Block dependence detecting via the lasso

Nana Wang
Department of Statistics
University of California
Davis, CA 956161
nnawang@ucdavis.edu

Abstract

Meinshausen and Buhlmann (2006) proposed variable selection with the lasso for Gaussian graphical model. For a dependent network, I consider a similar method to detect the conditional dependence among blocks. Similar theoretical results are already done. In this project, I mainly work on the algorithms for solving the lasso problem and apply the lasso method to solve both the optimization problems in block dependence detecting and Meinshausen and Buhlmann's graph estimation. As expected, the simulation results are consistent with the theoretical results.

1 Introduction

It is well-known that the lasso[4] is a powerful shrinkage and selection method in linear models. In Gaussian graphical model, neighborhood selection with the lasso[3] is pretty attractive for sparse high-dimesnional graphs as well. Similarly, in a dependent block model, we can detect the dependence among blocks via the lasso.

Consider the block dependence network with N nodes: we have the adjacency matrix $Y = (Y_{i,j})_{i,j=1,\cdots,N}$ of a binary network with K known blocks, where $Y_{i,j} = 1$ denotes there is a connection between node i and node j and $Y_{i,j} = 0$ otherwise; $p_{k,l}$ is the probability of observing an edge between a node in block k and a node in block l and $z[i] \in \{1, \cdots, K\}$ denotes the label of the neighborhood of node i. Let $\eta_k = \log(\frac{p_{k,k}}{1-p_{k,k}})$. The block model says $Y_{i,j}|p_{z[i],z[j]} \sim \text{Bernoulli}(p_{z[i],z[j]})$ independently, while the dependence among blocks is induced by $\eta_k = x_k^t\beta + \epsilon_k$ with $\epsilon = (\epsilon_1, \cdots, \epsilon_k)^t \sim N(0, \Sigma)$. The p-dimensional vector covariates x_k, assumed to be known, describes some properties of neigborhood k and β is a p-dimensional parameter vector. Moreover, $p_{k,l}$ $(k \neq l)$ is independent with all the other $p_{s,t}$ when $(k, l) \neq (s, t)$. In order to detect the dependence among blocks, let

$$\hat{p}_{k,k} = \frac{\sum_{z[i]=z[j]=k} Y_{i,j}}{n_k} \quad \text{and} \quad \hat{\eta}_k = \log(\frac{\hat{p}_{k,k}}{1-\hat{p}_{k,k}}) \quad \text{for } k = 1, \cdots, K,$$

where n_k is the number of all the possible edges in block k. We assume n iid observed networks $Y^{(i)}$ with unobserved iid $p^{(i)}$ (or iid $\eta^{(i)}$). Let $\hat{\eta} = (\hat{\eta}_1, \cdots, \hat{\eta}_K)^t$. The statistics $\hat{\eta}^{(i)}$ are thus also iid. Let $\hat{\boldsymbol{\eta}}$ be the $n \times K(n)$-dimensional matrix containing n independent observations of $\hat{\eta}$ so that the columns $\hat{\boldsymbol{\eta}}_a$ are the vector of n independent observations of $\hat{\eta}_a$ for all $a \in \{1, \cdots, K\}$. Consider the optimization problem

$$\hat{\theta}^{a,\lambda} = \underset{\theta:\theta_a=0}{\arg\min} \left(n^{-1}\|\hat{\boldsymbol{\eta}}_a - \hat{\boldsymbol{\eta}}\theta\|_2^2 + \lambda\|\theta\|_1\right). \tag{1.1}$$

Based on $\{\hat{\theta}^{a,\lambda} : a \in \{1, \cdots, K\}\}$, we can get the estimation of the block dependence set E, denoted by $\hat{E}^\lambda$:

$$\hat{E}^\lambda = \{(a, b) : \hat{\theta}_b^{a,\lambda} \neq 0 \text{ and } \hat{\theta}_a^{b,\lambda} \neq 0\} \quad \text{or} \quad \hat{E}^\lambda = \{(a, b) : \hat{\theta}_b^{a,\lambda} \neq 0 \text{ or } \hat{\theta}_a^{b,\lambda} \neq 0\}. \tag{1.2}$$

Thus, it is vitally important to have a efficient algorithm to solve problem (1.1). This is classical optimization problem by the lasso method.

In Section 2, we first talk about two of the most popular algorithms for solving the lasso problem: coordinate descent and the modified LARS. Later in this section, we show some theoretical results of the Meinshausen-Buhlmann (MB) graph estimation and the dependent block network. Some simulation results are shown in section 3, while Section 4 contains a summary of the project.

2 Method

2.1 Algorithms for the lasso

The 'lars' and 'glmnet' are two well-known R packages for solving the lasso problem. In 'lars' package, a modified least angle regression (LARS) algorithm implements the lasso. The LARS with a simple modification calculates all possible estimates for a given lasso problem [1]. Coordinate descent algorithm for the lasso, which is faster and more efficient for fixed λ, is implemented in 'glmnet'. For my optimization problem (1.1), coordinate descent algorithm is already enough.

2.1.1 Coordinate descent algorithm for the lasso[2]

We consider a simple linear regression model: we have a response variable $Y \in \mathbb{R}$ and a predictor vector $X \in \mathbb{R}^p$, and the regression function is approximate by a linear model $E(Y|X = x) = x^T\beta$. Assume we have n observation pairs (x_i, y_i), the lasso problem is

$$\min_{\beta\in\mathbb{R}^p}\left\{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\|\beta\|_1\right\}. \tag{2.1}$$

Consider a coordinate desecent step for solving (2.1): suppose we have estimates $\tilde{\beta}_l$ for $l \neq j$ and we wish to partially optimize with respect to β_j. Let

$$R_\lambda(\tilde{\beta}_j) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i^T\tilde{\beta})^2 + \lambda\|\tilde{\beta}\|_1,$$

we have

$$\partial R_\lambda(\tilde{\beta}_j) = \left\{-\frac{2}{n}\sum_{i=1}^{n}x_{ij}(y_i - x_i^T\tilde{\beta}) + \text{sign}(\tilde{\beta}_j)\lambda\right\} \qquad \text{for } \tilde{\beta}_j \neq 0$$

and

$$\partial R_\lambda(0) = \left\{-\frac{2}{n}\sum_{i=1}^{n}x_{ij}(y_i - x_i^T\tilde{\beta}) + g\lambda : |g| \leq 1\right\}.$$

Since $R_\lambda(\tilde{\beta}_j)$ is convex,

$$\tilde{\beta}_j \text{ is optimal iff } 0 \in \partial R_\lambda(\tilde{\beta}_j). \tag{2.2}$$

By calculation, the coordinate-wise update has the form

$$\tilde{\beta}_j \leftarrow \frac{S\left(\frac{1}{n}\sum_{i=1}^{n}x_{ij}(y_i - \tilde{y}_i^{(j)}), \frac{\lambda}{2}\right)}{\frac{1}{n}\sum_{i=1}^{n}x_{ij}^2} \tag{2.3}$$

where $\tilde{y}_i^{(j)} = \sum_{l\neq j}x_{il}\tilde{\beta}_l$ and $S(z, \gamma) = \text{sign}(z)(|z| - \gamma)_+$ is the soft-thresholding operator.

Coordinate descent algorithm for the lasso

- Initilization. $\beta_j^{(c)} = \sum_{i=1}^{n}x_{ij}y_i$ for $j = 1, \cdots, p$.
- Updating steps. Cycle through $j = 1, \cdots, p$ until convergence.

$$\tilde{\beta}_j \leftarrow \frac{S\left(\frac{1}{n}\sum_{i=1}^{n}x_{ij}(y_i - \tilde{y}_i^{(j)}), \frac{\lambda}{2}\right)}{\frac{1}{n}\sum_{i=1}^{n}x_{ij}^2}$$

  where $\tilde{y}_i^{(j)} = \sum_{l\neq j}x_{il}\tilde{\beta}_l$.

2.1.2 Least angle regression for the lasso[4]

We know that LARS can be modified to generate the full set of lasso solutions. Consider the setup: let $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_p$ be n-vectors representing the covariates and $\boldsymbol{y}$ be the n-vector of responses with assumptions

$$\sum_{i=1}^{n} y_i = 0, \quad \sum_{i=1}^{p} x_{ij} = 0, \quad \sum_{i=1}^{n} x_{ij}^2 = 1 \qquad \text{for } j = 1, 2, \cdots, p.$$

The LARS algorithm

- Initialization. $\hat{\boldsymbol{\mu}}_0 = \mathbf{0}$.
- Updating steps. $\hat{\boldsymbol{\mu}}_{\mathcal{A}}$ is the current LARS estimate corresponds to a lasso solution $\hat{\boldsymbol{\beta}}$.
  (1) Compute $\hat{\boldsymbol{c}} = \boldsymbol{x}'(\boldsymbol{y} - \hat{\boldsymbol{\mu}}_{\mathcal{A}})$, where $\boldsymbol{x} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_p)$.
  (2) Let $\hat{C} = \max_j\{|\hat{c}_j|\}$ and $\mathcal{A} = \{j : |\hat{c}_j| = \hat{C}\}$.
  (3) Define $X_{\mathcal{A}} = (\cdots s_j \boldsymbol{x}_j \cdots)_{j \in \mathcal{A}}$ where $s_j = \text{sign}\{\hat{c}_j\}$ for $j \in \mathcal{A}$.
  (4) Compute

$$\begin{aligned} &\mathcal{G}_{\mathcal{A}} = X'_{\mathcal{A}} X_{\mathcal{A}} \quad \text{and} \quad A_{\mathcal{A}} = (\mathbf{1}'_{\mathcal{A}} \mathcal{G}_{\mathcal{A}}^{-1} \mathbf{1}_{\mathcal{A}})^{-1/2} \\ &\boldsymbol{u}_{\mathcal{A}} = X_{\mathcal{A}} w_{\mathcal{A}} \quad \text{where } w_{\mathcal{A}} = A_{\mathcal{A}} \mathcal{G}_{\mathcal{A}}^{-1} \mathbf{1}_{\mathcal{A}} \\ &\hat{\gamma} = \min_{j \in \mathcal{A}^c}{}^{+} \left\{ \frac{\hat{C} - \hat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\hat{C} + \hat{c}_j}{A_{\mathcal{A}} + a_j} \right\} \quad \text{where} \quad \boldsymbol{a} = X' \boldsymbol{u}_{\mathcal{A}} \end{aligned}$$

  (5) Update $\hat{\boldsymbol{\mu}}_{i+1} = \hat{\boldsymbol{\mu}}_i + \hat{\gamma} \boldsymbol{u}_{\mathcal{A}}$

Lasso modification: let $\hat{\boldsymbol{d}}$ be the p-vector equaling $s_j w_{\mathcal{A}j}$ for $j \in \mathcal{A}$ and zero elsewhere. Compute

$$\gamma_j = -\hat{\beta}_j / \hat{d}_j \quad \text{and} \quad \tilde{\gamma} = \min_{\gamma_j > 0}\{\gamma_j\},$$

$\tilde{\gamma}$ equals infinity by definition if there is no $\gamma_j > 0$. Let $\tilde{j}$ be the index such that $\gamma_j = \tilde{\gamma}$. If $\tilde{\gamma} < \hat{\gamma}$, stop the ongoing LARS step but update

$$\hat{\boldsymbol{\mu}}_{\mathcal{A}_+} = \hat{\boldsymbol{\mu}}_{\mathcal{A}} + \tilde{\gamma} \boldsymbol{u}_{\mathcal{A}} \quad \text{and} \quad \mathcal{A} = \mathcal{A} - \{\tilde{j}\}.$$

It has been shown in [4], the modified LARS algorithm yields all lasso solutions.

2.2 Meinshausen-Buhlmann graph estimation

Meinshausen and Buhlmann (2006) consider a graphical model $G = (V, E)$ of the K-dimensional multivariate normal distributed random variable $\eta = (\eta_1, \cdots, \eta_K) \sim N(0, \Sigma)$ with positive-definite Σ, where $V = \{1, \ldots, K\}$ and $E = \{(a, b) \in V \times V : \eta_a \not\perp\!\!\!\perp \eta_b | \eta_{-(a,b)}\}$. Assume n iid observations of η, they proposed a edge set estimation method with the lasso: first consider the lasso problem

$$\tilde{\theta}^{a,\lambda} = \underset{\theta : \theta_a = 0}{\arg\min} \left(n^{-1} \|\boldsymbol{\eta}_a - \boldsymbol{\eta}\theta\|_2^2 + \lambda \|\theta\|_1 \right), \tag{2.4}$$

then the edge set estimation is

$$\hat{E}^{\lambda} = \{(a, b) : \tilde{\theta}_b^{a,\lambda} \neq 0 \text{ and } \tilde{\theta}_a^{b,\lambda} \neq 0\} \quad \text{or} \quad \hat{E}^{\lambda} = \{(a, b) : \tilde{\theta}_b^{a,\lambda} \neq 0 \text{ or } \tilde{\theta}_a^{b,\lambda} \neq 0\}. \tag{2.5}$$

Moreover, they show us consistency of the estimation method:

Theorem. Let assumptions 1-6 in [3] hold. Let $\lambda_n \sim d n^{-(1-\epsilon)/2}$ with $\kappa < \epsilon < \xi$ and $d > 0$. There exists some $c > 0$ so that,

$$P(\hat{E}^{\lambda} = E) = 1 - O(\exp(-c n^{\epsilon})) \quad \textit{for } n \to \infty. \tag{2.6}$$

2.3 Block dependence detecting

As discussed in the introduction section, for the dependent block network, since the underlying $\boldsymbol{\eta}$ is unknown, we apply Meinshausen and Buhlmann's graph estimation method to $\hat{\boldsymbol{\eta}}$, the estimation of $\boldsymbol{\eta}$, instead of $\boldsymbol{\eta}$. Let $m(n)$ be the minimum number of the possible edges in each blocks. We have similar results:

Theorem. Let assumptions 1-6 in [3] hold and $m(n) \sim n^{\nu}$ *with* $\nu > \max\{4-\xi-3\epsilon, 2+2\kappa-2\epsilon\}$ *for* $\kappa < \epsilon < \xi$. *Let* $\lambda_n \sim dn^{-(1-\epsilon)/2}$ *with* $d > 0$. *We have* $\forall \rho > 0$,

$$P(\hat{E}^{\lambda} = E) = 1 - o(n^{-\rho}) \quad \textit{for } n \to \infty.$$

3 Results

3.1 Solution path for the lasso via modified lars

I randomly generate a $\boldsymbol{x}_{4\times 5}$ and $\boldsymbol{y}_{5\times 1}$. The value of x and y are in appendix. By applying both my function of the modified LARS algorithm and the function in the 'lars' package, the solution paths for this lasso problem are shown in figure 1. As expected, the results are the same.

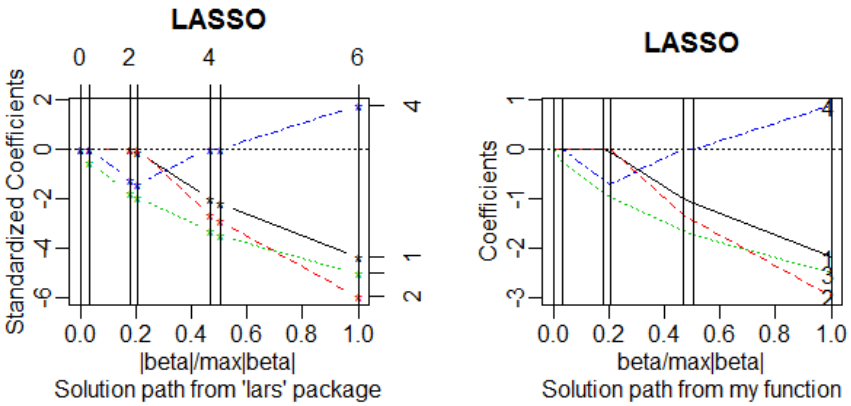


Figure 1

3.2 Meinshausen-Buhlmann graph estimation

In the simulation study, since I only need to solve the lasso problem with fixed λ but not the whole solution path, it is more efficient to apply coordinate descent algorithm. I first consider the error probability $P(E \neq \hat{E})$ as $n \to \infty$ of the case: $K = O(n^{1/2})$, $\Sigma = I_K$ and $\lambda = n^{-1/3}$ with my own function of coordinate descent for lasso. The result is shown in figure 2: as $n \to \infty$, $P(E \neq \hat{E}) = O(\exp(-cn^{\epsilon}))$ approximately. As expected, the simulation result is consistent with the theorem.

For the case: $\Sigma = I$, $K = O(n^{0.1})$ and $\lambda = 0.1 * n^{-0.05}$, I compute the error ratio versus the sample size n by applying both my function of coordinate descent algorithm for lasso and the function in package 'glmnet'. The results are shown in figure 3(a). As expected, the error ratio goes to 0 as $n \to \infty$. Moreover, the results from my function and the function in package 'glmnet' are almost the same.

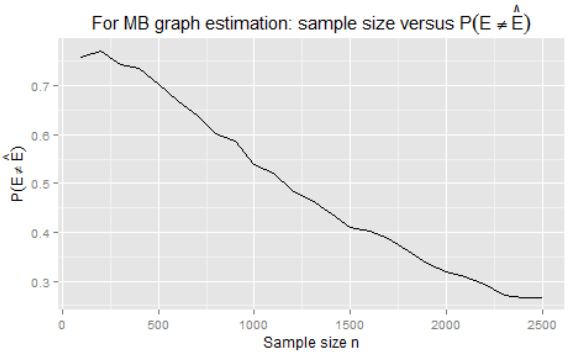
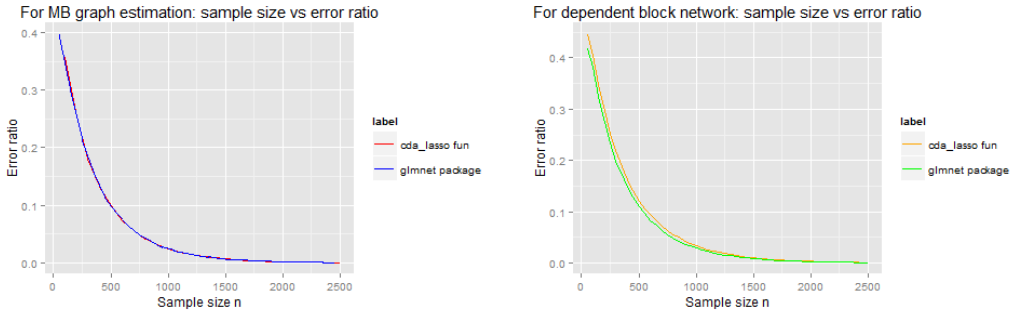


Figure 2



(a) For MB graph estimation

(b) For block dependence detecting

Figure 3: Sample size versus error ratio

3.3 Block dependence detecting

Similar as in section 3.1, I am curious about the the error probability $P(E \neq \hat{E})$ versus the sample size n. Here I consider the case: $K = O(n^{0.1})$, $\Sigma = I_K$, $m(n) \sim n^{0.4}$ and $\lambda = 0.25 * n^{-0.05}$. The simulation result is in figure 4. As expected, the error probablity, $P(E \neq \hat{E})$, goes to 0 as $n \to \infty$. This is consistent with the theorem.

Figure 4

Then I compute the error ratio of detecting dependence for the case: $\Sigma = I$, $K = O(n^{0.1})$, $m(n) \sim n^{0.4}$ and $\lambda = 0.1 * n^{-0.05}$ with both my function and the function in package 'glmnet'. The results are shown in figure 3(b). The error ratio of detecting dependence goes to 0 as the sample size $n \to \infty$.

Moreover, figure 5 shows the error ratio versus sample size of both MB graph estimation and the block dependence detecting for the case: $\Sigma = I$, $K = O(n^{0.1})$ and $m(n) \sim n^{0.4}$ for the depdent block network with $\lambda = 0.1 * n^{-0.05}$. From the figure, for both the MB graph estimation and the block dependence detecting, the error ratio from my function are almost the same as that from the function in package 'glmnet' . The error ratio of the block dependence detecting is slightly greater than that of the MB graph estimation. This is reasonable since we observe the true $\boldsymbol{\eta}$ for the MB graph estimation, however, for the block dependence detecting, only $\hat{\boldsymbol{\eta}}$, the estimation of $\boldsymbol{\eta}$, is known.

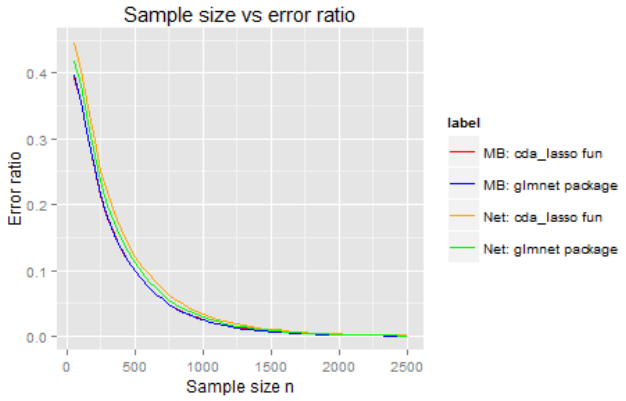


Figure 5

4 Summary

In this project, I implement both the coordinate descent algorithm and the modified LARS algorithm for solving the lasso problem. The coordinate descent algorithm solves problem (1.1) with fixed λ efficiently. The modified LARS algorithm gives all possible solutions for a given lasso problem in a few steps. For both the two algorithms, the results of my functions are almost the same as that of the functions in the 'glmnet' package or 'lars' package.

Next, in the simulation study, since I want to solve the lasso problem with fixed λ, the coordinate descent algorithm is a better chocie. For both Meinshausen-Buhlmann graph estimation and block dependence detecting, I consider the error probability and the error ratio versus the sample size $n \to \infty$. The simulation results are consistent with the theoretical results.

References

[1] Efron, Bradley, et al. "Least angle regression." The Annals of statistics 32.2 (2004): 407-499.

[2] Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. "Regularization paths for generalized linear models via coordinate descent." Journal of statistical software 33.1 (2010): 1.

[3] Meinshausen, Nicolai, and Peter Bhlmann. "High-dimensional graphs and variable selection with the lasso." The Annals of Statistics (2006): 1436-1462.

[4] Tibshirani, Robert. "Regression shrinkage and selection via the lasso." Journal of the Royal Statistical Society. Series B (Methodological) (1996): 267-288.

Appendix

Data in Section 3.1

$$\boldsymbol{x} = \begin{bmatrix} -0.8969 & 0.132 & 0.418 & -2.3111 \\ 0.1848 & 0.708 & 0.982 & 0.8786 \\ 1.5878 & -0.240 & -0.393 & 0.0358 \\ -1.1304 & 1.984 & -1.040 & 1.0128 \\ -0.0803 & -0.139 & 1.782 & 0.4323 \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} 2.0908 \\ -1.1999 \\ 1.5896 \\ 1.9547 \\ 0.0049 \end{bmatrix}$$

Code

1. **Julia code: coordinate descent algorithm for the lasso**

```
#soft-thresholding operator
function S(x,r)
    if abs(x)>r
        s=sign(x)*(abs(x)-r)
    else
        s=0
    end
    return s
end

#The coordinate descent function
function lasso_cda(x,y,lambda; optTol=1e-8,maxIts=1e6)
    n,p=size(x)
    ###initialization
    beta=[sum([x[i,j]*y[i] for i in 1:n]) for j in 1:p]
    its=0; id=0
    for k=1:maxIts
        rd=k%p
        j=(rd==0)? p:rd
        yc=[y[i]-sum([x[i,l]*beta[l] for l=1:p])+x[i,j]*beta[j]
           for i in 1:n]
        bj=copy(beta[j])
        beta[j]=S(sum([x[i,j]*yc[i] for i in 1:n])/n,lambda)
        id=(abs(bj-beta[j])<optTol)?(id+1):0
        if id>=p
            break;
        end
        its=its+1
    end
    if (its==maxIts)
        println("Not converge!")
    end
    return beta, its
end
```

2. **R code: coordinate descent algorithm for the lasso**

```
#soft-thresholding operator
S=function(x,r){
  s=sign(x)*(abs(x)-r)*(abs(x)-r>0)
  return(s)
}

#The coordinate descent function
cda_lasso=function(x,y,lambda,tol=1e-16,maxIts=1e6){
  beta0=t(y)%*%x
  n=nrow(x); p=ncol(x)
  its=0; id=0
  meanx2=apply(x^2,2,mean)
  x2=t(x)%*%x
  beta=beta0
```

```
  for (i in 1:maxIts){
    j=ifelse(i%%p==0,p,i%%p)
    bj=beta[j]
    beta[j]=S((beta0[j]-x2[j,-j]%*%beta[-j])/n,lambda)/meanx2[j]
    id=ifelse(abs(beta[j]-bj)<tol,id+1,0)
    if (id>p){
      break
    }
    its=its+1
  }
  if (its>=maxIts){
    print("Not converge!")
  }
  return(beta)
}
```

3. **R code: least angle regression for lasso**

```
library(MASS)

lars_lasso=function(x, y, maxIts=100,tol=1e-8, lasso=TRUE,
                    standardize=TRUE, pt=TRUE){
  if (standardize){
    y=y-mean(y); x=t((t(x)-apply(x,2,mean))/apply(x,2,sd))
  }
  m=ncol(x); n=nrow(x)
  nvars=min(m,n)

  #Initialization
  beta=rep(0,m); mu=rep(0,n);
  beta.all=beta; t.all=sum(abs(beta))
  inactive=1:m; active=numeric(0)
  lassocond=0;
  its=0; vars=0

  while(vars<nvars && its<maxIts){

    #updat
    c=t(x)%*%(y-mu); c.max=max(abs(c))
    #c.max=max(abs(c[inactive]));
    j=inactive[which(abs(abs(c[inactive])-c.max)<tol)]

    if (!lassocond){
      active=c(active, j)
      inactive=inactive[!(inactive %in% j)]
      vars=vars+length(j)
      its=its+1
      if (pt){
        cat("LARS step",its, "variable", j, "added \n" )
      }
    }

    s=rep(0,m); s[active]=sign(c[active])
    xa=(x%*%diag(s))[,active]
    gi=ginv(t(xa)%*%(xa)); l=length(active)
    A=as.vector(1/sqrt(matrix(1,1,1)%*%gi%*%matrix(1,1,1)))
    w=A*gi%*%matrix(1,1,1)
    u=xa%*%w
    d=rep(0,m); d[active]=as.vector(s[active]*w)

    if (vars==nvars){
      r=c.max/A;
    }else{
      a=t(x)%*%u
      temp=c((c.max-c[inactive])/(A-a[inactive]),
```

```
        (c.max+c[inactive])/(A+a[inactive]))
      r=min(c(temp[temp>0], c.max/A))
    }

    if (lasso){
      lassocond=0
      temp=-beta[active]/d[active]
      r.tilde=ifelse(any(temp>0), min(temp[temp>0]), Inf)
      j=which(temp==r.tilde)
      if (r.tilde<r){
        r=r.tilde
        lassocond=1
      }
    }

    mu=mu+r*u
    beta=beta+r*d

    beta.all=cbind(beta.all, beta)

    if (lassocond==1){
      j=(active[j])[1]
      inactive=c(inactive, j)
      active=active[!(active %in% j)]
      vars=vars-1
      its=its+1
      if (pt){
        cat("Lasso Step ", its, "variable", j, " dropped\n")
      }
    }
  }

  return(list(beta.all, its))
}


###results
set.seed(2)
x=matrix(rnorm(20),5,4); y=rnorm(5)

#from 'lars' package
library(lars)
fit=lars(x,y,trace=TRUE)

#from lars_lasso function
beta.all=lars_lasso(x,y)[[1]]
beta=apply(beta.all, 2, function(x) sum(abs(x)))
beta.s=beta/max(abs(beta))
```

4. **R code for simulation**

```
##########The coordinate descent algorithm for solving lasso
#soft-thresholding operator
S=function(x,r){
  s=sign(x)*(abs(x)-r)*(abs(x)-r>0)
  return(s)
}

#The coordinate descent function
cda_lasso=function(x,y,lambda,tol=1e-16,maxIts=1e6){
  beta0=t(y)%*%x
  n=nrow(x); p=ncol(x)
  its=0; id=0
  meanx2=apply(x^2,2,mean)
  x2=t(x)%*%x
```

```
  beta=beta0
  for (i in 1:maxIts){
    j=ifelse(i%%p==0,p,i%%p)
    bj=beta[j]
    beta[j]=S((beta0[j]-x2[j,-j]%*%beta[-j])/n,lambda)/meanx2[j]
    id=ifelse(abs(beta[j]-bj)<tol,id+1,0)
    if (id>p){
      break
    }
    its=its+1
  }
  if (its>=maxIts){
    print("Not converge!")
  }
  return(beta)
}

#beta estimation with glmnet
beta.hat=function(x,y,lambda){
  fit=glmnet(x,y,lambda=lambda)
  beta=predict(fit,type="coef")[-1]
  return(beta)
}

##########error-ratio simulation: for simple mb model with glmnet
library(glmnet)
library(MASS)
library(foreach)
library(doSNOW)

r.ssmb0=function(n){
  p=50*floor(n^(0.1)); lambda=0.1*n^(-0.05)
  x=mvrnorm(n,mu=rep(0,p),Sigma=diag(p))
  E.mean=sapply(1:p, function(i)
  mean(beta.hat(x[,-i],x[,i],lambda)!=0))
  return(mean(E.mean))
}

ratio.ssmb0=function(n,m) mean(sapply(1:m, function(o) r.ssmb0(n)))

cl=makeCluster(8)
registerDoSNOW(cl)
ratio.smb0=foreach(i=1:50,.combine=c,.packages=c("MASS","glmnet"))
          %dopar%
{
  ratio.ssmb0(50*i,100)
}

stopCluster(cl)

#results
ratio.smb0
write.csv(ratio.smb0,"errorratio-smb0.csv")
#########error-ratio simulation: for simple mb model with cda_lasso
library(MASS)
library(foreach)
library(doSNOW)

r.ssmb=function(n){
  p=50*floor(n^(0.1)); lambda=0.1*n^(-0.05)
  x=mvrnorm(n,mu=rep(0,p),Sigma=diag(p))
  E.mean=sapply(1:p, function(i)
  mean(cda_lasso(x[,-i],x[,i],lambda)!=0))
  return(mean(E.mean))
}
```

```
ratio.ssmb=function(n,m) mean(sapply(1:m, function(o) r.ssmb(n)))

cl=makeCluster(8)
registerDoSNOW(cl)
ratio.smb=foreach(i=1:50,.combine=c,.packages="MASS")%dopar%
{
  ratio.ssmb(50*i,100)
}

stopCluster(cl)

#results
ratio.smb
write.csv(ratio.smb,"errorratio-smb.csv")

########error-ratio simulation: for simple network model with glmnet
library("glmnet")
library(MASS)
library(foreach)
library(doSNOW)

eta.hat=function(eta,m){
  pb=1/(1+exp(-eta)); k=length(pb)
  pb.hat=sapply(1:k, function(i) mean(sample(c(0,1),m, replace =
   TRUE, prob=c(1-pb[i],pb[i]))))
  pb.hat=pb.hat+(pb.hat==1)*(0.99-pb.hat)
  pb.hat=pb.hat+(pb.hat==0)*(0.01-pb.hat)
  et.hat=log(pb.hat/(1-pb.hat))
  return(et.hat)
}

r.ssmb.glm=function(n){
  p=50*floor(n^(0.1)); lambda=0.1*n^(-0.05); m=10*ceiling(n^(0.4))
  x=mvrnorm(n,mu=rep(0,p),Sigma=diag(p))
  x.hat=t(sapply(1:n, function(i) eta.hat(x[i,],m)))
  E.mean=sapply(1:p, function(i)
  mean(beta.hat(x.hat[,-i],x.hat[,i],lambda)!=0))
  return(mean(E.mean))
}

ratio.ssmb.glm=function(n,m) mean(sapply(1:m, function(o)
r.ssmb.glm(n)))

cl=makeCluster(8)
registerDoSNOW(cl)
ratio.glm=foreach(i=1:50,.combine=c,.packages=c("MASS",
"glmnet"))%dopar%
{
  ratio.ssmb.glm(50*i,100)
}

stopCluster(cl)

####results
ratio.glm
write.csv(ratio.glm,"errorratio-glm.csv")

######error-ratio simulation: for simple network model with cda_lasso
library(MASS)
library(foreach)
library(doSNOW)

eta.hat=function(eta,m){
  pb=1/(1+exp(-eta)); k=length(pb)
```

```
  pb.hat=sapply(1:k, function(i) mean(sample(c(0,1),m, replace =
   TRUE, prob=c(1-pb[i],pb[i]))))
  pb.hat=pb.hat+(pb.hat==1)*(0.99-pb.hat)
  pb.hat=pb.hat+(pb.hat==0)*(0.01-pb.hat)
  et.hat=log(pb.hat/(1-pb.hat))
  return(et.hat)
}

r.ssmb.cda=function(n){
  p=50*floor(n^(0.1)); lambda=0.1*n^(-0.05); m=10*ceiling(n^(0.4))
  x=mvrnorm(n,mu=rep(0,p),Sigma=diag(p))
  x.hat=t(sapply(1:n, function(i) eta.hat(x[i,],m)))
  E.mean=sapply(1:p, function(i) mean(cda_lasso(x.hat[,-i],x.hat[,i],
  lambda)!=0))
  return(mean(E.mean))
}

ratio.ssmb.cda=function(n,m) mean(sapply(1:m, function(o) r.ssmb.cda(n)))

cl=makeCluster(8)
registerDoSNOW(cl)
ratio.cda=foreach(i=1:50,.combine=c,.packages="MASS")%dopar%
{
  ratio.ssmb.cda(50*i,100)
}

stopCluster(cl)

#results
ratio.cda
write.csv(ratio.cda,"errorratio-cda.csv")

##########error probability: simple mb case
library(MASS)
library(foreach)
library(doSNOW)
ssmb=function(n){
  p=floor(n^(1/2)); lambda=n^(-1/3)
  x=mvrnorm(n,mu=rep(0,p),Sigma=diag(p))
  E.mean=sapply(1:p, function(i)
   mean(cda_lasso(x[,-i],x[,i],lambda)!=0))
  return(mean(E.mean))
}


p.ssmb=function(n,m) mean(sapply(1:m, function(o) ssmb(n)!=0))

cl=makeCluster(10)
registerDoSNOW(cl)
p=foreach(i=1:25,.combine=c,.packages="MASS") %dopar%
{
  p.ssmb(100*i,10000)
}

stopCluster(cl)

#results
p
write.csv(p,"mb-p.csv")

##########error probability: block dependent case
#beta estimation with glmnet
beta.hat=function(x,y,lambda){
  fit=glmnet(x,y,lambda=lambda)
  beta=predict(fit,type="coef")[-1]
```

```
  return(beta)
}

#simulate-simple-network case
library("glmnet")
library(MASS)
library(foreach)
library(doSNOW)

eta.hat=function(eta,m){
  pb=1/(1+exp(-eta)); k=length(pb)
  pb.hat=sapply(1:k, function(i) mean(sample(c(0,1),m,
  replace = TRUE,prob=c(1-pb[i],pb[i]))))
  pb.hat=pb.hat+(pb.hat==1)*(0.99-pb.hat)
  pb.hat=pb.hat+(pb.hat==0)*(0.01-pb.hat)
  et.hat=log(pb.hat/(1-pb.hat))
  return(et.hat)
}

r.ssmb.glm=function(n){
  p=50*floor(n^(0.1)); lambda=0.25*n^(-0.05);
  m=10*ceiling(n^(0.4))
  x=mvrnorm(n,mu=rep(0,p),Sigma=diag(p))
  x.hat=t(sapply(1:n, function(i) eta.hat(x[i,],m)))
  E.mean=sapply(1:p, function(i)
  mean(beta.hat(x.hat[,-i],x.hat[,i],lambda)!=0))
  return(mean(E.mean))
}

p.ssmb.glm=function(n,m) mean(sapply(1:m, function(o) r.ssmb.glm(n)!=0))

cl=makeCluster(10)
registerDoSNOW(cl)
net.p=foreach(i=25:75,.combine=c,.packages=c("MASS", "glmnet"))
%dopar%
{
  p.ssmb.glm(10*i,10000)
}

stopCluster(cl)

#results
net.p
write.csv(net.p,"net-p.csv")
```

5. **R code: figures**

```
#figure 1
par(mfrow=c(1,2))
plot(fit, mgp=c(1.5, 0.5,0), sub="Solution path from 'lars' package" )

matplot(beta.s, t(beta.all), type="l",
        xlab="beta/max|beta|", ylab="Coefficients",
        main="LASSO", mgp=c(1.5,0.5,0),
        sub="Solution path from my function")
abline(h=0, lty=3)
for (i in 1:length(beta)){
  abline(v=beta.s[i])
}

for (i in 1: nrow(beta.all)){
  text(x=0.99, y=beta.all[i, ncol(beta.all)], label=i)
}

setwd("E:/study/study-ucd/2015-2016/2015 Fall/Mat258a/
```

```
Final Project/code/summary")
library("ggplot2")

#fiugre 2
p.mb=read.csv("mb-p.csv")
p.mb=data.frame(x=100*(1:25), y=p.mb[,2])

ggplot(aes(x,y),data=p.mb)+geom_line()+
   ggtitle(expression(paste("For MB graph estimation:
  sample size versus ", P(E!=hat(E)))))+labs(x="Sample size n",
   y=expression(P(E!=hat(E))))+
   theme(plot.title = element_text(size = 15))

#figure 3
#(a)
r.smb0=read.csv("errorratio-smb0.csv")
r.smb0[,1]=50*(1:50)
r.smb0=data.frame(x=r.smb0[,1], y=r.smb0[,2])

r.smb1=read.csv("errorratio-smb.csv")
r.smb1[,1]=50*(1:50)
r.smb1=data.frame(x=r.smb1[,1], y=r.smb1[,2])

r.smb=rbind(r.smb0, r.smb1)
r.smb$label=rep(c("glmnet package","cda_lasso fun"),50)

ggplot(aes(x,y,color=label,group=label),data=r.smb)+geom_line()
+scale_colour_manual(values = c("red","blue"))+
   ggtitle("For MB graph estimation: sample size vs error
   ratio")+labs(x="Sample size n", y="Error ratio")+
   theme(plot.title = element_text(size = 15))

#(b)
r.glm=read.csv("errorratio-glm.csv")
r.glm[,1]=50*(1:50)
r.glm=data.frame(x=r.glm[,1], y=r.glm[,2])

r.cda=read.csv("errorratio-cda.csv")
r.cda[,1]=50*(1:50)
r.cda=data.frame(x=r.cda[,1], y=r.cda[,2])

r.net=rbind(r.glm, r.cda)
r.net$label=rep(c("glmnet package","cda_lasso fun"),each=50)

ggplot(aes(x,y,color=label,group=label),data=r.net)+geom_line()
+scale_colour_manual(values = c("orange","green"))+
   ggtitle("For dependent block network: sample size vs error
   ratio")+labs(x="Sample size n", y="Error ratio")+
   theme(plot.title = element_text(size = 15))

#figure 4
p.net=read.csv("net-p.csv")
p.net=data.frame(x=10*(25:75), y=p.net[,2])
library("ggplot2")
ggplot(aes(x,y),data=p.net)+geom_line()+
   ggtitle(expression(paste("For block dependence detecting:
  sample size versus ", P(E!=hat(E)))))+labs(x="Sample size n",
    y=expression(P(E!=hat(E))))+
   theme(plot.title = element_text(size = 15))

#figure 5
r.smb0=read.csv("errorratio-smb0.csv")
r.smb0[,1]=50*(1:50)
r.smb0=data.frame(x=r.smb0[,1], y=r.smb0[,2])
```

```
r.smb1=read.csv("errorratio-smb.csv")
r.smb1[,1]=50*(1:50)
r.smb1=data.frame(x=r.smb1[,1], y=r.smb1[,2])

r.glm=read.csv("errorratio-glm.csv")
r.glm[,1]=50*(1:50)
r.glm=data.frame(x=r.glm[,1], y=r.glm[,2])

r.cda=read.csv("errorratio-cda.csv")
r.cda[,1]=50*(1:50)
r.cda=data.frame(x=r.cda[,1], y=r.cda[,2])

r=rbind(r.smb0, r.smb1, r.glm, r.cda)
r$label=rep(c("MB: glmnet package","MB: cda_lasso fun",
 "Net: glmnet package","Net: cda_lasso fun"),each=50)

ggplot(aes(x,y,color=label,group=label),data=r)+geom_line()+
scale_colour_manual(values = c("red","blue","orange","green"))+
   ggtitle("Sample size vs error ratio")+labs(x="Sample size n",
    y="Error ratio")+
   theme(plot.title = element_text(size = 15))
```