# The `fastclime` Package for Linear Programming and Constrained $l_1$-Minimization Approach to Sparse Precision Matrix Estimation in `R`

Haotian Pang * Han Liu † Robert Vanderbei ‡

October 28, 2013

**Abstract**

We develop an `R` package `fastclime` for solving a family of regularized linear programming (LP) problems. Our package efficiently implements the parametric simplex algorithm, which provides a scalable and sophisticated tool for solving large-scale linear programs. As an illustrative example, one use of our LP solver is to implement an important sparse precision matrix estimation method called *CLIME* (Constrained $L_1$ Minimization Estimator). Compared with existing packages for this problem such as `clime` and `flare`, our package has three advantages: (1) it efficiently calculates the full piecewise-linear regularization path; (2) it provides an accurate dual certificate as stopping criterion; (3) it is completely coded in C and is highly portable. This package is designed to be useful to statisticians and machine learning researchers for solving a wide range of problems.

## 1 The Parametric Simplex Method

In this section, we will follow the normal linear programming convention and use $x$ as the normal linear programming variable which has dimension $n$. Here we briefly describe the primal simplex method and introduces the parametric simplex methods.

We give the standard linear programming in inequality form:

$$\max c^T x \quad \text{subject to: } Ax \leq b, \quad x \geq 0 \quad x \in \mathbb{R}^n \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ are given.
For the primal simplex method, we require that $b \geq 0$. This property is called *primal feasibility*. We change the standard inequality form into equality form by adding *slack variables* $w$:

$$\max \begin{bmatrix} c & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \quad \text{subject to: } \begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} = b \quad x, w \geq 0 \quad x \in \mathbb{R}^n \quad w \in \mathbb{R}^m \tag{2}$$

For notation convenience, we rewrite the variable $x$ as:

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

The new variables, $x_{n+1}, ..., x_{n+m}$, are called *slack variables*. We separate the $m + n$ variables into two parts: *basic variables* and *nonbasic variables*. Basics variables are used to represent the nonbasic variables. Initially, the slack variables are basic variables and the original variables are nonbasic variables. We separate the matrix $\begin{bmatrix} A & I \end{bmatrix}$ into the basic part and the nonbasic part as well (with permutation). Initially the matrix $A$ corresponds to nonbasic variables and the identity matrix part corresponds to basic variables. The vector $\begin{bmatrix} c & 0 \end{bmatrix}$ is separated in the same fashion:

*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

$$\begin{bmatrix} A & I \end{bmatrix} = \begin{bmatrix} N & B \end{bmatrix} \quad \begin{bmatrix} x \\ w \end{bmatrix} = \begin{bmatrix} x_N \\ x_B \end{bmatrix} \quad \begin{bmatrix} c \\ 0 \end{bmatrix} = \begin{bmatrix} c_N \\ c_B \end{bmatrix}$$

Now the constraints of (2) will become: $Nx_N + Bx_B = b$. We denote the objective $c^T x$ as $\zeta$, then we have:

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

$$\begin{aligned} \zeta =& c^T x \\ =& c_B^T x_B + c_N^T x_N \\ =& c_B^T(B^{-1}b - B^{-1}Nx_N) + c_N^T x_N \\ =& c_B^T B^{-1}b - ((B^{-1}N)^T c_B - c_N)^T x_N \\ =& \zeta^* - (z_N^*)^T x_N \end{aligned} \tag{3}$$

(4)

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 equations (3) and (4) the *primal dictionary* associated with the current basis $B$. Primal feasibility requires that $x_B \geq 0$, which is initially guaranteed by $b \geq 0$ . We read off the specific values of $x_B$ and $\zeta$ by setting $x_N$ to be zero. For next iteration, we swap one variable from the basics part and one variable from the nonbasic part, and then we write down the dictionary and read off the values such that the objective value is always increasing.

The dual of the problem (1) is:

$$\min b^T y \quad \text{subject to: } A^T y \geq c, \quad y \geq 0 \quad y \in \mathbb{R}^m \tag{5}$$

As with the primal problem, We form by introducing slack variables here as well:

$$\min b^T y \quad \text{subject to: } A^T y - z = c, \quad y, z \geq 0 \quad y \in \mathbb{R}^m \quad z \in \mathbb{R}^n \tag{6}$$

The dual variables are 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 (3) and (4), the corresponding *dual dictionary* is given by:

$$z_N = (B^{-1}N)^T c_B - c_N + (B^{-1}N)^T z_B = z_N^* + (B^{-1}N)^T z_B \tag{7}$$

$$-\xi = -c_B^T B^{-1}b + (B^{-1}b)^T z_B = -\zeta^* - (x_B^*)^T z_B \tag{8}$$

where $\xi$ denotes the objective function in the dual form.

For each dictionary, we set $x_N$ and $z_B$ to 0 (complementarity) and read off the solutions to $x_B$ and $z_N$ according to (3) and (7). 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 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 $z_N \geq 0$ (dual feasibility), and this is the optimality condition since it satisfies primal feasibility, dual feasibility and complementarity.

Now we introduce the parametric simplex method [Vanderbei, 2008]. Generally speaking, we cannot make the initial primal feasible assumption ($b \geq 0$). The method we use here is to add some nonnegative perturbation times a positive parameter $\lambda$ to both objective function ($\bar{c}$) and the right hind side of the primal problem ($\bar{b}$). Now (1) becomes:

$$\max(c + \lambda\bar{c})^T x \quad \text{subject to: } Ax \leq b + \lambda\bar{b}, \quad x \geq 0 \quad x \in \mathbb{R}^n \tag{9}$$

The dictionary of equations (3), (4), (7) and (8) will become:

$$x_B = (x_B^* + \lambda \overline{x}_B) - B^{-1}Nx_N \tag{10}$$

$$\zeta = \zeta^* - (z_N^* + \lambda \overline{z}_N)^T x_N \tag{11}$$

$$z_N = (z_N^* + \lambda \overline{z}_N) + (B^{-1}N)^T z_B \tag{12}$$

$$-\xi = -\zeta^* - (x_B^* + \lambda \overline{x}_B)^T z_B \tag{13}$$

with $\overline{x}_B = B^{-1}\overline{b}$ and $\overline{z}_N = (B^{-1}N)^T \overline{c}_B - \overline{c}_N$.

we choose $\overline{b}$ and $\overline{c}$, so that when $\lambda$ is large, the dictionary will be both primal and dual feasible ($x_B^* + \lambda \overline{x}_B \geq 0$ and $z_N^* + \lambda \overline{z}_N \geq 0$). At this point, we start to decrease $\lambda$. The smallest value of $\lambda$ is given by

$$\lambda^* = \min\{\lambda : z_N^* + \lambda \overline{z}_N \geq 0 \text{ and } x_B^* + \lambda \overline{x}_B \geq 0\} \tag{14}$$

We interchange one basic variable and one nonbasic variable and update the dictionary in a way so that the value of $\lambda$ will be decreased. Eventually the value of $\lambda$ will be decreased to zero which corresponds to the original problem. For each update, we need to make sure that the primal and dual feasibility is satisfied. The detail of one iteration of the parametric simplex method can be found in page 119-121 of [Vanderbei, 2008]. This algorithm must stop in finite time because: (1) the value of $\lambda$ is decreasing for each iteration; (2) the optimal value corresponding to $\lambda$ is 0 (3) there are finitely many bases and it indicates that there are only finitely many steps (unless it cycles). The linear programming solver based on the algorithm described above is contained in our package.

This methods allows us to solve a full range of linear programming based regularized learning problems since the parameter used here corresponds exactly to the regularization parameter in many learning problems. In fact, if an regularized learning problem can be constructed in form (9), then the entire solution path of that problem can be obtained by solving the LP only once with our parametric simplex method. Therefore, we would like to apply this method to learning problems such as sparse precision matrix estimation.

In Section 2, we introduce CLIME and apply the parametric simplex method to solve CLIME. In Section 3, we introduce the design of our software and provide some examples. In Section 4, numerical benchmarks and comparisons with other implementations are provided.

# 2 CLIME

In this section, Constrained $l_1$ Minimization Approach to Sparse Precision Matrix Estimation (CLIME) will be discussed and the normal notation for statistical learning will be used.

Estimating large covariance or precision matrices is a fundamental problem which has many applications in modern statistics and machine learning. This problem is formulated as follows: Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ be $n$ observations of a $d$-dimensional random vector $\boldsymbol{X} = (X_1, \ldots, X_d)^T$. Without loss of generality, we assume $\mathbb{E}\boldsymbol{X} = 0$. We denote the population covariance matrix $\Sigma = \mathbb{E}\boldsymbol{X}\boldsymbol{X}^T$ and the precision matrix $\Omega = \Sigma^{-1}$. Our task is to estimate $\Omega$ even when the dimension $d$ may be larger than the sample size $n$.

Sparse precision matrices are closely related to undirected graphs. Under a Gaussian model, the sparse precision matrix $\Omega$ encodes the conditional independence relationships among the variables $X_1, \ldots, X_d$. More specifically, we define an undirected graph $G = (V, E)$, where $V$ contains nodes corresponding to the $d$ variables in $\boldsymbol{X}$ and the edge $(j, k) \in E$ if and only if $\Omega_{jk} \neq 0$. Let $\boldsymbol{X}_{\setminus\{j,k\}} = \{\boldsymbol{X}_\ell : \ \ell \neq j, k\}$. We have $X_j$ is independent of $X_k$ given $\boldsymbol{X}_{\setminus\{j,k\}}$ for all $(j, k) \notin E$. Therefore, the graph estimation problem is equivalent to estimating the sparse precision matrix $\Omega$. Besides graphical models, the estimated sparse precision matrix can also be applied for high-dimensional discriminant analysis and principal component analysis.

Recently, several sparse precision matrix estimation methods have been proposed. See [Friedman et al., 2007a, 2010a, Liu et al., 2009, 2010]. One famous estimator for the precision matrix is called

$l_1$ regularized log-determinant program [Banerjee et al., 2008], and it is a Maximum Likelihood Estimation (MLE)-type method:

$$\Omega = \text{ argmin trace}(\Sigma_n X) - \log(\det(X)) + \lambda \|X\|_1, \quad \text{subject to: } X \succ 0 \tag{15}$$

This estimator can be solved efficiently by solvers such as Graphical Lasso (GLASSO)[Friedman et al., 2007b, 2010b] and QUIC [Hsieh et al., 2011].
Another particular interesting method called CLIME [Cai et al.], and is based on linear programming. Tt intends to solve the following problem:

$$\min \|\Omega\|_1 \quad \text{subject to: } |\Sigma_n \Omega - I_d|_{\max} \leq \lambda, \quad \Omega \in \mathbb{R}^{d \times d} \tag{16}$$

where $\lambda$ is defined as a tuning parameter, $\Sigma_n$ is the sample covariance matrix and $I_d$ is the $d$-dimensional identity matrix. This minimization problem can be further decomposed into d smaller problems by simply recovering the precision matrix column by column:

$$\min \|\boldsymbol{\beta}\|_1 \quad \text{subject to: } |\Sigma_n \boldsymbol{\beta} - e_i|_\infty \leq \lambda \quad \boldsymbol{\beta} \in \mathbb{R}^d \tag{17}$$

where $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{d} |\beta_j|$ and $e_i \in \mathbb{R}^d$ is the $i$-th basis vector.

Of course, the estimator we obtain by solving these series of linear programming problems are not symmetric, but we can simply take the smaller value of $\widehat{\Omega}_{ij}$ and $\widehat{\Omega}_{ji}$ as the value chosen by the estimator. The result can be shown to be positive definite with high probability. The error between the estimator $\hat{\Omega}$ and $\Omega$ satisfies $\|\hat{\Omega} - \Omega\|_2 = O_p(s\sqrt{\log(d)/n})$ and $|\hat{\Omega} - \Omega|_\infty = O_p(\sqrt{\log(d)/n})$, where $s$ denotes the sparsity of the precision matrix. Moreover, it has been shown numerically that, in terms of accuracy, CLIME uniformly outperforms MLE-type estimator such as GLASSO [Cai et al.]. The reason is very obvious: there is a primal-dual gap to guarantee the machine precision in linear programming based approach while the stopping criterion for MLE-type estimator is to simply control the difference of norm during each iteration. This is the reason why CLIME is particularly important and useful.

Now we are ready to apply the parametric simplex method to solve CLIME. By setting $\boldsymbol{\beta} = \boldsymbol{\beta}^+ - \boldsymbol{\beta}^-$ and $\|\boldsymbol{\beta}\|_1 = \boldsymbol{\beta}^+ + \boldsymbol{\beta}^-$ and let $\boldsymbol{\beta}^+ \geq 0$ $\boldsymbol{\beta}^- \geq 0$, and one of $\boldsymbol{\beta}^+$, $\boldsymbol{\beta}^-$ be 0, equation (18) of CLIME can be written in the following form:

$$\min \boldsymbol{\beta}^+ + \boldsymbol{\beta}^- \quad \text{subject to: } \begin{pmatrix} \Sigma_n & -\Sigma_n \\ -\Sigma_n & \Sigma_n \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}^+ \\ \boldsymbol{\beta}^- \end{pmatrix} \leq \begin{pmatrix} \lambda + e_i \\ \lambda - e_i \end{pmatrix} \tag{18}$$

Equation (19) is clearly an initial setup of a parametric simplex problem. Comparing it with (9), we have the following identifications:

$$A = \begin{pmatrix} \Sigma_n & -\Sigma_n \\ -\Sigma_n & \Sigma_n \end{pmatrix} \quad b = \begin{pmatrix} e_i \\ -e_i \end{pmatrix} \quad c = \begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix} \quad \overline{c} = 0 \quad \overline{b} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

By constructing this setup and plugging the problem into our parametric simplex linear programming solver, we are able to recover the full piecewise-linear regularization path in a relatively short time.

# 3 Design, Implementation and Examples

The package `fastclime` consists of two major parts: a pair of linear programming solvers and a series of functions used to estimate precision matrix: data generator, graph estimation, graph visualization.

For the data generate part, the function `fastclime.generator` is able to generate multivariate Gaussian data with different graph structures.

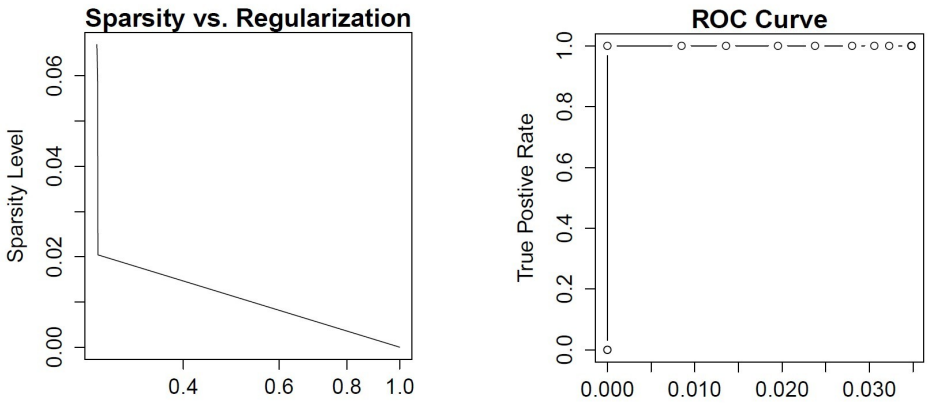
`fastclime` is the main function used to estimate the graph. It takes three parameters. The first parameter can be either a data matrix or a sample covariance matrix. The second parameter

is the required sparsity level. The third parameter is the designed iteration numbers required for each column (the length of the solution path). The function estimates the precision matrix column by column, and stops when the required sparsity level has been reached at each column or the designed iteration number has been reached. The estimator is based on a parametric simplex linear programming solver written in C. In order to maintain the speed of the program, the default path length is set to be 50. The user must be very careful when asking for a large solution path when the dimension $d$ is large. This estimator is designed to take advantage of the 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 meaningful.
The output object of the main function has several components. The list of precision matrices is stored in *icov* and the list of adjacency matrices 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. For each column, we calculate at what *mu* value the solution changes then store that value. *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.

The plotting functions `fastclime.plot` provides a method to plot the undirected graph at each path. Please note only the adjacency matrix is allowed for this function. `plot` provides visualizations of the sparsity level versus the tuning parameter $\lambda$ in the first column and fastclime.roc is used to plot the region of convergence (ROC) curve.

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
> 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)
```

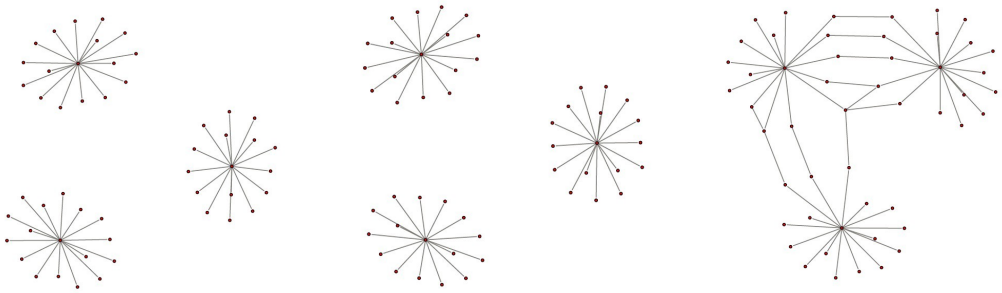



(a) The solution path

(b) The ROC curve

Figure 1: The solution path and the ROC curve of the first example

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, the sample covariance matrix, the true precision matrix and the adjacency matrix are stored in object $L$. We try to estimate the inverse covariance matrix by `fastclime`, and 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 Region of Convergence (ROC) curve and the solution path is shown in Figure 1(a) and Figure 1(b), respectively. As shown in Figure 1(a), this method gives a almost perfect ROC curve. The true sparsity level is 0.06. After three iterations, it recovers all the tree correlation between the variables. The user has to be aware that as the path length increases, it is likely to obtain additional undirected edges in the graph, as shown in Figure 2(c).



(a) Truth (b) Graph obtained in 3 iterations (c) Graph obtained in 4 iterations

Figure 2: Graphs estimated by fastclime

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 (n) from 452 stocks (d). We give a approximate sparsity ratio to be 0.05. The program automatically calculates the solution path. We reach the sparsity level 0.05 after 30 iterations. The solution path for the first column is shown in Figure 3, and a few example of estimated graphs with corresponding sparsity level labeled are shown in Figure 4.

```
> 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]])
```

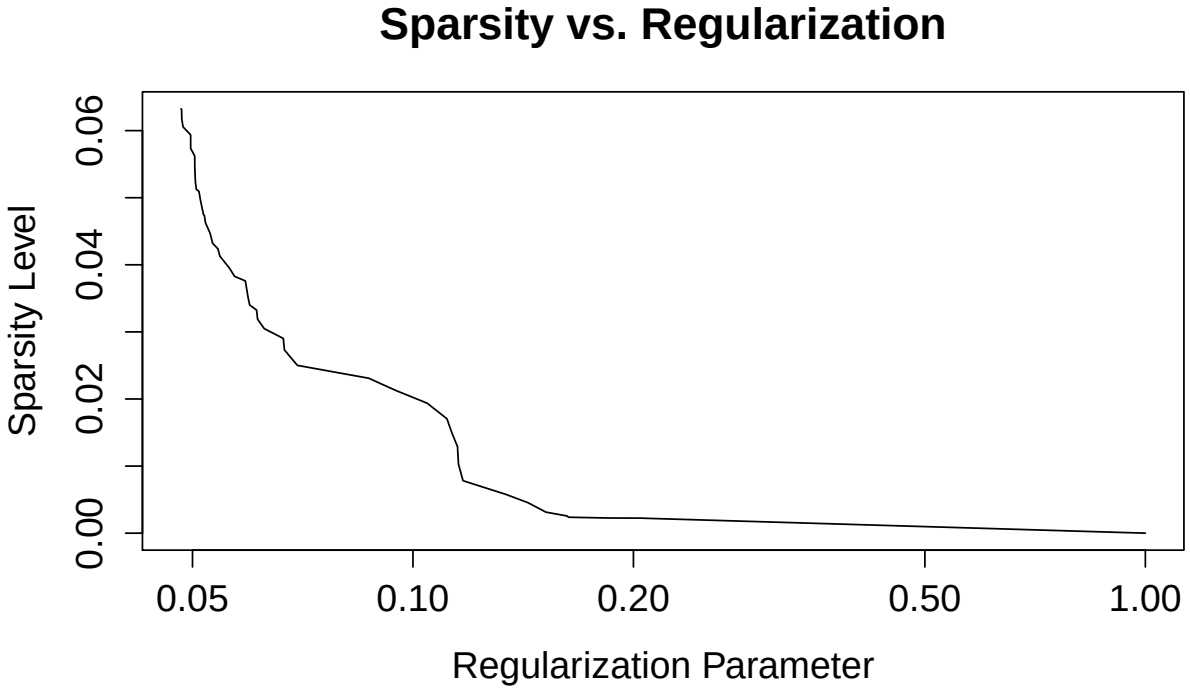
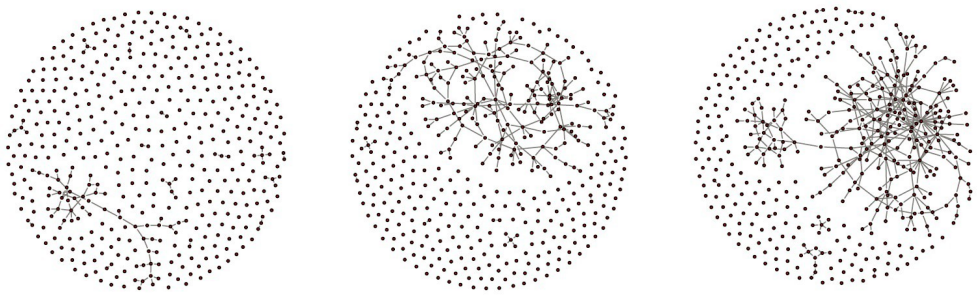



Figure 3: The solution path of the stock data

The linear programming solver function `fastlp` is used to solve a general linear programming problem in standard inequality form (1). The solver will automatically give a random perturbation and solve it by the parametric simple method. If the original problem is not in this form, the



(a) sparsity=0.0031 (b) sparsity=0.0045 (c) sparsity=0.0058

Figure 4: Graphs estimated by fastclime based on the stock data

user has to convert it into this form. For example, if the original problem has an equation, then this equation can be separated into two inequalities. If the original variables are not nonnegative, then the trick mentioned in last section to derive (19) can be used to make sure the variables be nonnegative. `Fastlp` has four parameters. The first three are the objective vector (length $n$), the constraint matrix (dimension $m \times n$) and the right hand side vector (length $m$). The last parameter lambda is used to specify the stopping rule for the linear programming solver. Whenever the calculated parameter $\lambda$ in the solver is smaller than this value, the program will stop at that point and the function will give the optimal solution corresponding to that $\lambda$ The default value of lambda is zero, which corresponds to the optimal value of the original LP problem. The function will also indicate if the input problem is unbounded or infeasible. For example, we generate a random feasible LP problem and solves it by `fastlp`.

```
> n=100
> A=matrix(mvrnorm(n^2,0, 1),nrow=n)
> x=runif(n, min = 0, max = 5)
> y=runif(n, min = 0, max = 5)
> w=runif(n, min = 0, max = 5)
> z=runif(n, min = 0, max = 5)
> b=A%*%x+w
> c=t(A)%*%y-z
> out1<-fastlp(c,A,b)
```

Another linear programming solver function `paralp` is used to solve a parameterized LP problem in form (9). It takes six parameters. The only difference is that this time the user can specify the two perturbation vectors ($\overline{b}$ and $\overline{c}$). The rest four parameters have the same meaning as in `fastlp`. Again, the function stops at the value of lambda provided by the user and give the optimal solution corresponding to that lambda. Notice the perturbation vectors must be nonnegative in order to apply the parametric simplex method. Here is a simple example:

```
> c<-c(1,2,1,1)
> b<-c(8,12,18)
> A<-matrix(c(2,2,3,1,2,1,5,0,2,1,4,0),nrow=4)
> b_bar<-c(1,1,1)
> c_bar<-c(1,1)
> opt2<-paralp(c,A,b,c_bar,b_bar)
```

# 4 Performance Benchmark

Since CLIME has many advantages over MLE-type methods, we focus on the comparison between packages based on CLIME method. We evaluate the timing performance of our package with comparison to the packages `flare` and `clime` while estimating the precision matrix. `Flare` [Li et al., 2012] uses Alternating Direction Method of Multiplier (ADMM) as the method to evaluate CLIME and `clime` solves a series of LP problems for different values of $\lambda$. We simulate the data from several multivariate normal distributions. We fix the sample size $n$ to be 200 and vary the

data dimension $d$ from 50 to 800. We generate our data by `fastclime.generator`, without any particular data structure.

The error tolerance (primal and dual gap) used for `clime` and `fastclime` is $10^{-5}$ and the error tolerance (differences between iterations) used for `flare` is also $10^{-5}$. Our package calculates its own $\lambda$ sequences while solving the LP only once, the $\lambda$ sequence is stored in the output. These $\lambda$ sequences used for the other two methods are all the default sequences from the packages. To be specific, the $\lambda$ sequence used by `clime` is of length 100 from 0.8 to 100 and the sequence used by `flare` is of length 10 from $0.5\lambda_{\max}$ to $\lambda_{\max}$. The default value of $\lambda_{\max}$ is the minimum regularization parameter, which yields an all-zero off-diagonal estimates.

As can be seen from Table 1, `fastclime` performance significantly faster than `clime` when $d$ is 50 or 100. When $d$ is large, we are not able to obtain results directly from `clime` in one hour. We also notice that, in most cases, `fastclime` performances better than `flare`, and it has a smaller deviation compared with `flare`. The units are in seconds and all experiments are carried on a PC with Intel Core i5-3320 2.6GHz processor and the R version used is 2.15.0.

Table 1: Average Timing Performance of Three Solvers

| solve | d=50 | d=100 | d=200 | d=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) |

# 5 Conclusions

We developed a new package named `fastclime` for generic and parameterized linear programming problems as well as conditional independence graph estimation and precision matrix recovery. The package is based on CLIME and the parametric simplex method. Compared with the existing package `clime`, it has many additional features: it is much faster and it can recover the full piece-wise linear regularization solution path. We hope the CRAN community could benefit from our contributions. 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.

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

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2), 2007a.

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

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 2010a.

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, 2010b.

C-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. Ravikumar. Sparse inverse covariance matrix estimation using quadratic approximation. *Advances in Neural Information Processing Systems*, 24, 2011.

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

H. Liu, J. Lafferty, and L. Wasserman. The nonparanormal semiparametric estimation of high dimensional undirected graphs. *Journal of Machine Learning Research*, 10:2295–2328, 2009.

H. Liu, K. Roeder, and L. Wasserman. Stability approach to regularization selection for high dimensional graphical models. *Advances in Neural Information Processing Systems*, 2010.

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