The `fastclime` Package for Fast Constrained l_1 Minimization Approach to Sparse Precision Matrix Estimation in `R`

Haotian Pang * Han Liu † Robert Vanderbei ‡

May 20, 2013

Abstract

We describe an `R` package called `fastclime` (ver 1.0), which provides functions to estimate sparse precision matrix by CLIME using parametric simplex method. Compared with existing package `clime`, it has many extra advantages: (1) It gives a full path of the tuning parameter $lambda_n$ for each column while solving the linear programming problem (2) It only requires a few iteration to recover each sparse precision matrix column. (3) The entire solver is written in C and it is easy to modify. (4) It recovers both precision matrix and adjacency matrix for each iteration.

1 Background and the CLIME estimator

Estimating the covariance matrix and its inverse has been widely discussed in many statistical analysis. Let $X = (X_1, ..., X_p)^T$ be a p-variate random vector with Gaussian distribution $N(\mu, \Sigma)$. The covariance matrix Σ can be easily estimated by the sample covariance:

$$\Sigma_n = \frac{1}{n}\sum_{k=1}^{n}(x_k - \overline{x})(x_k - \overline{x})^T \tag{1}$$

where $\overline{x} = \frac{1}{n}\sum_{k=1}^{n} x_k$ is the sample mean.

The precision matrix, or sometimes called the concentration matrix, Ω, is defined as $\Omega = \Sigma^{-1}$. Notice for Gaussian distribution, $\Omega_{jk} = 0$ indicates that X_i X_j are independent given other variables $X_k (k \neq i, j)$. A closed related problem is the selection of graphical model. Let a graph $G = (V, E)$ denotes the conditional independence relation between the variables X. The vertex set V is the set of variables $X_1, X_2, ..., X_p$. An edge between node i and node j is contained in edge set E if and only if the two variables X_i and X_j is not independent given $X_k (k \neq i, j)$. Therefore, for gaussian distribution, recovering the structure of the conditional independence graph is the same as estimating the support of a precision matrix.

Unlike precision matrix, there is no direct method to estimate the precision matrix. One famous method is called l_1 regularized log-determinant program [Banerjee et al., 2008]:

$$\Omega = \text{ argmin } trace(\Sigma_n X) - \log(\det(X)) + \lambda\|X\|_1 \text{ subject to: } X \succ 0 \tag{2}$$

This estimator can be solved efficiently by methods such as GLASSO (Graphical Lasso)[Friedman et al., 2007] [Friedman et al., 2010] and ADMM (Alternating Direction Method of Multipliers) [Boyd et al., 2010].

*email: hpang@princeton.edu, Department of Electrical Engineering, Princeton University

†email: hanliu@princeton.edu, Department of Operational Research and Financial Engineering, Princeton University

‡email: rvdb@princeton.edu, Department of Operational Research and Financial Engineering, Princeton University

Another simple estimator proposed recently is called CLIME (Constrained l_1 Minimization Estimator). Cai et al. [2010] defines the CLIME estimator $\widehat{\Omega}$ as the solution to the following problem:

$$\min \|\Omega\|_1 \text{ subject to: } |\Sigma_n \Omega - I|_\infty \leq \lambda_n,\ \Omega \in \mathbb{R}^{p \times p} \tag{3}$$

λ_n is defined as a tuning parameter. This minimization problem can be further decomposed into p small problems, simply recovering the precision matrix column by column:

$$\min \|\beta\|_1 \text{ subject to: } |\Sigma_n \beta - e_i|_\infty \leq \lambda_n \tag{4}$$

Of course, the estimator we obtained by solving these series of linear programming problems are not symmetric, but we can simply take the average value of $\widehat{\Omega}_{ij}$ and $\widehat{\Omega}_{ji}$. The result will be positive definite with high probability. The error between the estimator $\hat{\Omega}$ and Ω satisfies $\|\hat{\Omega} - \Omega\|_2 = O_p(s\sqrt{log(p)/n})$ and $|\hat{\Omega} - \Omega|_\infty = O_p(\sqrt{log(p)/n})$, where s denotes the sparsity of the precision matrix. [Cai et al., 2010]

2 Parametric Simplex Method

We briefly describe the primal simplex method and introduces the parametric simplex methods in this section.

Standard Linear Programming form is given by

$$\max c^T x \quad \text{subject to: } Ax \leq b, \quad x \geq 0 \tag{5}$$

For primal simplex method, we requires that $b \geq 0$. We call this property as primal feasibility. We change the standard form into equality form with slack variables:

$$\max c^T x \quad \text{subject to: } Ax + w = b, \quad x, w \geq 0$$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & 1 & & \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & & 1 & \\ \vdots & \vdots & & \vdots & & & \ddots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & & & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 & b_2 & \cdots & b_m \end{bmatrix}^T$$

$$c = \begin{bmatrix} c_1 & c_2 & \cdots & c_n & 0 & \cdots & 0 \end{bmatrix}^T$$

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n & w_1 & \cdots & w_m \end{bmatrix}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_n & x_{1+1} & \cdots & x_{n+m} \end{bmatrix}^T$$

The new variables $x_{n+1}, ..., x_{n+m}$ are called slack variables. Initially, we called the slack variables basic variables and the original variables nonbasic variables. We separate the matrix A, the vector b and the vector c into the basic and nonbasic parts as well. The first part of A corresponds to basic variables and the identity matrix part corresponds to basic variables.

$$A = \begin{bmatrix} N & B \end{bmatrix} \quad x = \begin{bmatrix} x_N \\ x_B \end{bmatrix} \quad c = \begin{bmatrix} c_N \\ c_B \end{bmatrix}$$

$Ax = b$ can be written as $Nx_N + Bx_B = b$. We denote the objective $c^T x$ as ζ, then we have:

$$\begin{aligned} x_B =& B^{-1}b - B^{-1}Nx_N \\ =& x_B^* - B^{-1}Nx_N \end{aligned}$$

$$\begin{aligned} \zeta =& c^T x \\ =& c_B^T x_B + c_N^T x_N \\ =& c_B^T (B^{-1}b - B^{-1}Nx_N) + c_N^T x_N \\ =& c_B^T B^{-1} b - ((B^{-1}N)^T c_B - c_N)^T x_N \\ =& \zeta^* - (z_N^*)^T x_N \end{aligned} \tag{6}$$

$$\tag{7}$$

with $\zeta^* = c_B^T B^{-1} b$, $x_B^* = B^{-1} b$ and $z_N^* = (B^{-1}N)^T c_B - c_N$.

We call this two equations as "primal dictionary" associated with current basis B. Primal feasibility requires that $x_B \geq 0$, which is initially guaranteed by $b \geq 0$. We read off the values of x_B and ζ by setting x_N to be zero.

The dual form of the Linear Programming Problem has the following form:

$$\min b^T y \quad \text{subject to: } A^T y \geq c, \quad y \geq 0 \tag{8}$$

We change it into equality form with slack variables as well:

$$\min b^T y \quad \text{subject to: } A^T y - z = c, \quad y, z \geq 0 \tag{9}$$

The dual solution is given by:

$$z = \begin{bmatrix} z_1 & z_2 & \cdots & z_n & y_1 & \cdots & y_m \end{bmatrix}^T = \begin{bmatrix} z_1 & z_2 & \cdots & z_n & z_{n+1} & \cdots & z_{n+m} \end{bmatrix}^T$$

Similar to equation (6) and (7), the corresponding "dual dictionary" is given by:

$$z_N = (B^{-1}N)^T c_B - c_N + (B^{-1}N)^T z_B = z_N^* + (B^{-1}N)^T z_B \tag{10}$$

$$-\xi = -c_B^T B^{-1} b + (B^{-1}b)^T z_B = -\zeta^* - (x_B^*)^T z_B \tag{11}$$

where ξ denotes the objective function in the dual form.

For each dictionary, we set x_N and z_B as 0 and reads off the solutions to x_B and z_N. Next, we update the dictionary by removing one basic index and replacing it with a nonbasic index, then we get an updated dictionary. The simplex method produces a sequence of steps to "adjacent" bases such that the objective value of the objective function is always increasing at each step. Primal feasibility requires that $x_B \geq 0$, so while we update the dictionary, primal feasibility must always be satisfied. This process will stop when $z_N \geq 0$, and this is the optimality condition since it satisfies primal feasibility, dual feasibility and complementarity.

Now we introduce parametric simplex method [Vanderbei, 2008]. Generally speaking, we cannot make the initial primal feasible assumption ($b \geq 0$). The method we used here is adding some nonnegative perturbation times a positive parameter μ to the dictionary. Now equations (6), (7), (10) and (11) will become:

$$x_B = (x_B^* + \mu \overline{x}_B) - B^{-1} N x_N \tag{12}$$

$$\zeta = \zeta^* - (z_N^* + \mu \overline{z}_N)^T x_N \tag{13}$$

$$z_N = (z_N^* + \mu \overline{z}_N) + (B^{-1}N)^T z_B \tag{14}$$

$$-\xi = -\zeta^* - (x_B^* + \mu \overline{x}_B)^T z_B \tag{15}$$

When μ is large, the dictionary must be both primal and dual feasible ($x_B \geq 0$ and $z_N \geq 0$). At this point, we start to decrease μ. The smallest value of μ is given by

$$\mu^* = \min\{\mu : z_N^* + \mu \overline{z}_N \geq 0 \text{ and } x_B^* + \mu \overline{x}_B \geq 0\} \tag{16}$$

We interchange one basic variable and one nonbasic variable and update the dictionary so that the value of μ will be decreased. Eventually the value of μ will be decreased to zero which corresponds to the optimal value. For each update, we need to make sure that the primal and dual feasibility is satisfied.

Now we are ready to apply the parametric simplex method to solve CLIME. By setting $\beta = \beta^+ - \beta^-$ and $\|\beta\|_1 = \beta^+ + \beta^-$, equation (4) of CLIME can be written in the following form:

$$\min \beta_+ + \beta_- \quad \text{subject to: } \begin{bmatrix} \Sigma_n & -\Sigma_n \\ -\Sigma_n & \Sigma_n \end{bmatrix} \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix} \leq \begin{pmatrix} \lambda_n + e_i \\ \lambda_n - e_i \end{pmatrix} \tag{17}$$

This is clearly in the form given by equation (12) and (13) with the following identification:

$$b = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad c_N = \begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix} \quad \overline{z}_N = 0 \quad \overline{x}_B = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

Notice $x_B^* = B^{-1}b$ and $z_N^* = (B^{-1}N)^T c_B - c_N$.

3 Design and Implementation

The package `fastclime` consists of three parts: data generator, graph estimation and visualization.

M1. Data Generator: The function `fastclime.generator()` is able to generate multivariate Gaussian data with different graph structures, such as hub, cluster , band scale free, and Erdös-Rényi random graphs. The user is also able to control and adjust the sparsity level of the graph and the signal to noise ratio.

M2. Graph Estimation: `fastclime` is the main function used to estimate the graph. It takes two parameters. The first parameter can be either a data matrix or a sample covariance matrix. The second parameter is the required sparsity level. The function estimate the precision matrix column by column, and stops when the required sparsity level has been reached at each column. The estimator is based on a parametric simplex Linear Programming solver written in C. In order to maintain the speed of the program, the path length is limited to be less than 50. Therefore, when the required sparsity level is higher and the data dimension is huge, it will not be able to recover the required sparsity level. This estimator is designed to take advantage of parametric simplex method to recover sparse precision matrix only. When the precision matrix is not sparse, it will take a long time to recover and the result it not that meaningful.

The list of precision matrix is stored in "icov" and the list of adjacency matrix is stored in "path". "mu" includes the full path information for every column, with zero filled in when the sparsity level has been achieved in that column. "sparsity" shows the sparsity level at each path step. "df" is a matrix, whose row contains the number of nonzero coefficients along the solution path. "nlambda" is the length of the solution path.

M3. Graph Visualization: The plotting functions `fastclime.plot()` provides a method to plot the undirected graph at each path. Please note only adjacency matrix is allowed for this function. Because of the limitation of `igraph`, only up to 2000 nodes can be shown when using `fastclime.plot()`. `plot()` provides visualizations of the sparsity level versus the tuning parameter $/lambda_n$ in the first column. To view the full path of another column, one can view the "sparsity" and "lambda" component of the output object.

4 User Interface by Example

We illustrate the user interface by two examples. The first one is based on the data generated by `fastclime.generator()`,

```
> library(fastclime)                                  # Load the package huge
> L = fastclime.generator(n=200,d=50,graph="hub") # Generate data with hub structures
> out = fastclime(L$data)                             # Estimate the solution path
> fastclime.roc(out$path,L$theta)                     # Plot the ROC curve
> fastclime.plot(out$path[[3]])
> plot(out)
```

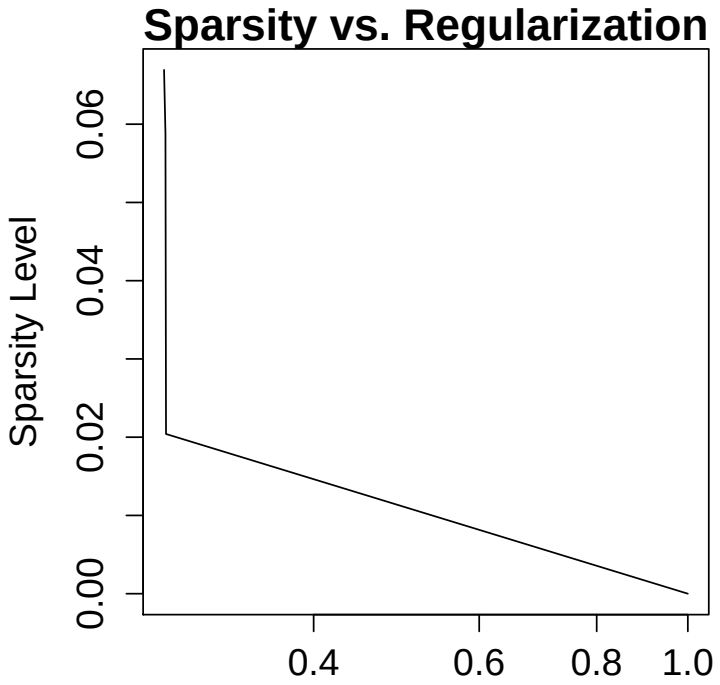


Figure 1: The solution path

In this example, we first generate 200 samples from a 50-dimensional Gaussian distribution with hub structure. All information, such as the true covariance matrix, sample covariance matrix, the true precision matrix and adjacency matrix are stored in object "L". We try to estimate the inverse covariance matrix by `fastclime`, we assume the sparsity level is about 0.1, which is the default value. The estimator we obtain is stored in an object called "out". We observe the result has a path length of 14. The roc curve and the solution path is shown in Figure 1 and Figure 2, respectively. The true sparsity level is 0.06, and we observe at path[[3]], we obtained the true graph in "hub" mode. Path[[1]] does nothing and path[[2]] simply recovers the relation of each variable with itself, so actually one further step recovers all the true correlation between the variables. The user has to be aware that as the path number increases, it is likely to get additional undirected graph edge, as shown in Figure 3(c).

The second example is based on some stock market data which is contained in the package. The data contains stock price from S&P 500 during the period between Jan 1st, 2003 and Jan 1st, 2008 It gives 1258 samples from 452 stocks which remained in the stock during the entire time period. After loading the data, we need to transform the data by calculating the log-ratio of the price at time t to price at time t-1, then we subtract the mean and adjust the variance to be one. We give a approximate sparsity ratio to be 0.05. The program automatically calculate the solution path up to a path of length 50. We reached the sparsity level 0.05 in the 30th path. The solution path for the first column is shown in Figure 4, and a few example of estimated graphs with corresponding sparsity level labeled are shown in Figure 5.

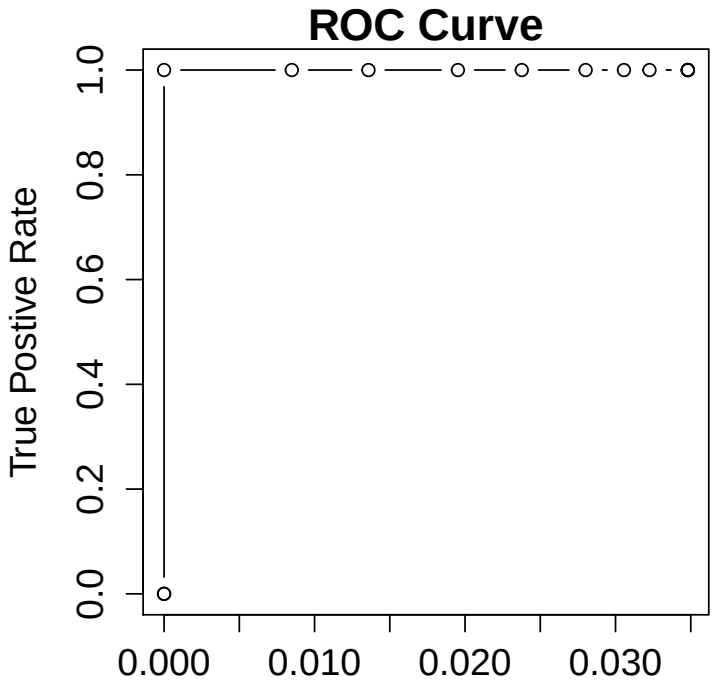
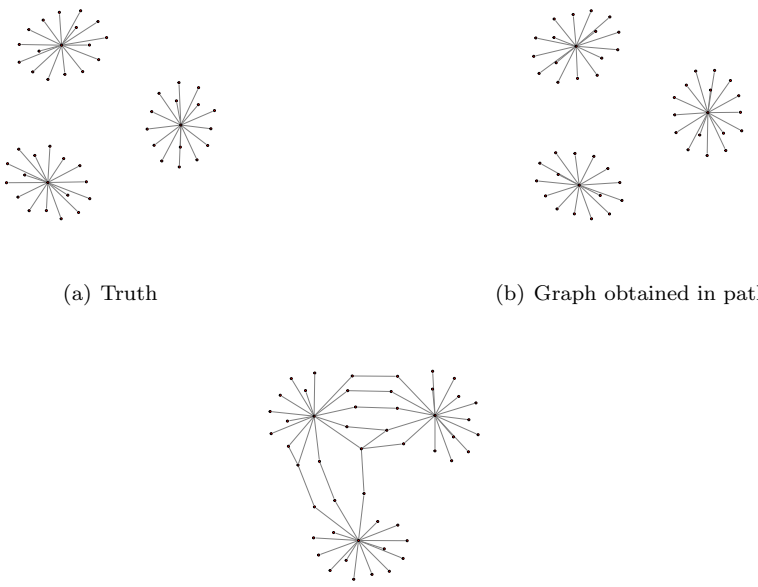


Figure 2: The ROC curve



(a) Truth

(b) Graph obtained in path3

(c) Graph obtained in path4

Figure 3: Graphs estimated by fastclime

```
> data(stockdata)                                          #Load the stock data
> Y = log(stockdata$data[2:1258,]/stockdata$data[1:1257,])  #Preprocessing
> out = fastclime(Y,0.06)                                  #Estimate the graph
> plot(out)
> fastclime.plot(out$path[[7]])
```

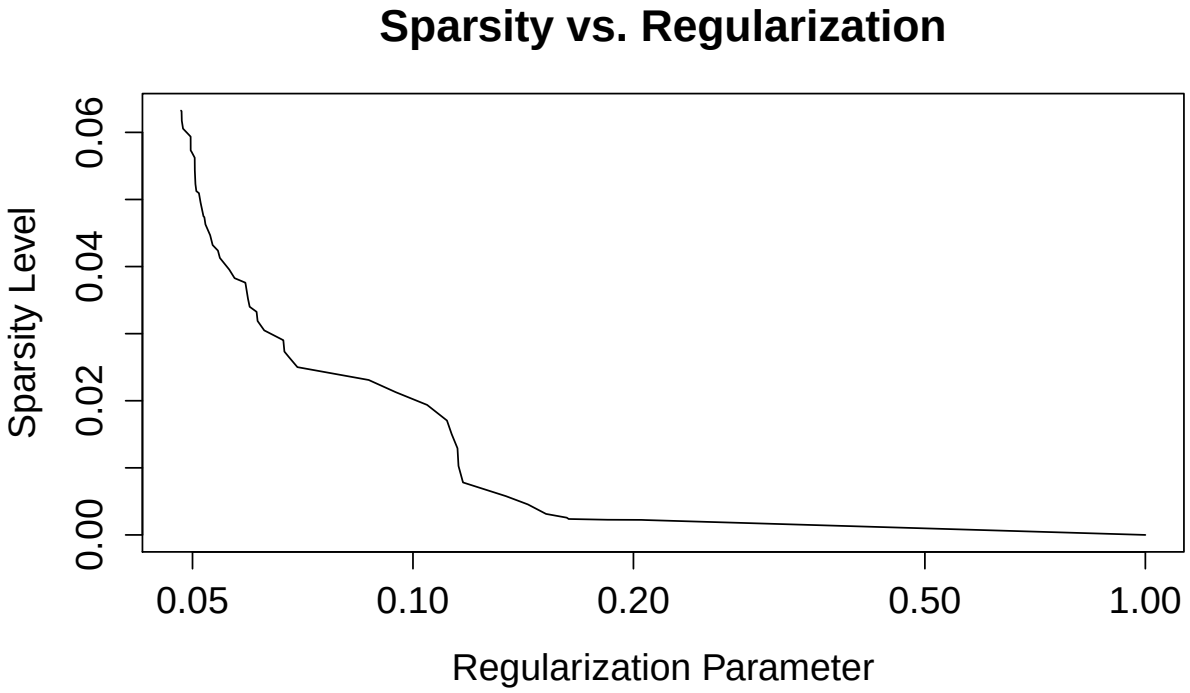
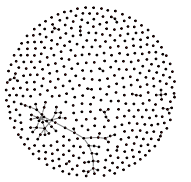
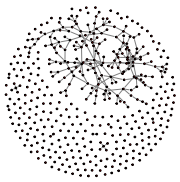


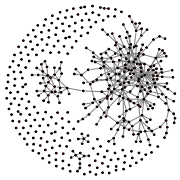
Figure 4: The solution path of the stock data



(a) sparsity=0.0031



(b) sparsity=0.0045



(c) sparsity=0.0058

Figure 5: Graphs estimated by fastclime based on the stock data

5 Performance Benchmark

We evaluate the timing performance of our package with comparison to the packages `flare` and `clime`. `flare` [Li et al., 2012] uses ADMM as the method to evaluate CLIME. We simulate the data from several multivariate normal distributions. We fix the sample size n = 200 and vary the data dimension p from 50 to 800. We generate our data by `fastclime.generator`, without any particular data structure. As can be seen from Table 1, `fastclime` performance significantly faster than `clime` when p is 50 or 100. When p is large, we are not able to obtain results directly from `clime` in one hour. We also notice that `fastclime` performances consistently better than `flare`, and it has a smaller deviation compared with `flare`. All experiments are carried on a PC with Intel Core i5-3320 2.6GHz processor.

Table 1: Average Timing Performance of Three Solvers

solve	p=50	p=100	p=200	p=400	p=800
clime	103.52(9.11)	937.37(6.77)	N/A	N/A	N/A
flare	0.632(0.335)	1.886(0.755)	10.770(0.184)	74.106(33.940)	763.632(135.724)
fastclime	0.248(0.0148)	0.928(0.0268)	9.928(3.702)	53.038(1.488)	386.880(58.210)

6 Conclusions

We developed a new package named `fastclime`, for conditional independence graph estimation and precision matrix recovery. The package is based on the CLIME and parametric simplex method. Compared with the existing package `clime`, it has many additional features: It is much faster and it can recover the full solution path column by column. We plan to maintain and support this package in the future.

References

O. Banerjee, L. E. Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation. *Journal of Machine Learning Research*, 9:485–516, 2008.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statitical learning via the alternating direction method of multipliers. *FTML*, pages 1–122, 2010.

T. Cai, W. Liu, and X. Luo. A constrained l_1 minimization approach to sparse precision matrix estimation. Technical report, University of Pennsylvania, 2010.

J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2007.

J. Friedman, T. Hastie, and R. Tibshirani. Applications of the lasso and grouped lasso to the estimation of sparse graphical models. Technical report, Stanford University, 2010.

X. Li, T. Zhao, X. Yuan, and H. Liu. An r package flare for high dimensional linear regression and precision matrix estimator. 2012.

R. Vanderbei. *Linear Programming, Fundations and Extensions*. Springer, 2008.