-1}(\beta_0+\beta_1 x_i))^T$$

$$\nabla^2 l(\beta_0, \beta_1) = \begin{pmatrix} \sum_{i=1}^{4} -m_i \frac{e^{\beta_0+\beta_1 x_i}}{(1+e^{\beta_0+\beta_1 x_i})^2} & \sum_{i=1}^{4} -m_i x_i \frac{e^{\beta_0+\beta_1 x_i}}{(1+e^{\beta_0+\beta_1 x_i})^2} \\ \sum_{i=1}^{4} -m_i x_i \frac{e^{\beta_0+\beta_1 x_i}}{(1+e^{\beta_0+\beta_1 x_i})^2} & \sum_{i=1}^{4} -m_i x_i^2 \frac{e^{\beta_0+\beta_1 x_i}}{(1+e^{\beta_0+\beta_1 x_i})^2} \end{pmatrix}$$

Newton 法迭代公式为：

$$(\beta_0^{(t+1)}, \beta_1^{(t+1)}) = (\beta_0^{(t)}, \beta_1^{(t)}) - \nabla^2 l(\beta_0^{(t)}, \beta_1^{(t)})^{-1} \nabla l(\beta_0^{(t)}, \beta_1^{(t)})$$

实现 Newton 法的具体 R 程序如下：

```r
# 定义值
m <- c(55,157,159,16)
x <- c(7,14,27,51)
y <- c(0,2,7,3)

# 常用函数
expit <- function(x){
  exp(x)/(1+exp(x))
}

expit2 <- function(x){
  exp(x)/(exp(x)+1)^2
}

# 定义初始值
iter <- 0
beta <- c(0,0)
track1 <- NULL
track2 <- NULL
track1[1] <- beta[1]
track2[1] <- beta[2]
tol <- 1e-4

# 求梯度
gradient <- function(beta){
  grad <- c(0,0)
  for(i in 1:4){
    grad[1] <- grad[1] + y[i]-m[i]*expit(beta[1]+beta[2]*x[i])
    grad[2] <- grad[2] + x[i]*y[i]-m[i]*x[i]*expit(beta[1]+beta[2]*x[i])
```

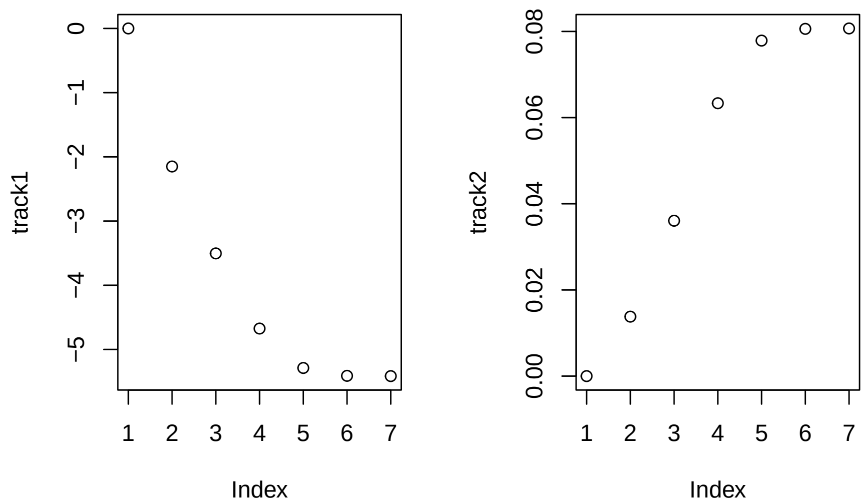
```r
  }
  return(grad)
}

# 求 Hessian 矩阵
Hessian <- function(beta){
  Hess <- matrix(0,2,2)
  Hess[1,1] <- sum(-m*expit2(beta[1]+beta[2]*x))
  Hess[1,2] <- sum(-m*x*expit2(beta[1]+beta[2]*x))
  Hess[2,1] <- Hess[1,2]
  Hess[2,2] <- sum(-m*x^2*expit2(beta[1]+beta[2]*x))
  return(Hess)
}

#Newton 法迭代
while(abs(gradient(beta)[1])> tol & abs(gradient(beta)[2])> tol & iter < 100){
  iter=iter+1
  beta = beta-solve(Hessian(beta))%*%gradient(beta)
  track1[iter+1] <- beta[1]
  track2[iter+1] <- beta[2]
}

print(iter)# 打印迭代次数
```

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

```r
print(beta)# 输出极大似然估计
```

```
##             [,1]
## [1,] -5.41517208
## [2,]  0.08069587
```

```
# 画出迭代轨迹
par(mfrow=c(1,2))
plot(track1)
plot(track2)
```



3. 课件 5 中的 Lasso Probelm：我们现在考虑一个具有稀疏结构的高维线性回归模型 (p>n):

$$y_i = x_i^T \beta + \epsilon_i, i = 1, \cdots, n$$

假设 $x_i \sim N(0,1)$, 回归系数 $\beta \in R^p$ 各项为 $\beta_j = 1/\sqrt{10}, j = 1, 2, \cdots, 10$ 以及 $\beta_j = 0, j = 11, \cdots, p$. 并且假设 $n = 100, p = 300$. 我们通过求解 Lasso 问题来估计回归系数：

$$min_{\beta \in R^p} \frac{1}{2n} ||Y_n - X_n \beta||_2^2 + \lambda ||\beta||_1$$

这里 $Y_n \in R^p, X_n \in R^{n \times p}$ 的每一行对应一对自变量与因变量样本 $(y_i, x_i), i = 1, \cdots, n$

利用 proximal gradient descent 算法来求解 lasso 问题。并利用 Lipschitz 常数取固定步长。这里令 $\lambda = 0.1, tol = 1e-2$, 最大迭代次数 $iter = 100$

Lipschitz 常数为：$L = (1/n)||X^TX||$

对应的 $prox_h(x) = S_\lambda(x)$ 是一个 *solf-thresholding operator*，定义为

$$[S_\lambda(x)]_j = sign(x_j)(|x_j| - \lambda)_+, j = 1, \cdots, p$$

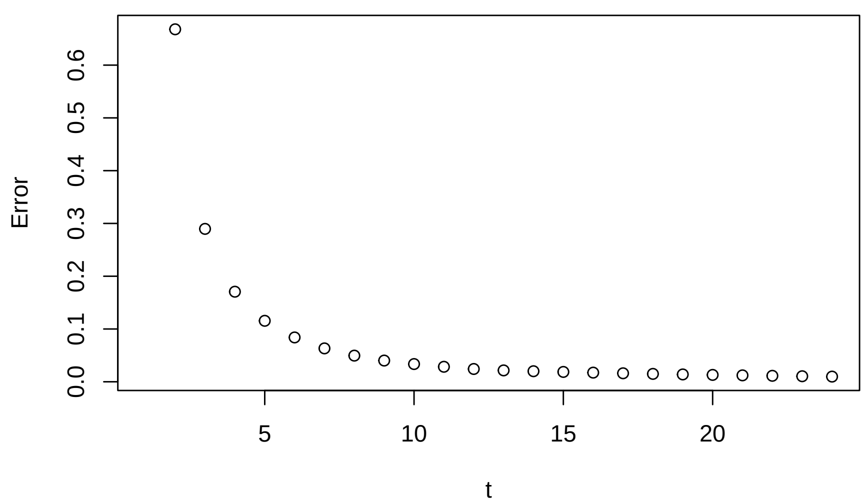
其中 $(x)_+ = max\{x, 0\}$

```
set.seed(1)
# 取初值
lambda = 0.1
tol = 1e-2
iter_max <- 100
n <- 100
p <- 300

# 样本与系数初值
X <- matrix(rnorm(n*p),ncol = p,nrow = n)
beta <- c(rep(1/sqrt(10),10),rep(0,p-10))
y <- X%*%beta+rnorm(n)

singluar <- (svd(t(X)%*%X)$d[1])/n# 利用 Lipschitz 常数取固定步长
s <- 1/singluar



# 函数定义
S_lambda <- function(x,lam){
  ifelse(abs(x)>lam,sign(x)*(abs(x)-lam),0)
}

b_old <- b_new <- rep(0,p)
b_diff <- 1
```

```r
b_diff_l <- c()
iter <- 0
#proximal gradient 求回归系数算法
while(b_diff > tol && iter <iter_max){
  iter <- iter+1
  b_new <- b_old + (1/n)*s*t(X)%*%(y- X%*%b_old)
  b_new <- S_lambda(b_new,lambda*s)
  b_diff <- max(abs(b_new-b_old))/max(abs(b_old))
  b_diff_l <- c(b_diff_l,b_diff)
  b_old <- b_new
}

print(iter)
```

```
## [1] 24
```

```r
plot(b_diff_l,xlab = "t",ylab = "Error")
```



```
cat("beta=",b_old)
```

```
## beta= 0.1484324 0.2089401 0.08216007 0.1298638 0.1149962 0.05173346 0.1911997 0.0878
```

4. (**Accelarated proximal gradient method**)Accelarated proximal gradient method 是 proximal gradient descent 算法的加速版本。一般地，对于优化问题：

$$min_{x \in R^p} g(x) + h(x)$$

where g is convex differentiable,and h is convex.Accelerated proximal gradient method 算法如下：

(i)$x_0 = x_{-1}$

(ii)For $k = 1, 2, \cdots$:

$$v = x_{k-1} + \frac{k-2}{k+1}(x_{k-1} - x_{k-2})$$

$$x_k = prox_{s_k h}(v - s_k \nabla g(v))$$

问题：

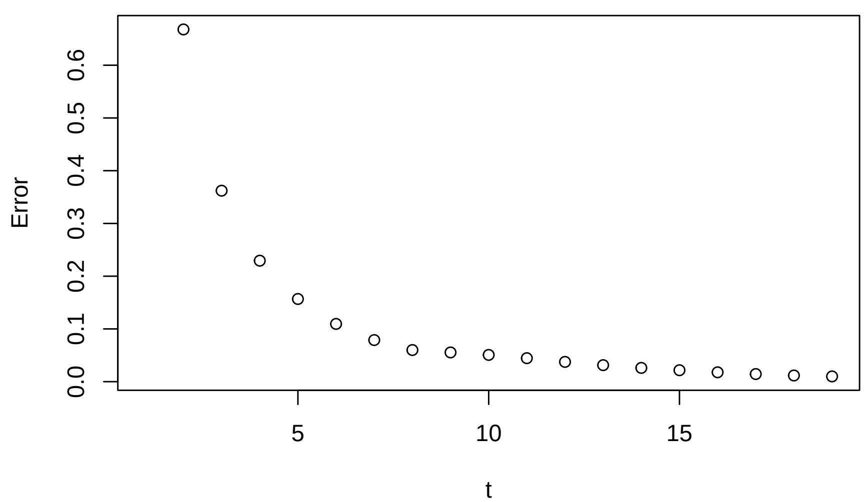
(a) 利用 Accelarated proximal gradient method 算法处理问题 3

```
set.seed(1)
b_old <- b_new <- rep(0,p)
b_diff <- 1
b_diff_l <- c()
iter <- 0
#Accelarated  proximal gradient method 求回归系数算法
while(b_diff >tol && iter <iter_max+1000){
  iter <- iter+1
  v <- b_new + (iter-2)/(iter+1)*(b_new-b_old)
```

```
  b_old <- b_new
  b_new <- v + (1/n)*s*t(X)%*%(y- X%*%v)
  b_new <- S_lambda(b_new,lambda*s)
  b_diff <- max(abs(b_new-b_old))/max(abs(b_old))
  b_diff_l <- c(b_diff_l,b_diff)
}

print(iter)
```

```
## [1] 19
```

```
plot(b_diff_l,xlab = "t",ylab = "Error")
```




```
cat("beta=",b_old)
```

```
## beta= 0.1676255 0.2744733 0.08213882 0.1236493 0.1144688 0.03486904 0.2186043 0.1091
```

(b) 按照问题 3 重复随机生成 50 组样本。对于每组样本使用 proximal gradient descent 和 Accelarated proximal gradient method，并记录收敛所用

的迭代次数，计算 50 组样本上的平均迭代次数，并比较。

```
#proximal gradient 求回归系数算法
proximal <- function(X,y){
  iter=0
  b_old <- b_new <- rep(0,p)
  b_diff <- 1
  singluar <- (svd(t(X)%*%X)$d[1])/n# 利用 Lipschitz 常数取固定步长
  s <- 1/singluar
  while(b_diff > tol && iter <iter_max){
  iter <- iter+1
  b_new <- b_old + (1/n)*s*t(X)%*%(y- X%*%b_old)
  b_new <- S_lambda(b_new,lambda*s)
  b_diff <- max(abs(b_new-b_old))/max(abs(b_old))# 相对大小作为收敛依据
  b_old <- b_new
  }
  return(iter)
}


#Accelarated  proximal gradient method 求回归系数算法
accelatrte <- function(X,y){
  iter=0
  b_old <- b_new <- rep(0,p)
  b_diff <- 1
  singluar <- (svd(t(X)%*%X)$d[1])/n# 利用 Lipschitz 常数取固定步长
  s <- 1/singluar
  while(b_diff >tol && iter <iter_max+1000){
  iter <- iter+1
  v <- b_new + (iter-2)/(iter+1)*(b_new-b_old)
  b_old <- b_new
  b_new <- v + (1/n)*s*t(X)%*%(y- X%*%v)
  b_new <- S_lambda(b_new,lambda*s)
  b_diff <- max(abs(b_new-b_old))/max(abs(b_old))
```

```
  }
  return(iter)
}



# 产生 50 组样本
X <- NULL
N=50
iter1 <- NULL
iter2 <- NULL

for(i in 1:N){
  X <- matrix(rnorm(n*p),ncol = p,nrow = n)
  y <- X%*%beta+rnorm(n)
  iter1[i] <- proximal(X,y)
  iter2[i] <- accelatrte(X,y)
}

cat("proximal gradient 平均迭代次数",mean(iter1),"\n")
```

```
## proximal gradient平均迭代次数 26.22
```

```
cat("Accelarated  proximal gradient method 平均迭代次数",mean(iter2),"\n")
```

```
## Accelarated  proximal gradient method平均迭代次数 22.82
```

可见：Accelarated proximal gradient method 迭代速度更快。

5. 考虑随机变量 $x \sim f(x|\theta^\star)$，其中 $\theta^\star \in R^p$ 是真实参数。对于样本 $x_1, x_2, \cdots, x_n$，对数似然函数为：

$$l(\theta) = \sum_{i=1}^{n} lnf(x_i|\theta)$$

在一些正则性条件下，Fisher 信息阵为

$$I_n(\theta) = -E\{\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}\}$$

说明：当样本量足够大且 θ_t 接近真实值 $\theta^\star$ 时，我们有近似结果：

$$I_n(\theta) \approx -\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}|_{\theta=\theta_t}$$

解：

因为：

$$I_n(\theta) = -E\{\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}\}$$

其中 $l(\theta)$ 是对数似然函数,θ 是参数。因此，Fisher 信息矩阵可以看作是 Hessian 矩阵的加权平均.

在 n 很大时，

$$E\{\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}\} = -I_n(\theta) \to \frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}|_{\theta=\theta^\star}$$

再有 θ_t 靠近 $\theta^\star$ 得

$$\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}|_{\theta=\theta^\star} \approx \frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}|_{\theta=\theta_t}$$

所以结合上面两个式子，可以得到：

$$I_n(\theta) \approx -\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^T}|_{\theta=\theta_t}$$

6. 假设 $x \in R^2$ 来自混合高斯模型：

$$x \sim \pi_1 N(\mu_1, \Sigma) + \pi_2 N(\mu_2, 2.25\Sigma) + \pi_3 N(\mu_3, 4\Sigma)$$

这里 $\pi_1 = 0.6, \pi_2 = 0.45, \pi_3 = 0.15, \mu_1 = (0,0)^T, \mu_2 = (5,5)^T, \mu_3 = (-1,5)^T, \Sigma = (\sigma_{ij}), \sigma_{11} = \sigma_{22} = 1$ 以及 $\sigma_{12} = \sigma_{21} = 0.5$, 生成 500 个服从上述混合高斯模型的样本。我们希望利用 EM 算法对样本进行类别

K=3 的聚类问题。初始值取值如下 $(\pi_1^{(0)},\pi_2^{(0)},\pi_2^{(0)}) = (0.5,0.4,0.1), \mu_1^{(0)} = (1,1)^T, \mu_2^{(0)} = (2,5)^T, \mu_1^{(0)} = (-2,-2)^T$,$\Sigma^{(0)} = I_2$，并考虑恰当的收敛准则。

(a) 利用 EM 算法估计混合高斯模型中的系数 $\theta = \{\pi_k, \mu_k, k = 1,2,3; \Sigma\}$。并比较与真实值之间的差异。

假设具有一个潜变量 $J_i \in \{1,2,3\}, Pr(J_i = k) = \pi_k$, 并且 $X_i|(J_i = k) \sim N(\mu_k, \Sigma_k)$

那么加入潜变量后对应的似然函数为：

$$
\begin{aligned}
l(\theta|X,J) &= \sum_{k=1}^{3}\sum_{i=1}^{m} I(J_i = k) log\{\pi_k f_k(X_i)\} \\
&= \sum_{k=1}^{3}\sum_{i=1}^{m} I(J_i = k) log(\pi_k) - \frac{1}{2}\sum_{k=1}^{3}\sum_{i=1}^{m} I(J_i = k)\{log|\Sigma_k| + (x_i-\mu_k)^T\Sigma_k^{-1}(x_i-\mu_k)\} + C
\end{aligned}
$$

- E step:

由于

$$
E(I(J_i = k)|X_i, \theta^{(t)}) = Pr(J_i = k|X_i, \theta^{(t)}) = \frac{\pi_k^{(t)} f_k(X_i)}{\sum_{k=1}^{3} \pi_k^{(t)} f_k(X_i)} := p_{ik}^{(t)}
$$

则 Q 函数为：

$$
Q(\theta|\theta^{(t)}) = \sum_{k=1}^{3}\sum_{i=1}^{m} p_{ik}^{(t)} log(\pi_k) - \frac{1}{2}\sum_{k=1}^{3}\sum_{i=1}^{m} p_{ik}^{(t)}\{log|\Sigma_k| + (x_i-\mu_k)^T\Sigma_k^{-1}(x_i-\mu_k)\} + C
$$

- M step:

$\theta^{(t+1)} = maxinum_\theta \; Q(\theta|\theta^{(t)})$

我们得到：

$$
\pi_k^{(t+1)} = \frac{1}{n}\sum_{i=1}^{m} p_{ik}^{(t)}, \mu_k^{(t+1)} = \frac{\sum_{i=1}^{m} p_{ik}^{(t)} X_i}{\sum_{i=1}^{m} p_{ik}^{(t)}}
$$

$$\Sigma^{(t+1)} = \sum_{i=1}^{m} p_{i1}^{(t)}(X_i - \mu_1^{(t+1)})(X_i - \mu_1^{(t+1)})^T) + \sum_{i=1}^{m} p_{i2}^{(t)}((X_i - \mu_2^{(t+1)})(X_i - \mu_2^{(t+1)})^T)/2.25 + \sum_{i=1}^{m} p_{i3}^{(t)}(X_i - \mu_3^{(t+1)})(X_i - \mu_3^{(t+1)})^T/4$$

```
library(MASS)
```

```
# 样本生成
m <- 500
pi_sa <- c(0.6,0.25,0.15)
mu1 <- c(0,0)
mu2 <- c(5,5)
mu3 <- c(-1,5)
Sigma <- matrix(c(1,0.5,0.5,1),2,2)
J=sample(1:3,size = m,replace = TRUE,prob = pi_sa)
x <- matrix(0,nrow = m,ncol = 2)
for(i in 1:m){
  if(J[i]==1) x[i,]=mvrnorm(1,mu1,Sigma)
  if(J[i]==2) x[i,]=mvrnorm(1,mu2,2.25*Sigma)
  if(J[i]==3) x[i,]=mvrnorm(1,mu3,4*Sigma)
}

head(x)
```

```
##            [,1]       [,2]
## [1,] -3.2860004  4.2804320
## [2,]  0.2514924 -1.3011569
## [3,] -0.1861334  0.9622731
## [4,] -0.6830064  1.3650112
## [5,]  1.6176215  1.7242064
## [6,] -2.9441810 -1.1570451
```

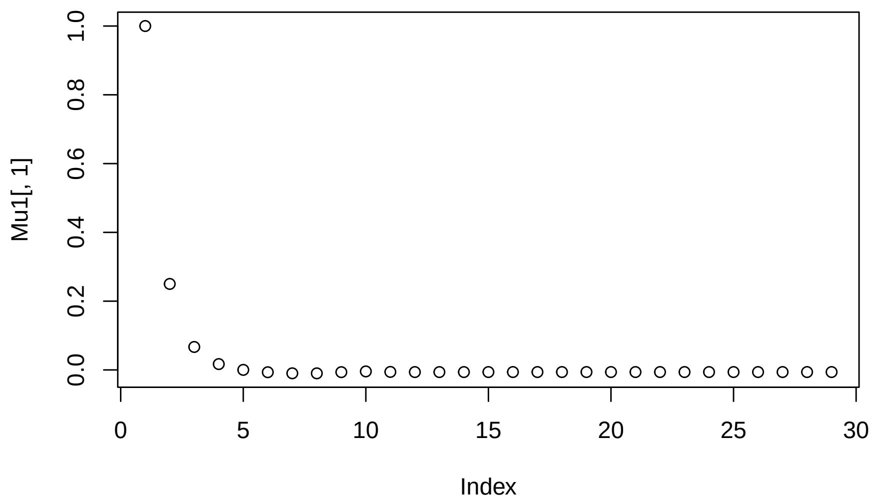
```
# 初始迭代值
Pi <- matrix(0,ncol = 3)
Mu1 <- matrix(0,ncol = 2)
Mu2 <- matrix(0,ncol = 2)
Mu3 <- matrix(0,ncol = 2)
Pi[1,] <- c(0.5,0.4,0.1)
rownames(Pi)[nrow(Pi)] <- nrow(Pi)
Mu1[1,] <- c(1,1)
Mu2[1,] <- c(2,5)
Mu3[1,] <- c(-2,-2)
Sigma1 <- matrix(c(1,0,0,1),2,2)
pi_diff  <- mu_diff <- sigma_diff <- 1


# 定义一些用到的函数
coverge_pi <- function(pi1,pi2){
  abs(sum((pi1-pi2)))
}

coverge_mu <- function(mu1,mu2){
  abs(sum(mu1-mu2))
}

coverge_sigma <- function(sigma1,sigma2){
  abs(sum(sigma1-sigma2))
}




library(mvtnorm)

####EM 算法
tol <- .Machine$double.eps^0.5
```

```
iter <- 0
iter_max <- 1000
while(pi_diff>tol | mu_diff>tol |sigma_diff>tol & iter < iter_max){
  iter <- iter+1
  option <- Sigma1
  #E step
  f1=f2=f3=py=qy=ry=NULL
  for(i in 1:m){
    f1[i] <- dmvnorm(t(x[i,]),t(Mu1[iter,]),Sigma1)*Pi[iter,1]
    f2[i] <- dmvnorm(t(x[i,]),t(Mu2[iter,]),2.25*Sigma1)*Pi[iter,2]
    f3[i] <- dmvnorm(t(x[i,]),t(Mu3[iter,]),4*Sigma1)*Pi[iter,3]
    py[i] <- f1[i]/(f1[i]+f2[i]+f3[i])
    qy[i] <- f2[i]/(f1[i]+f2[i]+f3[i])
    ry[i] <- f3[i]/(f1[i]+f2[i]+f3[i])
  }

  #M step
  new_row <- c(mean(py),mean(qy),mean(ry))
  Pi <- rbind(Pi,new_row)
  rownames(Pi)[nrow(Pi)] <- nrow(Pi)

  Mucompute <- function(py,x){
    Hq=c(0,0)
    for(i in 1:m){
      Hq=Hq+py[i]*as.vector(x[i,])
    }
    return(Hq/sum(py))
  }

  Mu1 <- rbind(Mu1,Mucompute(py,x))
  Mu2 <- rbind(Mu2,Mucompute(qy,x))
  Mu3 <- rbind(Mu3,Mucompute(ry,x))
  Sigma1=matrix(0,2,2)
```

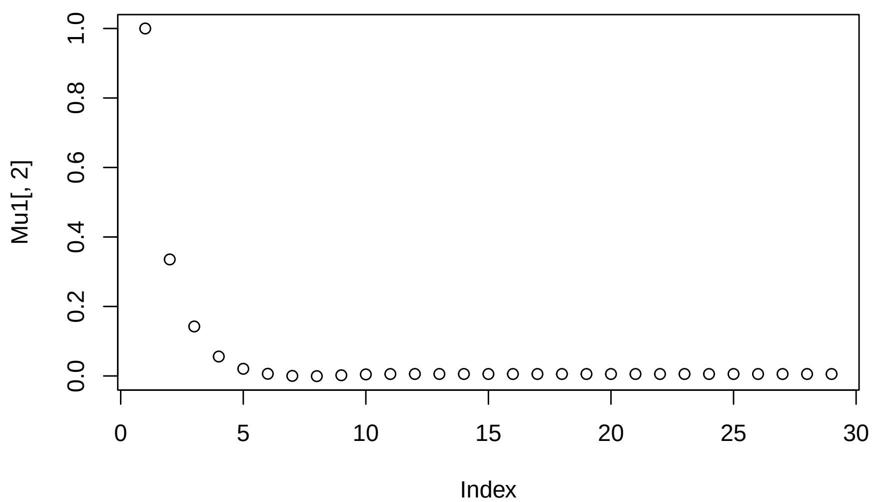
```
  for(i in 1:m){
    Sigma1 <- Sigma1+py[i]*tcrossprod(x[i,]-Mu1[iter+1,])+qy[i]/2.25*tcrossprod(x[i,]-M
  }
  Sigma1=Sigma1/m
  pi_diff <- coverge_pi(Pi[iter,],Pi[iter+1,])
  mu_diff <- coverge_mu(c(Mu1[iter,],Mu2[iter,],Mu3[iter,]),c(Mu1[iter+1,],Mu2[iter+1,]
  sigma_diff <- coverge_sigma(option,Sigma1)
}

print(list(pi=Pi[iter+1,],mu1=Mu1[iter+1,],mu2=Mu2[iter+1,],mu3=Mu3[iter+1,],
           sigma=Sigma1,iter=iter))
```

```
## $pi
## [1] 0.6143317 0.2207931 0.1648752
##
## $mu1
## [1] -0.006136316  0.005615981
##
## $mu2
## [1] 4.818490 4.942827
##
## $mu3
## [1] -1.292943  4.885536
##
## $sigma
##           [,1]      [,2]
## [1,] 0.9432325 0.4655727
## [2,] 0.4655727 1.0012151
##
## $iter
## [1] 28
```

```
plot(Mu1[,1])
```




```
plot(Mu1[,2])
```



可见，与真实值相差较小，且仅需 23 次迭代便可

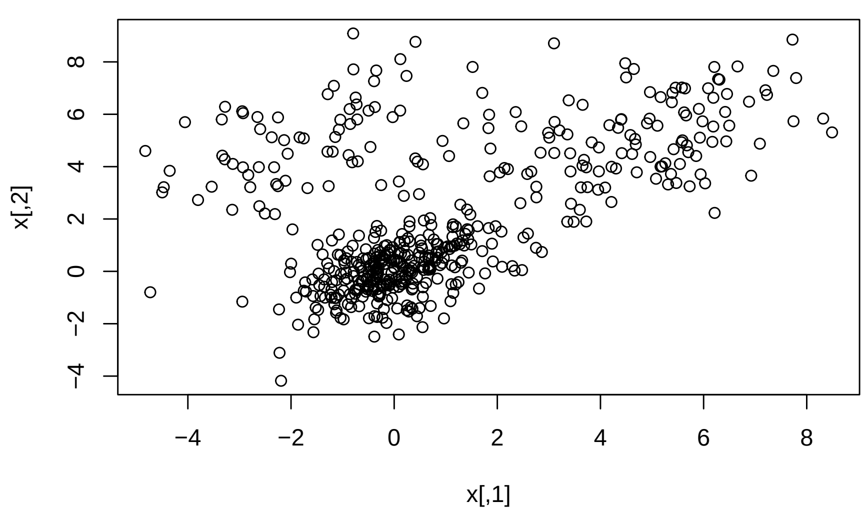
(b) 算法收敛后，利用 Bayes 准则给每个样本赋予标签：

$$y_i = argmax_{k=1,2,3} p_{ik}^{(t)}$$

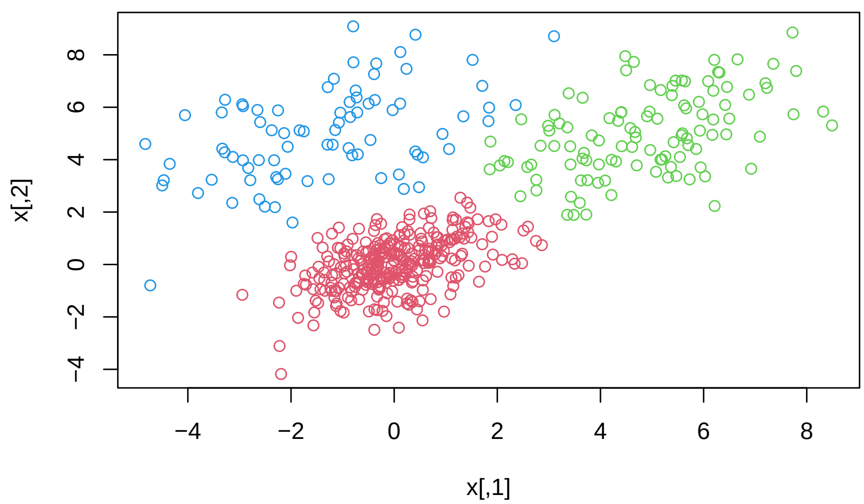
```
y <- NULL
for (i in 1:m){
  option <- max(py[i],qy[i],ry[i])
  if(option==py[i]) y[i]=1
  if(option==qy[i]) y[i]=2
  if(option==ry[i]) y[i]=3
}
```

(c) 做两幅散点图，一个是没有标签的样本散点图，一个是通过 EM 算法赋予标签后的样本散点图。观察聚类效果。

```
# 没有赋予标签的散点图
plot(x)
```



```
# 赋予标签的散点图
plot(x,col=y+1)
```



从图中可以看出，聚类效果较好。

7.《统计计算》习题 6/4

设 $f(x), g(x)$ 是定义在集合 A 上的两个密度，$f(x), g(x)$ 在 A 上都为正值。证明如下信息不等式：

$$\int_A [lnf(x)]f(x)dx \geq \int_A [ln(g(x))]f(x)dx$$

解：上面式子相当于证明交叉熵小于等于熵

不妨设 $X \sim f(x)$，则 $\int_A [lnf(x)]f(x)dx = Eln(f(X)), \int_A [lng(x)]f(x)dx = Eln(g(X))$ 所以上式等价于

$$\int_A ln\frac{g(x)}{f(x)}f(x)dx = Eln\frac{g(X)}{f(X)} \leq 0$$

由 ln(x) 凸性及 Jensen 不等式可得

$$Eln\frac{g(X)}{f(X)} \leq ln(E\frac{g(X)}{f(X)}) = ln\int_A \frac{g(x)}{f(x)}f(x)dx = ln1 = 0$$

所以上面的不等式成立。