Solution of Lasso Regression by Newton's method

Chen Zhiyuan

目录

1 函数编写与单次实验检验

编写的计算函数放在文件 czy_midfuncs.r 中，正文中只是调用；并展示在附录中。

1.1 初始化数据

```
library(mvtnorm)          # 用于生成多元正态分布的 package
# 真值参数初始化
n <- 100
p <- 6
e.sd <- 2
beta <- c(1, 0.8, 0.6, 0, 0, 0)
e <- rnorm(n, 0, e.sd)
x <- rmvnorm(n, rep(0,p), diag(p))
y <- x %*% beta + e
d = 0.01
thd2 = 1e-4
thd1 <- 1e-4
M <- 10000
# 初始化 beta0, 其值为最小二乘估计
# 不把 beta0 放在函数里面是为了重复模拟减少计算量，所以当函数用在其他数据集上时需要单独初始化
beta0 <- solve(t(x)%*%x)%*%t(x)%*%y
```

1.2 压缩估计：从 LQA 算法出发求解 Lasso

核心迭代式：

$$\beta^{(k+1)} = [X'X + n\Sigma_\lambda(\beta^{(k)})]^{-1}$$

```
source("czy_midfuncs.r")                              # 自编函数
opt.lam <- GCV.lambda(x, y, seq(0.1,1,0.01))          # 用 GCV 选出的最优值
LASSO.LQA(beta0, x, y, opt.lam, thd1, M, thd2)        # 在迭代过程中把 beta 分量压缩为零
```

```
##               [,1]
## [1,]  5.560450e-01
## [2,]  3.707721e-01
## [3,]  5.294939e-01
## [4,]  2.777636e-02
## [5,]  1.552943e-05
## [6,] -2.468598e-07
```

```
LASSO.LQA2(beta0, x, y, opt.lam, thd1, M, thd2)      # 在收敛后、输出前把 beta 分量压缩为零
```

```
##                [,1]
## [1,]  5.560450e-01
## [2,]  3.707721e-01
## [3,]  5.294939e-01
## [4,]  2.777636e-02
## [5,]  1.552943e-05
## [6,] -2.468598e-07
```

```
LASSO.MM(beta0, x, y, opt.lam, d, thd1, thd2)       # 不用 large value 代替被压缩为零的 beta 分量，而是在
```

```
##               [,1]
## [1,]  0.567062274
## [2,]  0.387944202
## [3,]  0.538560880
## [4,]  0.049429604
## [5,]  0.003174069
## [6,] -0.001326380
```

```
Adaptive.Lasso(beta0, x, y, opt.lam, d, thd1, 0.05)
```

```
##                [,1]
## [1,]  2.730254e-01
## [2,]  3.630798e-02
## [3,]  2.028867e-01
## [4,]  5.085652e-07
## [5,]  3.483121e-11
## [6,] -9.428361e-13
```

从少次重复的单次实验看：
(1) LQA 的两种实现方式结果没有区别。
(2) LQA 和 MM 的估计结果有差距，但并不大。
(3) Adaptive LASSO 的稳定性较差，跳出阈值需设定在 0.05~0.1 才会收敛，所以选择了理论上更稳定的 MM 来估计，并且规定最多迭代 70 次（LQA 也已经尝试过，在重复实验中稳定性确实不如 MM，在 0.01 水平尚不能收敛，此处不再展示。）；且结果与 LASSO 相差较大。

1.3 变量选择：最优子集选择与 AIC/BIC

AIC 与 BIC 准则：

$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2)$$
$$BIC = \frac{1}{n}(RSS + log(n)d\hat{\sigma}^2)$$

```
Best.Subset(x,y,criterion = 'AIC')
```

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

```
Best.Subset(x,y,criterion = 'BIC')
```

```
## [1] 1 2 3
```

从单次实验的结果看，基于 AIC 和 BIC 准则的最优子集选择模型效果相当好。

2 重复实验模拟

模拟 1000 次 Σ 为单位阵，ϵ 的方差为 2 的情况

2.1 样本量为 100

```
# 初始化
n <- 100
p <- 6
e.sd <- 2
x.sigma <- diag(p)
realbeta <- c(1, 0.8, 0.6, 0, 0, 0)
d = 0.01
thd2 = 1e-6
thd1 <- 1e-4
M <- 10000
lambdas <- seq(0.1, 0.7, 0.01)  # 从已经运行的实验中得出的该模拟条件下的经验范围，用以加快运行速度
rep.num <- 1000
# 模拟函数进行重复实验，只需指定 epsilon 的标准差，X 的协方差矩阵，
# 选择最优 lambda 的范围和 MM 估计中的参数 d
aaa <- simu(e.sd, x.sigma, lambdas, d)
print(as.matrix(aaa))
```

```
##           lasso   aic   bic adaptive
## correct   1.720 2.606 2.956    2.168
## incorrect 0.022 0.052 0.208    0.029
```

可以看到 lasso 的估计结果差强人意，adaptive lasso 的 correct 大于 lasso，但 incorrect 也大于 lasso。基于 aic 和 bic 准则的最优子集选择表现很棒，几乎能完全选出正确模型；而 BIC 无论是 correct 还是 incorrect 都高于 AIC，这是正常的，因为 BIC 的惩罚更重，更容易把系数压缩为零。

2.2 样本量为 200

```
n <- 200
bbb <- simu(e.sd, x.sigma, lambdas, d)
print(as.matrix(bbb))
```

```
##           lasso   aic   bic adaptive
## correct   1.797 2.574 2.975    1.950
## incorrect 0.002 0.001 0.029    0.001
```

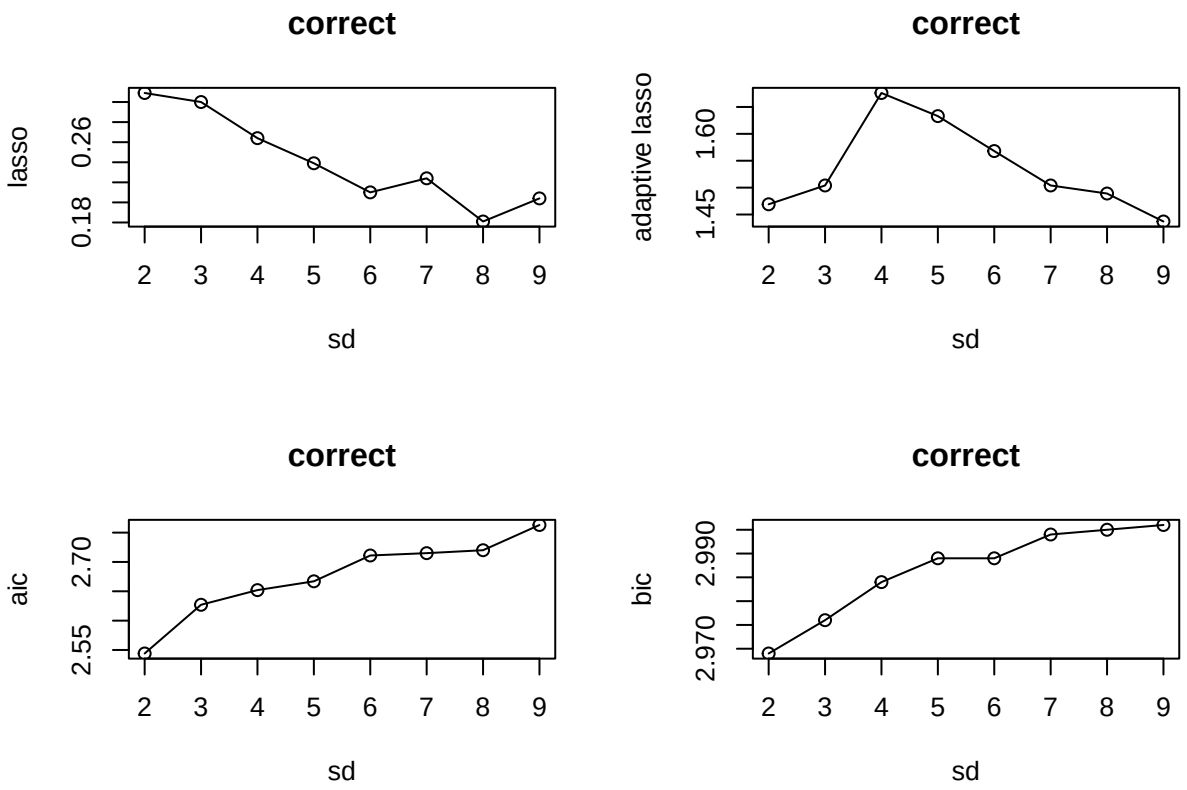
当样本量从 100 增加到 200 时，incorrect 的改善非常明显：四种方法的 incorrect 都减小了；而 lasso，AIC 和 BIC 的 correct 略有提高，adaptive lasso 的 correct 却下降了。

3 扩展条件的模拟

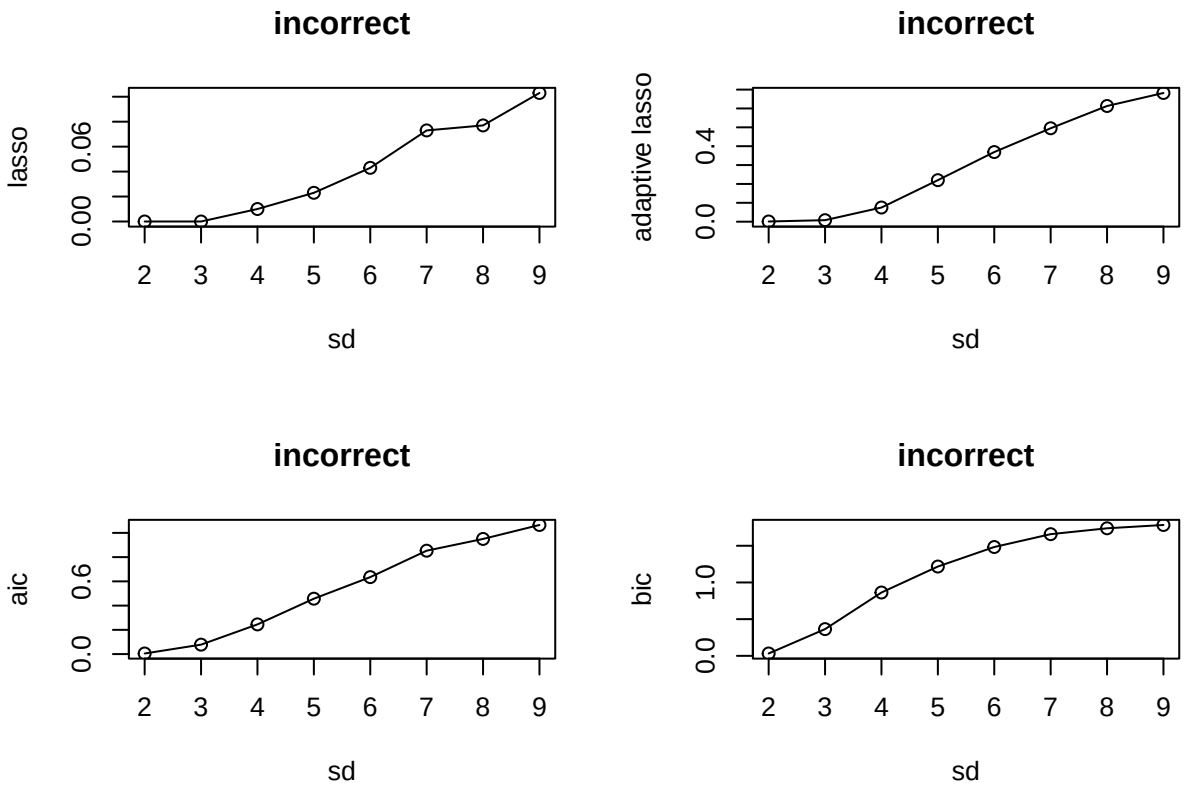
每次只改变一个条件，其他条件都沿用重复实验中的初始化条件（样本量沿用 n = 200）

3.1 当 epsilon 的方差更大时

```
e.sds <- seq(2, 9, 1)
correct.sd <- data.frame()
incorrect.sd <- data.frame()
for(new.e.sd in e.sds){
  aaa <- simu(new.e.sd, x.sigma, seq(0.1, 2, 0.01), 0.12)     # 根据提前模拟，标准差 =12 时最优 lambda 大
  correct.sd <- rbind.data.frame(correct.sd, aaa[1,])
  incorrect.sd <- rbind.data.frame(incorrect.sd, aaa[2,])
}
par(mfrow=c(2,2))
plot(e.sds, correct.sd$lasso, type = "o", xlab = "sd", ylab = "lasso", main = "correct")
plot(e.sds, correct.sd$adaptive, type = "o", xlab = "sd", ylab = "adaptive lasso", main = "correct")
plot(e.sds, correct.sd$aic, type = "o", xlab = "sd", ylab = "aic", main = "correct")
plot(e.sds, correct.sd$bic, type = "o", xlab = "sd", ylab = "bic", main = "correct")
```




```
par(mfrow=c(2,2))
plot(e.sds, incorrect.sd$lasso, type = "o", xlab = "sd", ylab = "lasso", main = "incorrect")
plot(e.sds, incorrect.sd$adaptive, type = "o", xlab = "sd", ylab = "adaptive lasso", main = "incorrec
plot(e.sds, incorrect.sd$aic, type = "o", xlab = "sd", ylab = "aic", main = "incorrect")
plot(e.sds, incorrect.sd$bic, type = "o", xlab = "sd", ylab = "bic", main = "incorrect")
```



(1) 首先遇到到的耐人寻味的问题是当方差增大时，R 会提醒我的 LASSO 的 $\beta^{(k+1)} = [X'X + n\Sigma_\lambda(\beta^{(k)})]^{-1}$ 部分逆不存在，这是由于我在 MM 算法中在分母上放置的 d 太小（0.01），有关 d 对 MM 估计的影响将在 3.3 中讨论。

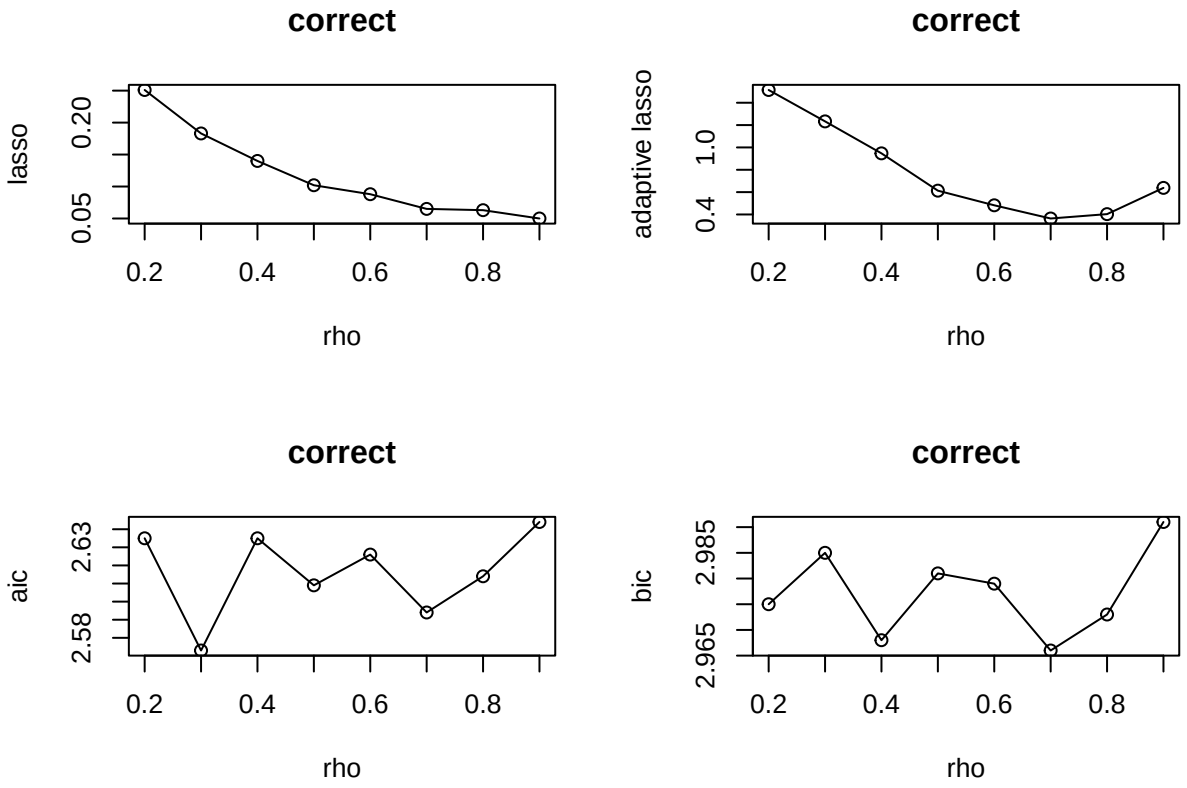
(2) 当方差增大时，lasso 和 adaptive lasso 的 correct 均呈波动下降趋势，但 adaptive lasso 的 correct 更高；AIC 和 BIC 的 correct 竟然呈波动上升趋势)。这说明在 epsilon 高方差时，基于 AIC 和 BIC 的最优子集选择方法有好的表现，而 adaptive lasso 要略优于 lasso。

(3) 当方差增大时，四种方法的 incorrect 都增大了，但是 adaptive lasso 比 lasso 更高，说明此时前者的压缩更狠；而 BIC 的 incorrect 却略低于 AIC，从这次实验的结果来看，基于 BIC 的最优子集选择应当是高方差情形下的最优方法。

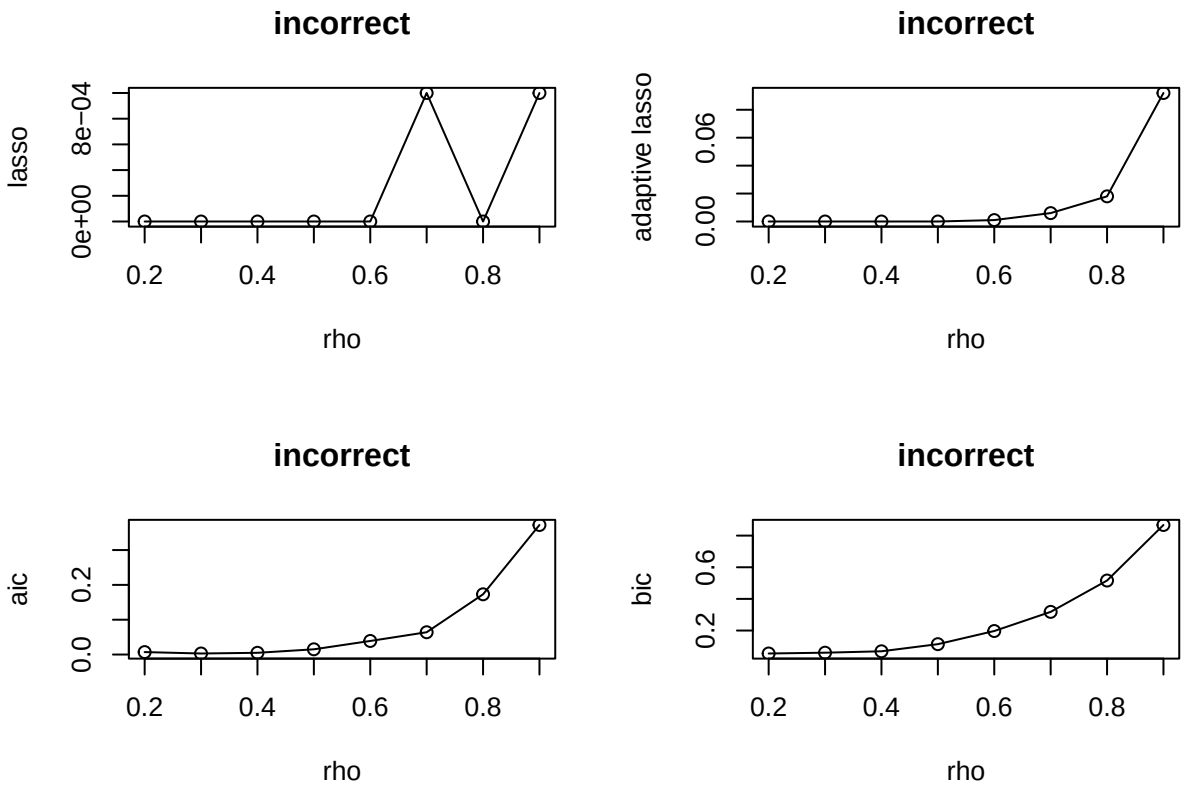
3.2 当 X 不独立且变量间相关系数增大时

```
correct.rho <- data.frame()
incorrect.rho <- data.frame()
rhos <- seq(0.2, 0.9, 0.1)
for(rho in rhos){
  x.sig <- matrix(rep(rho,p^2), p, p) - diag(rep(rho, p)) + diag(p)
  x <- rmvnorm(n, rep(0,p), x.sig)
```

```
  y <- x %*% beta + e
  aaa <- simu(e.sd, x.sig, seq(0.1, 2, 0.01), 0.1)
  correct.rho <- rbind.data.frame(correct.rho, aaa[1,])
  incorrect.rho <- rbind.data.frame(incorrect.rho, aaa[2,])
}
par(mfrow=c(2,2))
plot(rhos, correct.rho$lasso, type = "o", xlab = "rho", ylab = "lasso", main = "correct")
plot(rhos, correct.rho$adaptive, type = "o", xlab = "rho", ylab = "adaptive lasso", main = "correct")
plot(rhos, correct.rho$aic, type = "o", xlab = "rho", ylab = "aic", main = "correct")
plot(rhos, correct.rho$bic, type = "o", xlab = "rho", ylab = "bic", main = "correct")
```




```
par(mfrow=c(2,2))
plot(rhos, incorrect.rho$lasso, type = "o", xlab = "rho", ylab = "lasso", main = "incorrect")
plot(rhos, incorrect.rho$adaptive, type = "o", xlab = "rho", ylab = "adaptive lasso", main = "incorre
plot(rhos, incorrect.rho$aic, type = "o", xlab = "rho", ylab = "aic", main = "incorrect")
plot(rhos, incorrect.rho$bic, type = "o", xlab = "rho", ylab = "bic", main = "incorrect")
```



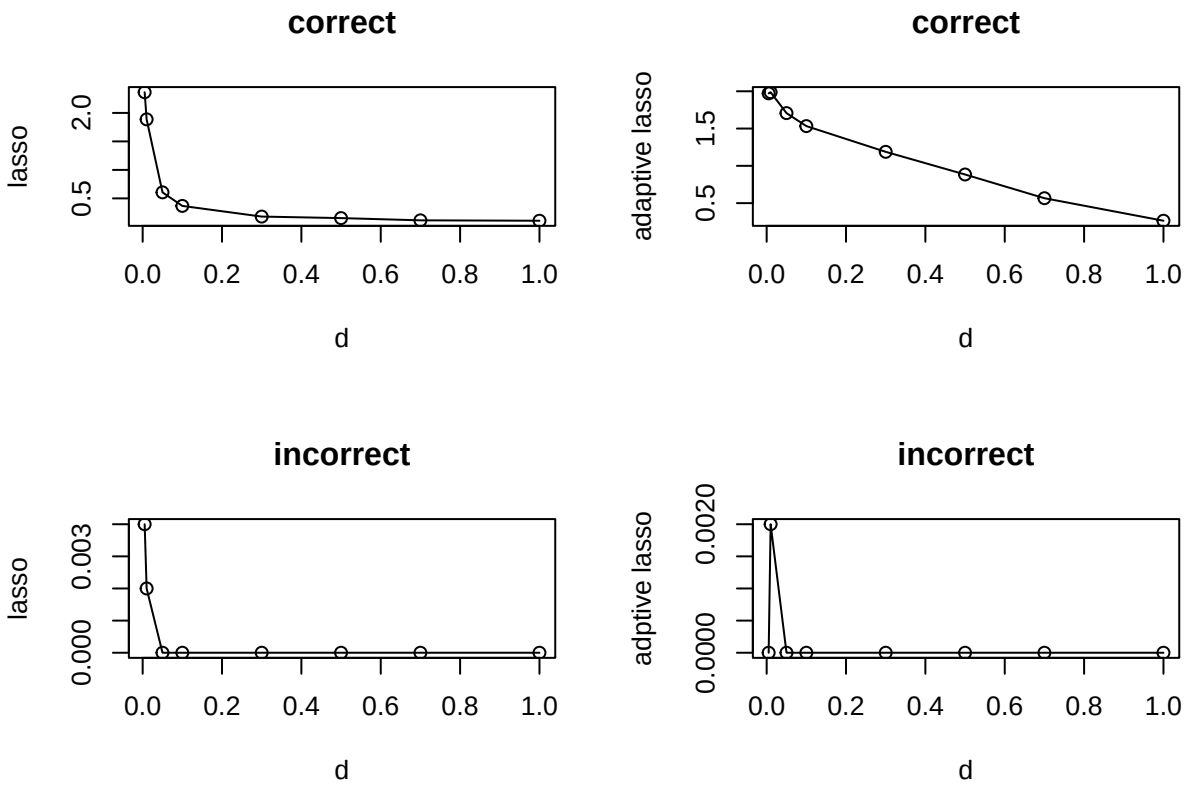
当 x 的变量间相关系数增大时：
(1) lasso 和 adaptive lasso 的 correct 都下降；AIC 和 BIC 的 correct 剧烈波动，看不出什么明显的趋势。(2) 四种方法的 incorrect 都上升，其中 AIC 和 BIC 均匀正常上升；lasso 和 adaptive lasso 则开始保持不变，当 rho $>$ 0.8 后突然大幅度上升。

3.3 当 d 变化时

d 是在 MM 算法中加在矩阵 $\Sigma_\lambda(\beta^{(k)})$ 对角线元素分母上的一个较小的常数，使得当被压缩为零的 β_k 分量代入对角线元素分母时，对角线元素的值不会过大而导致矩阵失真。下面来讨论 d 的大小变化对 Lasso 估计有何影响。

```
correct.d <- data.frame()
incorrect.d <- data.frame()
ds <- c(1, 0.7, 0.5, 0.3, 0.1, 0.05, 0.01, 0.005)
for(new.d in ds){
  aaa <- simu(e.sd, x.sigma, seq(0.1, 1, 0.01), new.d)
  correct.d <- rbind.data.frame(correct.d, aaa[1,])
  incorrect.d <- rbind.data.frame(incorrect.d, aaa[2,])
}
par(mfrow=c(2,2))
plot(ds, correct.d$lasso, type = "o", xlab = "d", ylab = "lasso", main = "correct")
```

```
plot(ds, correct.d$adaptive, type = "o", xlab = "d", ylab = "adaptive lasso", main = "correct")
plot(ds, incorrect.d$lasso, type = "o", xlab = "d", ylab = "lasso", main = "incorrect")
plot(ds, incorrect.d$adaptive, type = "o", xlab = "d", ylab = "adptive lasso", main = "incorrect")
```



3.3.1 结果讨论

可以看到当 d 变大时:
(1) lasso 的 correct 和 incorrect 都迅速下降然后趋于不变，这说明当 d $>$ 0.2 时 MM 算法估计的 lasso 结果是失真的。
(2) adaptive lasso 的 correct 呈现均匀下降的趋势，incorrect 在剧烈波动后也立刻下降然后不变。在 correct 上可能是 adaptive 的自我调节性质起了对抗失真的作用，但在 incorrect 上依然失真。

3.3.2 结论

在 3.1 改变方差的实验中已经发现当 d 太小时算法可能不时出现不收敛的情况，给我们的实验带来麻烦。而 3.3 的结果告诉我们：即使如此也不能把实验中的 d 设置得太大，否则当 correct 和 incorrect 都失真时（比如 incorrect 全为 0）也看不出任何结果。
所以，我们需要选出一个大小适中的 d 使得 3.1 和 3.2 的实验能正常运行。回到之前的结果，我们从初始条件的单次实验知道 0.01 是一个估计不错的 d，当需要增加它时，可以先尝试 0.02，0.03，0.05 和 0.1，一定不要超过 0.2。

3.4 在非设计数据上的模拟：与标准结果对比

为了测试编写的函数是否具有良好的泛化能力，决定在经典的波士顿房价数据集上检验其选择变量的效果，而标准结果则参考 glmnet 包的 lasso 和最优子集选择函数。

3.4.1 导入数据

设计阵不包括 chas 这一类别变量，因为自己编写的函数中没有专门设置虚拟变量；同时添加一列全为 1 的向量用于估计截距项（仅用于自编 lasso）。

3.4.2 自编函数的选择变量结果

```
# 用自编 lasso 等函数进行变量选择
opt.lam <- GCV.lambda(x,y,seq(0.1,2,0.01))
beta0.boston <- solve(t(x.boston.in)%*%x.boston.in)%*%t(x.boston.in)%*%y.boston
abs(LASSO.MM(beta0.boston, x.boston.in, y.boston, opt.lam, d, thd1, thd2)) < 0.005
```

```
##           [,1]
## crim    FALSE
## zn      FALSE
## indus    TRUE
## nox      TRUE
## rm      FALSE
## age      TRUE
## dis     FALSE
## rad     FALSE
## tax     FALSE
## ptratio FALSE
## black   FALSE
## lstat   FALSE
##         FALSE
```

```
Best.Subset(x.boston,y.boston,criterion = 'AIC')
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
Best.Subset(x.boston,y.boston,criterion = 'BIC')
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

自编的 Lasso 函数将 indus，nox 和 age 三个变量的系数估计为零，而基于 AIC 和 BIC 的最优子集选择则没有丢弃任何一个变量。

3.4.3 库函数的变量选择结果

```
# 使用 glmnet 和 leaps 库的参考结果
# lasso
lso.cv <- cv.glmnet(x.boston, y.boston, alpha=1)
lso.fit <- glmnet(x.boston, y.boston, alpha=1, lambda = lso.cv$lambda.min)
lso.fit$beta
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## crim     -0.099646779
## zn        0.039533857
## indus     .
## nox     -14.945043922
## rm        3.935451551
## age       .
## dis      -1.387459507
## rad       0.248655701
## tax      -0.009893485
## ptratio  -0.953834087
## black     0.009338204
## lstat    -0.527979243
```

```
# best subset
full.fit <- regsubsets(medv~.-chas, Boston, nvmax = 12)
full.summary <- summary(full.fit)
cat(" 根据 BIC 最小选出的最优模型个数为: ", which.min(full.summary$bic))
```

```
## 根据BIC最小选出的最优模型个数为:  10
```

```
print(full.summary$outmat)
```

```
##           crim zn  indus nox rm  age dis rad tax ptratio black lstat
## 1  ( 1 )  " "  " " " "   " " " " " " " " " " " "     " "   "*"
## 2  ( 1 )  " "  " " " "   " " "*" " " " " " " " " "     " "   "*"
## 3  ( 1 )  " "  " " " "   " " "*" " " " " " " " " "*"     " "   "*"
## 4  ( 1 )  " "  " " " "   " " "*" " " "*" " " " " "*"     " "   "*"
## 5  ( 1 )  " "  " " " "   "*" "*" " " "*" " " " " "*"     " "   "*"
## 6  ( 1 )  " "  " " " "   "*" "*" " " "*" " " " " "*"     "*"   "*"
## 7  ( 1 )  " "  "*" " "   "*" "*" " " "*" " " " " "*"     "*"   "*"
## 8  ( 1 )  "*"  " " " "   "*" "*" " " "*" "*" " " "*"     "*"   "*"
## 9  ( 1 )  "*"  " " " "   "*" "*" " " "*" "*" "*" "*"     "*"   "*"
## 10  ( 1 ) "*"  "*" " "   "*" "*" " " "*" "*" "*" "*"     "*"   "*"
```

```
## 11  ( 1 ) "*"  "*" "*"   "*" "*" " " "*" "*" "*" "*"     "*"   "*"
## 12  ( 1 ) "*"  "*" "*"   "*" "*" "*" "*" "*" "*" "*"     "*"   "*"
```

使用库函数的 lasso 将 indus 和 age 估计为零；而当 BIC 最小时的最优子集选择模型含变量数为 10 个，根据结果可知丢弃的变量为 indus 和 age。这一结果和自编 lasso 相当接近。
在 Boston 数据集上的测试表示：尽管 BIC 在先前实验中的 correct 和 incorrect 都表现得比 lasso 更好，但在这个实际问题的估计中 lasso 和参考结果更接近。

4 参考文献

[1] 李高荣, 吴密霞编著. 多元统计分析. 科学出版社, 2021.

5 附录

5.1 Lasso 的实现

5.1.1 用 GCV 准则选择最优

```
GCV.lambda <- function(x, y, lambdas){
  # 此方法用 GCV 方法选出最优的 lambdas (只能选出一个)
  # 传入的 lambda 为一个范围向量
  p <- dim(x)[2]
  GCV <- vector(length = length(lambdas))
  q = 1
  for(lambda in lambdas){
    beta.gcv <- LASSO.MM(beta0, x, y, lambda, d, thd1, thd2)                # 对输入的 lambda 拟合 lass
    P <- x %*% solve(t(x)%*%x + n*diag(as.vector(lambda/abs(beta.gcv)))) %*% t(x)    # 计算帽子矩阵
    d <- sum(diag(P))                                                                # 计算帽子矩阵的迹
    e2 <- sum((y-x%*%beta.gcv)^2)                                                    # 计算 lasso 估计
    gcv <- e2 / (1-d/n)^2                                                            # 计算 GCV
    GCV[q] <- gcv
    q = q + 1
  }
  bestlambda <- lambdas[which.min(GCV)]                                              # 找出使 GCV 最小
  return(bestlambda)
}
```

5.1.2 LQA 算法：两种实现

```
LASSO.LQA <- function(beta0, x, y, lambda, thd1, M, thd2){
  # 用 LQA 算法计算 lasso 的数值解
  # beta0 为初始系数向量 (p 维), n 为样本量, p 为变量维数, lambda 为调节参数的待选范围, thd1 为将   分量压缩为
  # 在迭代过程中将   小于阈值的分量压缩为零
  n <- dim(x)[1]
  p <- dim(x)[2]
  sigma.func <- function(beta){                    # Σ 矩阵
    beta.copy = beta
    for(i in 1:p){
      if(beta.copy[i] == 0) beta.copy[i] <- M      # 为使矩阵逆存在, 将 1/0 用一个很大的数 M 代替
      else beta.copy[i] <- 1/abs(beta.copy[i])
    }
    sigma <- diag(as.vector(lambda*beta.copy))     # 对角线为 lambda_i / |beta|
```

```
    return(sigma)
  }

  beta <- beta0                                    # 初值为最小二乘估计
  k = 0
  repeat{
    sigma <- sigma.func(beta)
    newbeta <- solve( t(x)%*%x + n*sigma ) %*% t(x) %*% y  # 计算 beta_k+1
    for(beta.i in newbeta){                                # 当其分量足够小时将其压缩为 0
      if(abs(beta.i) < thd1) beta.i <- 0
    }
    if(t(newbeta-beta)%*%(newbeta-beta) < thd2) break      # 当两次迭代的向量差距（差的二范数）足够小时停
    beta <- newbeta
    k = k + 1
    #cat("\r", k)                                         # 输出迭代次数
  }
  return(beta)
}

LASSO.LQA2 <- function(beta0, x, y, lambda, thd1, M, thd2){
  # 此函数在迭代过程中不将   小于阈值的分量压缩为零，而在收敛后再将小于阈值的分量压缩为零
  # 其余部分与 LASSO.LQA 相同
  n <- dim(x)[1]
  p <- dim(x)[2]
  sigma.func <- function(beta){
    sigma <- diag(as.vector(lambda/abs(beta)))
    return(sigma)
  }
  beta <- beta0
  k = 0
  repeat{
    sigma <- sigma.func(beta)
    newbeta <- solve( t(x)%*%x + n*sigma ) %*% t(x) %*% y
    if(t(newbeta-beta)%*%(newbeta-beta) < thd2) break
    beta <- newbeta
    k = k + 1
    #cat("\r", k)
  }
  for(beta.i in newbeta){                                # 收敛后判断：当其分量足够小时将其压缩为 0，再
    if(abs(beta.i) < thd1) beta.i <- 0
  }
```

```
  return(beta)
}
```

5.1.3 MM 算法：一点点优化

```
LASSO.MM <- function(beta0, x, y, lambda, d, thd1, thd2){
  # 此函数在迭代过程中将   小于阈值的分量压缩为零，但不将 1/|_k| 替换为 M，而是在分母上加上一个较小的数 d，使得
  # 稳定性更好
  n <- dim(x)[1]
  p <- dim(x)[2]
  sigma.func <- function(beta){
    beta.copy = beta
    for(i in 1:p){
      beta.copy[i] <- 1/(abs(beta.copy[i]) + d)           # 在分母上加上一个较小数 d 以保证矩阵可逆
    }
    sigma <- diag(as.vector(lambda*beta.copy))
    return(sigma)
  }
  beta <- beta0
  k = 0
  repeat{
    sigma <- sigma.func(beta)
    newbeta <- solve( t(x)%*%x + n*sigma ) %*% t(x) %*% y
    for(beta.i in newbeta){
      if(abs(beta.i) < thd1) beta.i <- 0                          # 当其分量足够小时将其压缩为 0
    }
    if(t(newbeta-beta)%*%(newbeta-beta) < thd2) break
    beta <- newbeta
    k = k + 1
    #cat("\r", k)
  }
  return(beta)
}
```

5.1.4 Adaptive Lasso

```
Adaptive.Lasso <- function(beta0, x, y, lambda0, d, thd1, thd2){
  # 用 MM 算法计算 adaptive lasso 的数值解，因为相对 LQA 更稳定
  # 这里用每次迭代的 1/|_k| 作为计算 _k+1 时的权重而非一直用 LS，是考虑到 LS 可能估计不好，而每次迭代的估计会
```

```
  # 指数姑且取 1
  n <- dim(x)[1]
  p <- dim(x)[2]
  sigma.func <- function(beta){
    lambda <- lambda0/abs(beta)                          # 用每次迭代的 1/|_k| 代替固定的一个 lambda 值
    beta.copy = beta
    for(i in 1:p){
      beta.copy[i] <- 1/(abs(beta.copy[i]) + d)
    }
    sigma <- diag(as.vector(lambda*beta.copy))
    return(sigma)
  }
  beta <- beta0
  k = 0
  for(l in 1:70){                                        # 因为 adaptive lasso 在实际中更难收敛，故设定迭
    sigma <- sigma.func(beta)
    newbeta <- solve( t(x)%*%x + n*sigma ) %*% t(x) %*% y
    for(beta.i in newbeta){
      if(abs(beta.i) < thd1) beta.i <- 0
    }
    if(t(newbeta-beta)%*%(newbeta-beta) < thd2) break
    beta <- newbeta
    k = k + 1
    #cat("\r", k)
  }
  return(beta)
}
```

5.2 最优子集选择：基于 AIC 与 BIC

```
Best.Subset <- function(x, y, criterion){
  # 用最优子集选择进行变量选择
  # criterion 是可选判断准则，此函数提供 AIC 和 BIC
  # x[,i] 代表列值，即 Xi
  n <- dim(x)[1]
  p <- dim(x)[2]
  models <- list()
  for(i in 1:p){
    # 拟合所有包含 i 个变量的模型，用 RSS 最小准则选出最优的 model_i
    choice <- combn(p, i)                              # 所有组合
```

```
    RSS1 <- vector(length = choose(p,i))          # 存放所有组合拟合结果
    for(j in choose(p,i)){
      x1 <- x[,choice[,j]]
      e1 <- y - x1 %*% solve(t(x1)%*%x1) %*% t(x1) %*% y
      rss1 <- t(e1) %*% e1
      RSS1[j] <- rss1                             # 计算所有组合的 rss 并存放 (同时得到了全模型 RSS, 后面
    }
    bestchoice <- choice[,which.min(RSS1)]        # 按 rss 最小选出最优 model_i
    models[[i]] <- bestchoice[1:i]
  }
  # 计算 p 个最优 model_i 的 RSS 和   方差估计, 为计算 AIC、BIC 做准备
  RSS2 <- vector(length = p)
  D2 <- 1:p
  q = 1
  for(model in models){
    x2 <- x[,model]
    d = length(model)
    e2 <- y - x2 %*% solve(t(x2)%*%x2) %*% t(x2) %*% y    # 残差
    rss2 <- t(e2) %*% e2                                  # 部分变量模型的 RSS
    RSS2[q] <- rss2
    q = q + 1
  }
  SIG2 <- rep(RSS2[p]/(n-p), p)
  if(criterion=='AIC'){
    AIC <- (RSS2 + 2*D2*SIG2)/(n*SIG2)
    bestmodel <- models[[which.min(AIC)]]
  }

  if(criterion=='BIC'){
    BIC <- (RSS2 + log(n)*D2*SIG2)/n
    bestmodel <- models[[which.min(BIC)]]
  }

  return(bestmodel)
}
```

5.3 模拟函数：为减少正文代码量而存在

```
simu <- function(e.sd, x.sigma, lambdas, d)
{
```

```r
# 只设计了 epsilon 的标准差，X 的协方差矩阵，选择最优 lambda 的范围和 MM 估计中的参数 d 作为可改变变量
# 因为接下来的不同条件的实验只会改变这些参数
# 其他参数指定为全局变量
correct.lasso.lqa <- c()
incorrect.lasso.lqa <- c()
correct.lasso.mm <- c()
incorrect.lasso.mm <- c()
correct.bic <- rep(0, rep.num)
incorrect.bic <- rep(0, rep.num)
correct.aic <- rep(0, rep.num)
incorrect.aic <- rep(0, rep.num)
correct.adaptivelasso <- c()
incorrect.adaptivelasso <- c()
for(i in 1:rep.num){
  # 更新
  e <- rnorm(n, 0, e.sd)
  x <- rmvnorm(n, rep(0,p), x.sigma)
  colnames(x) <- c("x1","x2","x3","x4","x5","x6")
  y <- x %*% realbeta + e
  beta0 <- solve(t(x)%*%x)%*%t(x)%*%y
  opt.lam <- GCV.lambda(x,y,lambdas)
  # mm
  correct.lasso.mm[i] <- sum(abs(LASSO.MM(beta0, x, y, opt.lam, d, thd1, thd2))[4:6]<0.005)
  incorrect.lasso.mm[i] <- sum(abs(LASSO.MM(beta0, x, y, opt.lam, d, thd1, thd2))[1:3]<0.005)
  # adaptive lasso
  correct.adaptivelasso[i] <- sum(abs(Adaptive.Lasso(beta0, x, y, opt.lam, d, thd1, 0.1))[4:6]<0.00
  incorrect.adaptivelasso[i] <- sum(abs(Adaptive.Lasso(beta0, x, y, opt.lam, d, thd1, 0.1))[1:3]<0.
  # aic bic
  for(j in 1:3){
    if(!(j %in% Best.Subset(x,y,criterion = 'AIC'))) incorrect.aic[i] = incorrect.aic[i] + 1
  }
  for(j in 4:6){
    if(!(j %in% Best.Subset(x,y,criterion = 'AIC'))) correct.aic[i] = correct.aic[i] + 1
  }
  for(j in 1:3){
    if(!(j %in% Best.Subset(x,y,criterion = 'BIC'))) incorrect.bic[i] = incorrect.bic[i] + 1
  }
  for(j in 4:6){
    if(!(j %in% Best.Subset(x,y,criterion = 'BIC'))) correct.bic[i] = correct.bic[i] + 1
  }
}
```

```
  # 把结果存放在 dataframe 里做成表格展示
  result.co <- colMeans(data.frame(correct.lasso.mm, correct.aic, correct.bic, correct.adaptivelasso)
  result.in <- colMeans(data.frame(incorrect.lasso.mm, incorrect.aic, incorrect.bic, incorrect.adapti
  result <- rbind.data.frame(result.co,result.in)
  colnames(result) <- c("lasso","aic","bic","adaptive")
  rownames(result) <- c("correct", "incorrect")
  return(result)
}
```