A Parametric Simplex Approach to Statistical Learning Problems

Haotian Pang∗ Tuo Zhao† Robert Vanderbei‡ Han Liu§

Abstract

In this paper, we show that the parametric simplex method is an efficient algorithm for solving various statistical learning problems that can be written as linear programs parametrized by a so-called regularization parameter. The parametric simplex method offers significant advantages over other methods: (1) it finds the complete solution path for all values of the regularization parameter by solving the problem only once; (2) it provides an accurate dual certificate stopping criterion; (3) for Lasso-type problems, it produces the sparse solutions in very few iterations. The learning problems we looked at include Dantzig selector estimation, LAD-Lasso, constrained ℓ1 minimization estimation (CLIME) for sparse precision matrix, linear programming discriminant rule for sparse linear discriminant analysis (LDA) as well as differential network estimation. We provide details on how to apply the parametric simplex method to these problems. Numerical performance of our method applied to these learning problems is investigated using simulated data.

1 Introduction

A broad spectrum of problems in the areas of statistics and machine learning can be formulated as high-dimensional optimization problems involving some parameter whose appropriate value is not known a priori. As an example, Lasso-type problems start with a traditional statistical problem in which some error term is minimized and then encourage a sparse solution by adding a weighted sparsity-inducing term to the minimization problem. Unfortunately, an appropriate choice of the

∗Department of Electrical Engineering, Princeton University, Princeton, NJ 08540, USA; Email: hpang@princeton.edu
†Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA; Email: tuoz@princeton.edu
‡Department of Operational Research and Financial Engineering, Princeton University, Princeton, NJ 08540, USA; Email: rvdb@princeton.edu, This work is supported in part by the U. S. Office of Naval Research under award number N000141612162.
§Department of Operational Research and Financial Engineering, Princeton University, Princeton, NJ 08540, USA; Email: hanliu@princeton.edu
weighting factor is not known ahead of time. Hence, one assumes that the algorithm will be tuned to a specific type of problem so that with future instances one can guess a reasonable starting value for the weighting parameter and get to the desired solution in a small number of trials. But, each trial involves solving a problem from scratch and is therefore time-consuming on high-dimensional problems. And, there is a chance that the result obtained is not actually the desired result. These are serious drawbacks to the existing methods for solving these types of problems. We will show in this paper that the parametric simplex method using the unknown weighting factor as the “parameter” provides a powerful and efficient way to address these shortcomings in cases where the underlying optimization problem is linear.

Geometrically speaking, simplex methods move from one intersection of hyperplanes to an “adjacent” intersection in search of an optimal solution. If the intersection is feasible for the problem, then it is a vertex of the polytope of feasible solutions. But, many variants of the simplex method, including the parametric simplex method, do not insist on feasibility at every iteration. Algebraically, the algorithm involves moving from one partition of the variables into basic and nonbasic variables to another. Each partition deviates from the previous in that one basic variable gets swapped with one nonbasic variable in a process called pivoting. Although in the worst-case, the simplex method can take an exponential number of pivots to find an optimal solution, in practice, the number of pivots is roughly linear in the number of variables in the problem. Even better, for Lasso-type problems, the desired sparse solution can often be found in very few pivots. Although there is no guaranteed that the number of pivot iterations will be small, we show numerically that desired sparse solutions can be found in a very few iterations.

In addition to the Lasso-type sparseness problem, there is a broad spectrum of applications in machine learning problems in which there is a regularization parameter used to tune the desired solution. We will extend the ideas described above to a variety of problems in machine learning that can be written as a linear programming problem with a regularization parameter.

To be specific, we are interested in parametric linear programming problems with equality or inequality constraints. The equality parametric linear programming is defined as the following:

\[
\max_x (c + \lambda \bar{c})^T x \quad \text{subject to: } Ax = b + \lambda \bar{b}, \ x \geq 0, \quad (1.1)
\]

The inequality parametric linear programming is defined as:

\[
\max_x (c + \lambda \bar{c})^T x \quad \text{subject to: } Ax \leq b + \lambda \bar{b}, \ x \geq 0. \quad (1.2)
\]

It is well-known that, if a problem can be cast as a parametric linear programming problem, then the parametric simplex method can solve the problem for all values of the parameter in the same time as another variant of the simplex method would require to solve for just one value of the parameter (Vanderbei, 1995; Murty, 1983). Recently, Yao and Lee (2014) applied the “parametric cost linear programming” idea with a perturbation on the right-hand-side to some problems in statistics. We extend this method to perturbation on both right-hand-side and the objective as described in Vanderbei (1995) so that a variety of problems in machine learning can benefit from it.
In addition to the computational benefits achieved by the parametric simplex method, some statistical problems also benefit from factoring a dense constraint matrix into a product of sparse matrices. Further dramatic computational speedups can be obtained by exploiting this factorization.

The paper is organized as follows: in §2 we describe a number of statistical problems that can be formulated as parametric linear programming problems. For completeness, we also review the primal simplex method. In §3 we present a detailed description of the parametric simplex method that starts from the (infeasible) all zero solution and proceeds to the desired sparse answer. In §4, we apply the parametric simplex method to the Dantzig selector as well as the differential network problem and present our numerical experiments. In §5, we summarize our results. Technical proofs are deferred to §6.

2 Background

We start with some basic notations and definitions. We use boldface for vectors and matrices. In particular, 1 is the vector of all ones and 0 is the zero vector. For a vector \(a = (a_1, ..., a_d)^T \in \mathbb{R}^d\), we let \(\|a\|_p = \left(\sum_j |a_j|^p\right)^{1/p}\) denote the p-norm. In particular, the 0-“norm”, \(\|a\|_0\), is the number of nonzero elements of \(a\), \(\|a\|_1 = \sum_j |a_j|\), \(\|a\|_2 = \sum_j a_j^2\) and \(\|a\|_\infty = \max_j |a_j|\). When comparing vectors, “\(\geq\)” and “\(\leq\)” are defined component-wise. For a square matrix \(A \in \mathbb{R}^{d \times d}\) with entries \(a_{jk}\), we use \(\|A\|\) to denote element-wise norms and \(\|A\|\) to denote matrix norms. So, \(\|A\|_0\) is the number of nonzero entries in \(A\), \(\|A\|_1 = \sum_{j,k}|a_{jk}|\), and \(\|A\|_\infty = \max_{j,k} |a_{jk}|\). For the matrix norms, we define \(\|A\|_1 = \max_k \sum_j |a_{jk}|\), \(\|A\|_\infty = \max_j \sum_k |a_{jk}|\), \(\|A\|_2 = \max_{\|a\|_2 \leq 1} \|Aa\|_2\) and \(\|A\|_F^2 = \sum_{j,k} a_{jk}^2\). We denote \(A_{i\setminus j}\) as the submatrix of \(A\) with \(i\)-th row and \(j\)-th column removed. We denote \(A_{\setminus ij}\) as the \(i\)-th row of \(A\) with its \(j\)-th entry removed and \(A_{\setminus ij}\) as the \(j\)-th column of \(A\) with its \(i\)-th entry removed. For any subset \(G\) of \(\{1, 2, ..., d\}\), we let \(A_G\) denote the submatrix of \(A \in \mathbb{R}^{p \times d}\) consisting of the corresponding columns of \(A\). The notation \(A \geq 0\) means all of \(A\)’s entries are nonnegative. Similarly, for a vector \(a \in \mathbb{R}^d\), we let \(a_G\) denote the subvector of \(a\) associated with the indices in \(G\). Finally, \(I_d\) denotes the \(d\)-dimensional identity matrix and \(e_i\) denotes vector that has a one in as it’s \(i\)-th element and is zero elsewhere. In a large matrix, we leave a submatrix blank when all of its entries are zeros.

Many sparse learning problems are formulated as convex programs in a generic form:

\[
\min_{\beta} \| \beta \|_1 \quad \text{subject to} \quad \| \nabla L(\beta) \|_\infty \leq \lambda, \quad (2.1)
\]

where \(L(\beta)\) is a smooth convex loss function, \(\nabla L(\beta)\) is the gradient of \(L(\beta)\), and \(\lambda > 0\) is a regularization parameter controlling the trade-off between \(\| \beta \|_1\) and \(\| \nabla L(\beta) \|_\infty\). Alternatively, if we set both the convex loss function and the regularization term in \(\ell_1\) norm, we can write the same problem in a regularization form:

\[
\min_{\beta} \| \nabla L(\beta) \|_1 + \lambda \| \beta \|_1. \quad (2.2)
\]

In this case, we replace the normal \(l_2\)-loss function by the non-smooth \(\ell_1\)-loss function and \(\| \beta \|_1\) serves as a convex penalty function which measures the model complexity. Both (2.1) and (2.2) are
naturally suited for our method. The complete piecewise-linear solution path as a function of the regularization parameter can be computed by a variant of the parametric simplex method.

We will introduce several loss functions associated with various different sparse learning problems.

### 2.1 Sparse Linear Regression

We start with the sparse linear regression problem. Let \( y \in \mathbb{R}^n \) be a response vector and let \( X \in \mathbb{R}^{n \times d} \) be the design matrix. We consider a linear model \( y = X\beta^0 + \epsilon \), where \( \beta^0 \in \mathbb{R}^d \) is the unknown regression coefficient vector, and \( \epsilon \) is the observational noise vector. In high dimensional statistical inference, \( d \) is much larger than \( n \): \( d \gg n \). Even though \( d \) is large, it is assumed that in reality most entries in \( \beta^0 \) are zero: \( \beta^0 \) is a sparse vector: \( \| \beta^0 \|_0 = s \ll n \). Statisticians have proposed many different regression methods to obtain sparse estimators of \( \beta_0 \) including Lasso (Tibshirani, 1996), Elastic Net (Zhou and Hastie, 2005), Dantzig Selector (Candes and Tao, 2007) and LAD-Lasso (Wang et al., 2007). Of these methods, we will focus on the Dantzig selector and LAD-Lasso as these can be computed by solving a simple linear program with a tuning parameter.

The Dantzig selector is defined as the solution to the following convex program:

\[
\min_{\beta} \| \beta \|_1 \quad \text{subject to} \quad \| X^T (y - X\beta) \|_\infty \leq \lambda. \tag{2.3}
\]

Using a common modeling trick, (2.3) can be rewritten as a parametric linear program: let \( \beta = \beta^+ - \beta^- \) where \( \beta^+ \) is the vector of positive parts of the elements of \( \beta \) and \( \beta^- \) is the corresponding vector of negative parts. By definition, the positive and negative parts are nonnegative and complementary to each other. In addition, it is easy to see that \( \| \beta \|_1 = 1^T \beta^+ + 1^T \beta^- \). It turns out that we can drop the complementarity condition because, at optimality, at least one of the two parts will be zero. Hence, (2.3) can be rewritten as

\[
\min_{\beta^+, \beta^-} 1^T (\beta^+ + \beta^-) \quad \text{subject to} \quad \begin{pmatrix} X^T X & -X^T X \\ -X^T X & X^T X \end{pmatrix} \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix} \leq \begin{pmatrix} \lambda 1 + X^T y \\ \lambda 1 - X^T y \end{pmatrix}, \quad \beta^+, \beta^- \geq 0. \tag{2.4}
\]

Now we see clearly that (2.4) fits into the parametric linear program form as (1.2) with

\[
A = \begin{pmatrix} X^T X & -X^T X \\ -X^T X & X^T X \end{pmatrix}, \quad b = \begin{pmatrix} X^T y \\ -X^T y \end{pmatrix}, \quad c = -1, \quad \bar{b} = 1, \quad \bar{c} = 0, \quad x = \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix}.
\]

Alternatively, the regression problem can be solved in a LAD-Lasso form:

\[
\min_{\beta} \| X\beta - y \|_1 + \lambda \| \beta \|_1. \tag{2.5}
\]

Letting \( t = X\beta - y \) and separating both \( t \) and \( \beta \) into positive and negative parts, the problem
becomes
\[
\min_{\beta^+, \beta^-, t^+, t^-} \quad \mathbf{1}^T (t^+ + t^-) + \lambda \mathbf{1}^T (\beta^+ + \beta^-) \\
\text{subject to } \begin{pmatrix} \mathbf{X} & -\mathbf{X} & -\mathbf{I} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \beta^+ \\ \beta^- \\ t^+ \\ t^- \end{pmatrix} = \mathbf{y} \quad \beta^+, \beta^-, t^+, t^- \geq 0.
\]  
\]

Clearly, (2.6) fits into form (1.1) with
\[
A = \begin{pmatrix} \mathbf{X} & -\mathbf{X} & -\mathbf{I} & \mathbf{I} \end{pmatrix}, \quad b = \mathbf{y}, \quad c = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad \bar{b} = 0, \quad \bar{c} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad x = \begin{pmatrix} \beta^+ \\ \beta^- \\ t^+ \\ t^- \end{pmatrix}.
\]

**2.2 Sparse Linear Discriminant Analysis**

The second problem we consider is the sparse linear discriminant analysis (LDA) for classification problem. Here, we assume that there are two classes of observations generated from a \(d\)-dimensional normal distribution with different means \(\mu_0^1, \mu_0^2\) and the same covariance matrix \(\Sigma^0\) and precision matrix \(\Omega^0 = (\Sigma^0)^{-1}\). Let \(\mathbf{x}_1, ..., \mathbf{x}_{n_1} \in \mathbb{R}^d\) be \(n_1\) independent and identically distributed samples from \(N(\mu_0^1, \Sigma^0)\) (class 1) and \(\mathbf{y}_1, ..., \mathbf{y}_{n_2} \in \mathbb{R}^d\) be \(n_2\) independent and identically distributed samples from \(N(\mu_0^2, \Sigma^0)\) (class 2). We then calculate sample means by
\[
\bar{x} = \frac{1}{n_1} \sum_{i=1}^{n_1} \mathbf{x}_i, \quad \bar{y} = \frac{1}{n_2} \sum_{i=1}^{n_2} \mathbf{y}_j,
\]

and we define
\[
\delta = (\bar{x} - \bar{y}).
\]

The sample covariance matrix is given by
\[
S = \frac{1}{n_1 + n_2} \left( \sum_{j=1}^{n_1} (\mathbf{x}_j - \bar{x})(\mathbf{x}_j - \bar{x})^T + \sum_{j=1}^{n_2} (\mathbf{y}_j - \bar{y})(\mathbf{y}_j - \bar{y})^T \right).
\]

Given a new sample \(\mathbf{z} \in \mathbb{R}^d\), Fisher’s linear discriminant rule classifies it to class 1 if
\[
(z - \mu^0)^T \Omega^0 \delta^0 \geq 0,
\]
where \(\mu^0 = (\mu_1 + \mu_2)/2\) and \(\delta^0 = \mu_1 - \mu_2\), and otherwise to class 2. This rule is proposed based on equal prior probabilities for both classes. In high dimensions, we have \(d \gg (n_1 + n_2)\). To make the estimation manageable, we assume that \(\|\beta^0\|_0 = s \ll (n_1 + n_2)\), where \(\beta^0 = \Omega^0 \delta^0\) (Cai and Liu, 2011). This leads to the following convex optimization problem:
\[
\min_{\beta} \|\beta\|_1 \quad \text{subject to } \|S\beta - \bar{\delta}\|_\infty \leq \lambda.
\]
Similar to sparse linear regression, (2.11) yields a sparse estimator. We then adopt the same reparametrization trick, and rewrite (2.11) as a parametric linear programming problem:

$$\min_{\beta^+, \beta^-} 1^T (\beta^+ + \beta^-) \quad \text{subject to} \quad \begin{pmatrix} S & -S \\ -S & S \end{pmatrix} \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix} \leq \begin{pmatrix} \lambda_1 + \tilde{\delta} \\ \lambda_1 - \tilde{\delta} \end{pmatrix}, \quad \beta^+, \beta^- \geq 0,$$

(2.12)

with

$$A = \begin{pmatrix} S & -S \\ -S & S \end{pmatrix}, \quad b = \begin{pmatrix} \tilde{\delta} \\ -\tilde{\delta} \end{pmatrix}, \quad c = -1, \quad \bar{b} = 1, \quad \bar{c} = 0, \quad x = \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix}.$$

### 2.3 Sparse Precision Matrix Estimation

The third problem of interest is the sparse precision matrix estimation problem. In a typical graphical model setting, a $d$-dimensional random vector $x = (x^1, ..., x^d)^T$ can be represented by an undirected graph $G = (V, E)$, where graph node set $V$ represents the $d$ variables in $x$ and the edge set $E$ represents the conditional independence relationship among $x^1, ..., x^d$. We say that the graphical model is a Gaussian graphical model when the random vector $x$ is assumed to be Gaussian: $x \sim N_d(\mu^0, \Sigma^0)$. We call the inverse of the covariance matrix the precision matrix: $\Theta^0 = (\Sigma^0)^{-1}$.

Under the Gaussian assumption, the edge between $x^i$ and $x^j$ is excluded from $E$ if and only if $\Theta^0_{ij} = 0$ (Dempster, 1972). Let $\{x_1, ..., x_n\}$ be $n$ independent and identically distributed samples from the distribution of $x$. We are interested in estimating the support of the true precision matrix. We denote the sample covariance matrix $S = \frac{1}{n} \sum_{j=1}^{n} (x_j - \bar{x})(x_j - \bar{x})^T$, where $\bar{x} = \frac{1}{n} \sum_{j=1}^{n} x_j$. Most methods used to estimate the precision matrix are based on maximum-likelihood estimation. For example, see Banerjee et al. (2008); Friedman et al. (2007b,a, 2010) for the details of these MLE methods. Other methods include neighborhood selection (Meinshausen and Bühlmann, 2006) and linear programming approaches. A first linear programming approach is introduced in Yuan (2011). The estimation is decomposed into $d$ subproblems, and each column of $\Omega^0$ is estimated separately. For the $i$-th subproblem, we first solve the following convex optimization problem:

$$\min_{\beta} ||\beta||_1 \quad \text{subject to:} \quad ||S_{\setminus i, i}\beta - S_{\setminus i, i}||_\infty \leq \lambda \quad \text{and} \quad \beta \in \mathbb{R}^{d-1}. \quad (2.13)$$

Let $\tilde{\theta}_i$ denote the optimizer to (2.13), then we set the estimator to the $i$-th column of the precision matrix as the following: $\Omega^i_{ii} = (S_{\setminus i, i} - 2\tilde{\theta}_i S_{\setminus i, i} + \tilde{\theta}_i^T S_{\setminus i, i} \tilde{\theta}_i)^{-1}$ and $\Omega^i_{\setminus i, i} = -\Omega^i_{i, i} \tilde{\theta}_i$. This process is repeated for $i = 1, \cdots, d$.

Another similar linear programming approach used to estimate sparse precision matrix, named Constrained $\ell_1$ minimization Estimation (CLIME) (Cai et al., 2011), also decomposes the estimation steps column-wise. It initially solves the following optimization problem:

$$\min_{\Omega} ||\Omega||_1 \quad \text{subject to:} \quad ||S\Omega - I_d||_\infty \leq \lambda \quad \text{and} \quad \Omega \in \mathbb{R}^{d \times d}, \quad (2.14)$$

where $S$ is the sample covariance matrix. We further decompose (2.14) into $d$ smaller problems:

For the $i$-th subproblem, we recover the $i$-th column of $\Omega$, denoted as $\beta$, by solving

$$\min_{\beta} ||\beta||_1 \quad \text{subject to:} \quad ||S\beta - e_i||_\infty \leq \lambda \quad \text{and} \quad \beta \in \mathbb{R}^d. \quad (2.15)$$

6
Similar to the previous two models, equation (2.13) and (2.15) and can be both casted into the parametric linear programming form. Take (2.15) as an example:

$$\min_{\beta^+, \beta^-} 1^T(\beta^+ + \beta^-) \text{ subject to } \begin{pmatrix} S & -S \\ -S & S \end{pmatrix} \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix} \leq \begin{pmatrix} \lambda 1 + e_i \\ \lambda 1 - e_i \end{pmatrix}, \quad \beta^+, \beta^- \geq 0, \quad (2.16)$$

with

$$A = \begin{pmatrix} S & -S \\ -S & S \end{pmatrix}, \quad b = \begin{pmatrix} e_i \\ -e_i \end{pmatrix}, \quad c = -1, \quad \bar{b} = 1, \quad \bar{c} = 0, \quad x = \begin{pmatrix} \beta^+ \\ \beta^- \end{pmatrix}. \quad (2.17)$$

A software package that solves CLIME using the parametric simplex method has already been developed and documented (Pang et al., 2014). We are interested in extending the application of this package to other problems such as those described above.

### 2.4 $\ell_1$-norm SVM

In the standard two-class classification problem, we are given a sequence of training data set $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i \in \mathbb{R}^d$ is the input and $y_i \in \{1, -1\}$ is the binary output. Each entries of a data point $x_i$ is called a feature. The Support Vector Machine (SVM) is a powerful supervised learning method introduced in Cortes and Vapnik (1995) used for classification problems. It finds the hyperplane that maximizes the distance to the nearest training data point in a certain space so that the the generalization error of the classifier is minimized. By changing the kernel functions and mapping the data points to different spaces, this method is able to classify data in a high-dimensional feature space. The exact solution to the linear space $l_2$-norm SVM can be found by Quadratic Programming. We are interested in the linear space $\ell_1$-norm SVM (Zhu et al., 2004), which usually results in a better classification when there is redundant noise in features. The constrained $\ell_1$-norm SVM solves the following problem:

$$\min_{\beta_0, \beta} \sum_{i=1}^n [1 - y_i (\beta_0 + \beta^T x_i)]_+ \text{ subject to: } \|\beta\|_1 \leq \lambda, \quad (2.17)$$

where $\beta_0 \in \mathbb{R}$, $\beta \in \mathbb{R}^d$. When new data $x_k$ comes, we use the sign of $\beta_0 + \beta^T x_k$ to classify it. Let $z_i = 1 - y_i (\beta_0 + \beta^T x_i)$, for $i = 1, \ldots, n$. $z_i$ can be expressed as $z_i = z_i^+ - z_i^-$. Notice $[1 - y_i (\beta_0 + \beta^T x_i)]_+$ can be represented as $z_i^+$. We split $\beta$ and $\beta_0$ into positive and negative parts as well: $\beta = \beta^+ - \beta^-$ and $\beta_0 = \beta_0^+ + \beta_0^-$ and add slack variable $w$ to the constraint so that the constraint becomes equality: $\beta^+ + \beta^- + w = \lambda 1$, $w \geq 0$. Now we are ready to cast the problem into the equality parametric simplex form (1.1). We identify each component of (1.1) as the following:

$$x = \begin{pmatrix} z^+ & z^- & \beta^+ & \beta^- & \beta_0^+ & \beta_0^- & w \end{pmatrix}^T \in \mathbb{R}^{(n+1) \times (2n+3d+2)}, \quad x \geq 0,$$

$$c = \begin{pmatrix} -1^T & 0^T & 0^T & 0^T & 0^T \end{pmatrix}^T \in \mathbb{R}^{2n+3d+2}, \quad \bar{c} = 0 \in \mathbb{R}^{2n+3d+2},$$

$$b = \begin{pmatrix} 1^T \\ 0^T \end{pmatrix}^T \in \mathbb{R}^{n+1}, \quad \bar{b} = \begin{pmatrix} 0^T \\ 1 \end{pmatrix}^T \in \mathbb{R}^{n+1},$$

$$\bar{b} = \begin{pmatrix} 0^T \\ 1 \end{pmatrix}^T \in \mathbb{R}^{n+1},$$

$$\bar{b} = \begin{pmatrix} 0^T \\ 1 \end{pmatrix}^T \in \mathbb{R}^{n+1},$$
and finally

\[ A = \begin{pmatrix} I_n & -I_n & Z & -Z & y & -y & 1^T \\ 1^T & 1^T & 1^T & 1^T \end{pmatrix} \in \mathbb{R}^{(n+1)\times(2n+3d+2)}, \]

where

\[ Z = \begin{pmatrix} y_1^T \\ \vdots \\ y_n^T \end{pmatrix} \in \mathbb{R}^{n\times d}. \]

2.5 Differential Network and its Sparse Formulation

We consider the problem of directly estimation of differential networks in high dimensional graphical models. Many recent different applications of differential networking analysis can be founded in biological literature (Hudson et al., 2009; Bandyopadhyaya et al., 2010; Ideker and Krogan, 2012). Lauritzen (1996) models the changes in graph structure between two different conditions as a differential network which encodes the conditional dependency information. Sometimes, it is more interesting to analysis the magnitude difference of two graphs when they have the same structure (Zhao et al., 2013; Danaher et al., 2013). Let \( x = (x_1, \ldots, x_d)^T \) and \( y = (y_1, \ldots, y_d)^T \) be two different \( d \)-dimensional random vectors, where \( x \sim N_d(\mu_0, \Sigma_0^x) \) and \( y \sim N_d(\mu_0, \Sigma_0^y) \). Given \( \{x_1, \ldots, x_{n_1}\} \) be \( n_1 \) independent and identically distributed samples from the distribution of \( x \), and similarly, \( \{y_1, \ldots, y_{n_2}\} \) be \( n_2 \) independent and identically distributed samples from the distribution of \( y \), we are interested in estimating the difference of the precision matrices: \( \Delta^0 = (\Sigma_0^x)^{-1} - (\Sigma_0^y)^{-1} \). We defined the sample covariance matrix \( S_x = \frac{1}{n_1} \sum_{j=1}^{n_1} (x_j - \bar{x})(x_j - \bar{x})^T \), where \( \bar{x} = \frac{1}{n_1} \sum_{j=1}^{n_1} x_j \). \( S_y \) is defined similarly.

The most natural and straightforward way is to estimate the precision matrices separately. All recent sparse precision matrix estimation methods we referred in the CLIME subsection can be used in this case. Another approach is to estimate the jointly estimate the precision matrices by using the assumption that they have similar features (Chiquet et al., 2011; Guo et al., 2011; Danaher et al., 2013). Zhao et al. (2013) propose that the difference matrix \( \Delta^0 \) can be directly estimated by an approach similar to CLIME. Compared with separate estimation of each matrices, this direct estimation does not require both precision matrices to be sparse; instead, only relative sparsity is important in this approach. This advantage allows each network to contain hub nodes structure. This is very important in reality since normally a differential network is considered as changes in the graph structure which should be relatively sparse. The estimator proposed in Zhao et al. (2013), solves the following problem:

\[
\min_{\Delta} \|\Delta\|_1 \quad \text{subject to:} \quad \|S_x \Delta S_y - S_x + S_y\|_\infty \leq \lambda, \quad \text{and} \quad \Delta \in \mathbb{R}^{d\times d},
\]

where \( S_x \) and \( S_y \) are the sample covariance matrices.

Now we explain how to formulate the problem described above as a parametric linear programming problem. In general, we are interested in solving:

\[
\min_{D} \|D\|_1 \quad \text{subject to:} \quad \|XDZ - Y\|_\infty \leq \lambda, \quad D \in \mathbb{R}^{n_1\times n_2},
\]
where $X \in \mathbb{R}^{m_1 \times n_1}$, $Z \in \mathbb{R}^{n_2 \times m_2}$ and $Y \in \mathbb{R}^{m_1 \times m_2}$ are given data matrix.

Instead of directly writing down the problem in terms of variable matrix $D$, we factor the matrix $D$ into a product of sparse matrices. The idea is simple—sparsity matters. A larger sparsely formulated problem can often be solved much faster than an equivalent smaller dense formulation. Although the number of the unknown variables increases, we have a more sparse constraint matrix and this increases the speed of the algorithm significantly. First let $XD = C$ and we write the matrix $D$ as $D = D^+ - D^-$. The components of the matrices $D^+$ and $D^-$ are all nonnegative and represent the positive and negative parts of matrix $D$ at optimality. Now form (2.19) becomes

$$\min_{D^+, D^-} \mathbf{1}^T (D^+ - D^-) \mathbf{1} \quad \text{subject to: } \|CZ - Y\|_\infty \leq \lambda, \quad X(D^+ - D^-) = C, \quad D^+, D^- \geq 0. \quad (2.20)$$

We then decompose the matrices $D^+$, $D^-$, $C$ and $Y$ column by column. Let $\text{vec}(D^+)$, $\text{vec}(D^-)$, $\text{vec}(C)$ and $\text{vec}(Y)$ be the vectors obtained by stacking the columns of matrices $D^+$, $D^-$, $C$ and $Y$, respectively. We write down (2.20) in the equality parametric linear programming form by adding positive slack variables to the inequality constraint:

$$\max_x c^T x \quad \text{subject to: } Ax = \bar{b} \lambda + b. \quad (2.21)$$

After some careful calculation, it is straightforward to check matrix $A$ in (2.21) has the following form:

$$A = \begin{pmatrix} X^0 & -X^0 & -I_{m_1 n_2} \\
Z^0 & I_{m_1 m_2} & -Z^0 \\
-I_{m_1 m_2} & -Z^0 & I_{m_1 m_2} \end{pmatrix}$$

with

$$X^0 = \begin{pmatrix} X \\
\vdots \\
X \end{pmatrix} \in \mathbb{R}^{m_1 n_2 \times n_1 n_2}, \quad Z^0 = \begin{pmatrix} z_{11} I_{m_1} & \cdots & z_{n_2 1} I_{m_1} \\
\vdots & \ddots & \vdots \\
-z_{1 n_2} I_{m_1} & \cdots & -z_{n_2 n_2} I_{m_1} \end{pmatrix} \in \mathbb{R}^{m_1 m_2 \times n_1 n_2},$$

where $z_{ij}$ denotes the $(i, j)$ entry of matrix $Z$;

$$x = \begin{bmatrix} \text{vec}(D^+) & \text{vec}(D^-) & \text{vec}(C) & w \end{bmatrix}^T \in \mathbb{R}^{2n_1 n_2 + m_1 n_2 + 2m_1 m_2}, \quad \text{vec}(D^+), \text{vec}(D^-), w \geq 0,$n

where $w \in \mathbb{R}^{2m_1 m_2}$ is nonnegative slack variable vector used to make the inequality become an equality. Note that the vec$(C)$ component does not have the nonnegative constraint but we can always declare it to be the basic variables. We identify the vectors $x$, $b$, $c$ and $\bar{b}$ in (2.21) as the following:

$$b = \begin{bmatrix} 0^T & \text{vec}(Y) & -\text{vec}(Y) \end{bmatrix}^T \in \mathbb{R}^{m_1 n_2 + 2m_1 m_2},$$

where the first $m_1 n_2$ components of vector $b$ are 0 and the rest components are from matrix $Y$;

$$\bar{b} = \begin{bmatrix} 0^T & 1^T & 1^T \end{bmatrix}^T \in \mathbb{R}^{m_1 n_2 + 2m_1 m_2},$$
where the first $m_1n_2$ components of vector $\bar{b}$ are 0 and the rest $2m_1m_2$ components are 1;

$$c = \begin{pmatrix} -1^T & -1^T & 0^T & 0^T \end{pmatrix}^T \in \mathbb{R}^{2n_1n_2 + m_1n_2 + 2m_1m_2},$$

where the first $2n_1n_2$ components of vector $c$ are $-1$ and the rest $m_1n_2 + 2m_1m_2$ components are 0.

2.6 The Primal Simplex Method

We now review the primal simplex method for solving the standard linear programming problem:

$$\max \ x^Tc \quad \text{subject to: } Ax = b, \quad x \geq 0, \quad x \in \mathbb{R}^n,$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ are given. Without loss of generality, we assume that $m \leq n$ and matrix $A$ has full row rank $m$. Throughout our analysis, we assume that an optimal solution exists (it need not be unique). The primal simplex method starts from a basic feasible solution (to be defined shortly—but geometrically can be thought of as any vertex of the feasible polytope) and proceeds step-by-step (vertex-by-vertex) to the optimal solution. Various techniques exist to find the first feasible solution. In many variants of the simplex method these techniques involve a so-called Phase I method. See Vanderbei (1995); Murty (1983); Dantzig (1951).

Algebraically, a basic solution corresponds to a partition of the indices $\{1, \ldots, n\}$ into $m$ basic indices denoted $B$ and $n-m$ non-basic indices denoted $N$. Note that not all partitions are allowed—the submatrix of $A$ consisting of the columns of $A$ associated with the basic indices, denoted $A_B$, must be invertible. The submatrix of $A$ corresponding to the nonbasic indices is denoted $A_N$. Suppressing the fact that the columns have been rearranged, we can write

$$A = \begin{bmatrix} A_N & A_B \end{bmatrix}.$$

If we rearrange the rows of $x$ and $c$ in the same way, we can introduce a corresponding partition of these vectors:

$$x = \begin{bmatrix} x_N \\ x_B \end{bmatrix}, \quad c = \begin{bmatrix} c_N \\ c_B \end{bmatrix}.$$

From the commutative property of addition, it is easy to see that $Ax = b$ can be written as

$$\begin{bmatrix} A_N & A_B \end{bmatrix} \begin{bmatrix} x_N \\ x_B \end{bmatrix} = A_Nx_N + A_Bx_B = b.$$

Since the matrix $A_B$ is assumed to be invertible, we can express $x_B$ in terms of $x_N$ as follows:

$$x_B = A_B^{-1}b - A_B^{-1}A_Nx_N = x_B^* - A_B^{-1}A_Nx_N^*,$$

where we have written $x_B^*$ as an abbreviation for $A_B^{-1}b$. This rearrangement of the equality constraints is called a dictionary because the basic variables are defined as functions of the nonbasic variables.
Denoting the objective $c^T x$ by $\zeta$, then we also can write:

$$\zeta = c^T x = c_B^T x_B + c_N^T x_N$$

$$= c_B^T (A_B^{-1} b - A_B^{-1} A_N x_N) + c_N^T x_N$$

$$= c_B^T A_B^{-1} b - ((A_B^{-1} A_N)^T c_B - c_N)^T x_N$$

$$= \zeta^* - (z_N^*)^T x_N,$$

where $\zeta^* = c_B^T A_B^{-1} b$, $x_B^* = A_B^{-1} b$ and $z_N^* = (A_B^{-1} A_N)^T c_B - c_N$.

We call equations (2.23) and (2.24) the **primal dictionary** associated with the current basis $B$. Corresponding to each dictionary, there is a **basic solution** (also called a dictionary solution) obtained by setting the nonbasic variables to zero and reading off values of the basic variables:

$$x_N = 0, \quad x_B = x_B^*.$$  

This particular “solution” satisfies the equality constraints of the problem by construction. To be a feasible solution one only needs to check that the values of the basic variables are nonnegative. So, we say that a basic solution is a **basic feasible solution** if $x_B^* \geq 0$.

An iteration of the simplex method, also called a **pivot**, is accomplished by swapping one basic variable with one nonbasic variable updating the dictionary appropriately. Geometrically, a pivot corresponds to jumping from one vertex to an adjacent vertex in the polytope of feasible solutions. The pair of indices that are swapped are chosen so that the objective function increases (since we are maximizing) and the dictionary solution remains feasible. These conditions guide but do not uniquely determine the indices to swap. One can employ various selection methods. The particular method chosen is called a **pivot rule**. Different pivot rules correspond to different variants of the simplex method.

The dual of (2.22) is given by

$$\max_y -b^T y \text{ subject to: } A^T y - z = c, \quad z \geq 0 \quad z \in \mathbb{R}^n, y \in \mathbb{R}^m. \quad (2.25)$$

In this case, we separate variable $z$ into basic and nonbasic parts as before:

$$[z] = [z_N, z_B].$$

Now (2.25) becomes

$$A_N^T y - z_N = c_N \quad (2.26)$$

and

$$A_B^T y - z_B = c_B \quad (2.27)$$

We can use (2.27) to solve for the “free” (i.e., not constrained to be nonnegative) variable $y$:

$$y = (A_B^T)^{-1}(c_B + z_B).$$
We plug this back to (2.26) and the dual objective. The corresponding dual dictionary is given by:

\[ z_N = (A_B^{-1}A_N)^T c_B - c_N + (A_B^{-1}A_N)^T z_B = z_N^* + (A_B^{-1}A_N)^T z_B, \]  

(2.28)

\[-\xi = -c_B^T A_B^{-1} b + (A_B^{-1} b)^T z_B = -\zeta^* - (x_B^*)^T z_B, \]  

(2.29)

where \( \xi \) denotes the objective function in the (2.25), \( \zeta^* = c_B^T A_B^{-1} b, x_B^* = A_B^{-1} b \) and \( z_N^* = (A_B^{-1}A_N)^T c_B - c_N. \)

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 (2.23) and (2.28). 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.

2.7 Initialization of the Simplex Method

It is important to declare the initial basic and nonbasic variables correctly. For the inequality form (1.2), it is very natural to add slack variables \( w \) to make the constraint be an equality and the initial slack variables \( w \) can be chosen as the basic variables since an identity matrix is obviously invertible. If the problem is in equality form (1.1), we need to manually select the basic variables so that the matrix \( A_B \) is invertible. For the LAD-Lasso problem in (2.6), it is very natural to select \( t^- \) as the basic variables. For the differential network problem, we are interested in solving (2.21). We explain how to select the basic variables in this case.

First, recall that the nonbasic variables are used to represent the basic variables. As a result, the coefficient of the matrix associated with the basic variables should be sparse so it can be “easily” inverted. In fact, we choose the first \( 2n_1n_2 \) variables from \( \text{vec}(D^+) \) and \( \text{vec}(D^-) \) to be nonbasic variables and the rest \( m_1n_2 + 2m_1m_2 \) variables from \( \text{vec}(C) \) to be basic variables. The slack variables added at the end in order to make the constraints be equality, of course, are basic variables. Second, the variables \( \text{vec}(D^+), \text{vec}(D^-) \) and slack variables \( w \) are nonnegative, but \( \text{vec}(C) \) are free variables. While we are implementing the parametric simplex method, we simply declare that the variables \( \text{vec}(C) \) are always basic. That means they never swap with other variables and this trick improves the speed of the algorithm significantly. Despite the fact that this matrix has large dimension, this formulation can still be solved efficiently because the constraint matrix and the vectors are quite sparse.

3 The Parametric Simplex Method

We introduce the parametric simplex method used to find the full solution path while solving the parametric linear programming problem only once. Different variant of the simplex method is defined by the rule for choosing the pair of variables to swap at each iteration. Here we describe the
rule used by the parametric simplex method: we add some positive perturbations ($\bar{b}$ and $\bar{c}$) times a positive parameter $\lambda$ to both objective function and the right hand side of the primal problem. The purpose of doing this is to guarantee the primal and dual feasibility when $\lambda$ is large. Since the problem is already primal feasible and dual feasible, there is no phase I stage required for the parametric simplex method. Furthermore, if the $i$-th entry of $b$ or the $j$-th entry of $c$ has already satisfied the feasibility condition ($b_i \geq 0$ or $c_j \leq 0$), then the corresponding perturbation $\bar{b}_i$ or $\bar{c}_j$ to that entry is allowed to be 0. With these perturbations, (2.22) becomes:

$$\max (c + \lambda \bar{c})^T x \quad \text{subject to: } A x = b + \lambda \bar{b}, \quad x \geq 0 \quad x \in \mathbb{R}^n. \quad (3.1)$$

We separate the perturbation vectors into basic and nonbasic parts as well and write down the dictionary with perturbations corresponding to (2.23),(2.24),(2.28) and (2.29) as:

$$x_B = (x_B^* + \lambda x_B) - A_B^{-1} A_N x_N, \quad (3.2)$$

$$\zeta = \zeta^* - (z_N^* + \lambda \bar{z}_N)^T x_N, \quad (3.3)$$

$$z_N = (z_N^* + \lambda \bar{z}_N) + (A_B^{-1} A_N)^T z_B, \quad (3.4)$$

$$\xi = -\zeta^* - (x_B^* + \lambda \bar{x}_B)^T z_B, \quad (3.5)$$

where $x_B^* = A_B^{-1} b$, $z_N^* = (A_B^{-1} A_N)^T c_B - c_N$, $x_B = A_B^{-1} \bar{b}$ and $\bar{z}_N = (A_B^{-1} A_N)^T \bar{c}_B - \bar{c}_N$.

When $\lambda$ is large, the dictionary will be both primal and dual feasible ($x_B^* + \lambda \bar{x}_B \geq 0$ and $z_N^* + \lambda \bar{z}_N \geq 0$). The corresponding primal solution is simple: $x_B = x_B^* + \lambda \bar{x}_B$ and $x_N = 0$. This solution is valid until $\lambda$ hits a lower bound which breaks the feasibility. The smallest value of $\lambda$ without break any feasibility is given by

$$\lambda^* = \min \{ \lambda : z_N^* + \lambda \bar{z}_N \geq 0 \text{ and } x_B^* + \lambda \bar{x}_B \geq 0 \}. \quad (3.6)$$

In other words, the dictionary and its corresponding solution $x_B = x_B^* + \lambda \bar{x}_B$ and $x_N = 0$ is optimal for the value of $\lambda \in [\lambda^*, \lambda_{\max}]$, where

$$\lambda^* = \max \left( \max_{j \in N, \bar{z}_{N_j} > 0} \frac{z_{N_j}^*}{\bar{z}_{N_j}}, \max_{i \in B, \bar{x}_{B_i} > 0} \frac{x_{B_i}^*}{\bar{x}_{B_i}} \right), \quad (3.7)$$

$$\lambda_{\max} = \min \left( \min_{j \in N, \bar{z}_{N_j} < 0} \frac{z_{N_j}^*}{\bar{z}_{N_j}}, \min_{i \in B, \bar{x}_{B_i} < 0} \frac{x_{B_i}^*}{\bar{x}_{B_i}} \right). \quad (3.8)$$

Note that although initially the perturbations are nonnegative, as the dictionary gets updated, the perturbation does not necessarily maintain nonnegativity. For each dictionary, there is a corresponding interval of $\lambda$ given by (3.7) and (3.8). We have characterized the optimal solution for this interval, and these together give us the solution path of the original parametric linear programming problem. Next, we show how the dictionary gets updated as the leaving variable and entering variable swap.

We expect that after swapping the entering variable $j$ and leaving variable $i$, the new solution in the dictionary (3.2) and (3.4) would slightly change to:

$$x_j^* = t, \quad x_j^* = \bar{t},$$
\( z_i^* = s, \quad \bar{z}_i^* = \bar{s}, \)

\[
\begin{align*}
\mathbf{x}_B^* &\leftarrow \mathbf{x}_B^* - t \mathbf{\Delta x}_B, \quad \mathbf{x}_B &\leftarrow \mathbf{x}_B - \bar{t} \mathbf{\Delta x}_B, \\
\mathbf{z}_N^* &\leftarrow \mathbf{z}_N^* - s \mathbf{\Delta z}_N, \quad \bar{\mathbf{z}}_N &\leftarrow \bar{\mathbf{z}}_N - \bar{s} \mathbf{\Delta z}_N,
\end{align*}
\]

where \( t \) and \( \bar{t} \) are the primal step length for the primal basic variables and perturbations, \( s \) and \( \bar{s} \) are the dual step length for the dual nonbasic variables and perturbations, \( \mathbf{\Delta x}_B \) and \( \mathbf{\Delta z}_N \) are the primal and dual step directions, respectively. We explain how to find these values in details now.

There is either a \( j \in \mathcal{N} \) for which \( \mathbf{z}_N^* + \lambda \bar{\mathbf{z}}_N = 0 \) or an \( i \in \mathcal{B} \) for which \( \mathbf{x}_B^* + \lambda \bar{\mathbf{x}}_B = 0 \) in (3.6). If it corresponds to a nonbasic index \( j \), then we do one step of the primal simplex. In this case, we declare \( j \) as the entering variable, then we need to find the primal step direction \( \mathbf{\Delta x}_B \). After the entering variable \( j \) has been selected, \( \mathbf{x}_N \) changes from 0 to \( t e_j \), where \( t \) is the primal step length, then according to (3.2), we have that \( \mathbf{x}_B = (\mathbf{x}_B^* + \lambda \bar{\mathbf{x}}_B) - \mathbf{A}^{-1}_B \mathbf{A}_N \mathbf{te}_j \). The step direction \( \mathbf{\Delta x}_B \) is given by \( \mathbf{\Delta x}_B = \mathbf{A}^{-1}_B \mathbf{A}_N \mathbf{e}_j \). We select the leaving variable next. In order to maintain primal feasibility, we need to keep \( \mathbf{x}_B \geq 0 \), therefore, the leaving variable \( i \) is selected such that \( i \in \mathcal{B} \) achieves the maximal value of \( \mathbf{\Delta x}_i^* \mathbf{x}_i^* + \lambda \bar{\mathbf{z}}_i^* \mathbf{z}_i^* \). It only remains to show how \( \mathbf{z}_N \) changes. Since \( i \) is the leaving variable, according to (3.4), we have \( \mathbf{\Delta z}_N = - (\mathbf{A}^{-1}_B \mathbf{A}_N)^T \mathbf{e}_i \). After we know the entering variables, the primal and dual step directions, the primal and dual step lengths can be found as

\[
\begin{align*}
t & = \frac{x_i^*}{\Delta x_i}, \quad \bar{t} = \frac{\bar{x}_i}{\Delta x_i}, \\
s & = \frac{z_j^*}{\Delta z_j}, \quad \bar{s} = \frac{\bar{z}_j}{\Delta z_j}.
\end{align*}
\]

If, on the other hand, the constraint in (3.6) corresponds to a basic index \( i \), we declare \( i \) as the leaving variable, then similar calculation can be made based on the dual simplex method (apply the primal simplex method to the dual problem). Since it is very similar to the primal simplex method, we omit the detailed description.

The algorithm will terminate whenever \( \lambda^* \leq 0 \). The corresponding solution is optimal since our dictionary always satisfies primal feasibility, dual feasibility and complementary slackness condition. The only concern during the entire process of the parametric simplex method is that \( \lambda \) does not equal to zero, so as long as \( \lambda \) can be set to be zero, we have the optimal solution to the original problem. We summarize the parametric simplex method for a given parametric linear programming problem in Algorithm 1:

The following theorem shows that the updated basic and nonbasic partition still gives the optimal solution.

**Theorem 3.1.** For a given dictionary with parameter \( \lambda \) in the form of (3.2), (3.3), (3.4) and (3.5), let \( \mathcal{B} \) be a basic index set and \( \mathcal{N} \) be an nonbasic index set. Assume this dictionary is optimal for \( \lambda \in [\lambda^*, \lambda_{\text{max}}] \), where \( \lambda^* \) and \( \lambda_{\text{max}} \) are given by (3.7) and (3.8), respectively. The updated dictionary with basic index set \( \mathcal{B}^* \) and nonbasic index set \( \mathcal{N}^* \) given by the parametric simplex method is still optimal at \( \lambda = \lambda^* \).
Write down the dictionary as in (3.2),(3.3),(3.4) and (3.5);
Find $\lambda^*$ given by (3.7);
while $\lambda^* > 0$ do
  if the constraint in (3.7) corresponds to an index $j \in \mathcal{N}$ then
    Declare $x_j$ as the entering variable;
    Compute primal step direction. $\Delta x_B = A_B^{-1} A_N e_j$;
    Select leaving variable. Need to find $i \in \mathcal{B}$ that achieves the maximal value of $\frac{\Delta x_i}{x_i^* + \lambda^* x_i}$;
    Compute dual step direction. It is given by $\Delta z_N = -(A_B^{-1} A_N)^T e_i$;
  else if the constraint in (3.7) corresponds to an index $i \in \mathcal{B}$ then
    Declare $z_i$ as the leaving variable;
    Compute dual step direction. $\Delta z_N = -(A_B^{-1} A_N)^T e_i$;
    Select entering variable. Need to find $j \in \mathcal{N}$ that achieves the maximal value of $\Delta z_j$;
    Compute primal step direction. It is given by $\Delta x_B = A_B^{-1} A_N e_j$;
  Compute the dual and primal step lengths for both variables and perturbations:
  $$t = \frac{x_i^*}{\Delta x_i}, \quad \bar{t} = \frac{x_i}{\Delta x_i}, \quad s = \frac{z_j^*}{\Delta z_j}, \quad \bar{s} = \frac{\bar{z}_j}{\Delta \bar{z}_j}.$$  
  Update the primal and dual solutions:
  $$x_j^* = t, \quad \bar{x}_j = \bar{t}, \quad z_i^* = s, \quad \bar{z}_i = \bar{s},$$
  $$x_B^* \leftarrow x_B - t \Delta x_B, \quad \bar{x}_B \leftarrow \bar{x}_B - \bar{t} \Delta x_B, \quad z_N^* \leftarrow z_N^* - s \Delta z_N, \quad \bar{z}_N \leftarrow \bar{z}_N - \bar{s} \Delta z_N.$$
  Update the basic and nonbasic index sets $\mathcal{B} := \mathcal{B} \setminus \{i\} \cap \{j\}$ and $\mathcal{N} := \mathcal{N} \setminus \{j\} \cap \{i\}$.
  Write down the new dictionary and compute $\lambda^*$ given by (3.7);
end
Set the nonbasic variables as 0s and read the values of the basic variables.

**Algorithm 1**: The parametric simplex method
During each iteration, there is an optimal solution corresponding to $\lambda \in [\lambda^*, \lambda_{\text{max}}]$. Notice each of these $\lambda$ ranges is determined by a partition between basic and nonbasic variables, and the number of the partition into basic and nonbasic variables is finite, so after finite steps, we must find the optimal solution corresponding to all $\lambda$ values.

4 Numerical Experiments

In this section, we present some numerical experiments and give some insights about how the parametric simplex method solves different linear programming problems. We verify the following assertions: (1) The parametric simplex method requires very few iterations to identify the nonzero component if the original problem is sparse. (2) The parametric simplex method is able to find the full solution path by solving the problem only once. It is fast, efficient and precise. (3) The parametric simplex method maintains the feasibility of the problem up to machine precision along the solution path.

4.1 Solution path of Dantzig selector

We start with a simple example that illustrates how the recovered solution path of the Dantzig selector model changes as the parametric simplex method iterates. We adopt the example used in Candès and Tao (2007). The design matrix $X$ has $n = 100$ rows and $d = 250$ columns. The entries of $X$ are generated from an array of independent Gaussian random variables that are then
normalized so that each column has a given norm. We randomly select $s = 8$ entries from the response vector $\beta^0$, and set them as $\beta^0_i = s_i(1 + a_i)$, where $s_i = 1$ or $-1$, with probability $1/2$ and $a_i \sim \mathcal{N}(0,1)$. The other entries of $\beta^0$ are set to zero. We form $y = X\beta^0 + \epsilon$, where $\epsilon_i \sim \mathcal{N}(0,\sigma)$, with $\sigma = 1$. We stop the parametric simplex method when $\lambda \leq \sigma \sqrt{n \log d/n}$. The solution path of the result is shown in Figure 1(a). We see that our method correctly identifies all nonzero entries of $\beta$ in less than 10 iterations. Some small overestimations occur in a few iterations after all nonzero entries have been identified. We also show how the parameter $\lambda$ evolves as the parametric simplex method iterates in Figure 1(b). As we see, $\lambda$ decreases sharply to less than 5 after all nonzero components have been identified. This reconciles with the theorem we developed. The algorithm itself only requires a very small number of iterations to correctly identify the nonzero entries of $\beta$.

In our example, each iteration in the parametric simplex method identifies one or two non-sparse entries in $\beta$.

### 4.2 Feasibility of Dantzig Selector

Another advantage of the parametric simplex method is that the solution is always feasible along the path while other estimating methods usually generate infeasible solutions along the path. We compare our algorithm with the R package “flare” (Li et al., 2015) which uses the Alternating Direction Method of Multipliers (ADMM) using the same example described above. We compute the values of $\|X^T X \beta^i - X^T y\|_\infty - \lambda_i$ along the solution path, where $\beta^i$ is the $i$-th basic solution (with corresponding $\lambda_i$) obtained while the parametric simplex method is iterating. Without any doubts, we always obtain 0s during each iteration. We plug the same list of $\lambda_i$ into “flare” and compute the solution path for this list as well. As shown in Table 1, the parametric simplex method is always feasible along the path since it is solving each iteration up to machine precision; while the solution path of the ADMM is almost always breaking the feasibility by a large amount, especially in the first few iterations which correspond to large $\lambda$ values. Each experiment is repeated for 100 times.

<table>
<thead>
<tr>
<th></th>
<th>Maximum violation</th>
<th>Minimum Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMM</td>
<td>498(122)</td>
<td>143(73.2)</td>
</tr>
<tr>
<td>PSM</td>
<td>0(0)</td>
<td>0(0)</td>
</tr>
</tbody>
</table>

### 4.3 Performance Benchmark of Dantzig Selector

In this part, we compare the timing performance of our algorithm with R package “flare”. We fix the sample size $n$ to be 200 and vary the data dimension $d$ from 100 to 5000. Again, each entries of $X$ is independent Gaussian and normalized so that the column has uniform norm. We randomly select 2% entries from vector $\beta$ to be nonzero and each entry is chosen as $\sim \mathcal{N}(0,1)$. We compute $y = X\beta + \epsilon$, with $\epsilon_i \sim \mathcal{N}(0,1)$ and try to recover vector $\beta$, given $X$ and $y$. Our method stops when $\lambda$ is less than $2\sigma \sqrt{\log d/n}$, so that the full solution path for all the values of $\lambda$ up to this
value is computed by the parametric simplex method. In “flare”, we estimate $\beta$ when $\lambda$ equals that number in the Dantzig selector model. This means “flare” has much less computation task than the parametric simplex method. As we can see in Table 1, our method has a much better performance than “flare” in terms of speed. We compare and present the timing performance of the two algorithms in seconds and each experiment is repeated for 100 times. In practice, only very few iterations is required when the response vector $\beta$ is sparse.

Table 2: Average timing performance (in seconds) with standard errors in the parentheses on Dantzig selector

<table>
<thead>
<tr>
<th></th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flare</td>
<td>19.5(2.72)</td>
<td>44.4(2.54)</td>
<td>142(11.5)</td>
<td>1500(231)</td>
</tr>
<tr>
<td>PSM</td>
<td>2.40(0.220)</td>
<td>29.7(1.39)</td>
<td>47.5(2.27)</td>
<td>649(89.8)</td>
</tr>
</tbody>
</table>

4.4 Performance Benchmark of Differential Network

We now apply this optimization method to the Differential Network model. We need two covariance matrices and the difference of their inverse has to be sparse. We generate $\Sigma_0 = U^T \Lambda U$, where $\Lambda \in \mathbb{R}^{d \times d}$ is a diagonal matrix and its entries are i.i.d. and uniform on $[1, 2]$, and $U \in \mathbb{R}^{d \times d}$ is a random matrix with i.i.d. entries from $\mathcal{N}(0,1)$. Let $D_1 \in \mathbb{R}^{d \times d}$ be a random sparse symmetric matrix with a certain sparsity level. Each entry of $D_1$ is i.i.d. and from $\mathcal{N}(0,1)$. We set $D = D_1 + 2|\lambda_{\min}(D_1)|I_d$ in order to guarantee the positive definiteness of $D$, where $\lambda_{\min}(D_1)$ is the smallest eigenvalue of $D_1$. Finally, we let $\Omega_X^0 = (\Sigma_X^0)^{-1}$ and $\Omega_Y^0 = \Omega_X^0 + D$.

We then generate data of sample size $n = 100$. The corresponding sample covariance matrices $S_X$ and $S_Y$ are also computed based on the data. Unfortunately, we are not able to find other software which can efficiently solve this problem, so we only list the timing performance of our algorithm as dimension $d$ varies from 25 to 200 in Table 2. We stop our algorithm whenever the solution achieved the desired sparsity level. When $d = 25, 50$ and 100, the sparsity level of $D_1$ is set to be 0.02 and when $d = 150$ and 200, the sparsity level of $D_1$ is set to be 0.002. Each experiment is repeated by 100 times.

Table 3: Average timing performance (in seconds) and iteration numbers with standard errors in the parentheses on differential network

<table>
<thead>
<tr>
<th></th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing</td>
<td>0.0185(0.00689)</td>
<td>0.376(0.124)</td>
<td>6.81(2.38)</td>
<td>13.41(1.26)</td>
<td>46.88(7.24)</td>
</tr>
<tr>
<td>Iteration Number</td>
<td>15.5(7.00)</td>
<td>55.3(18.8)</td>
<td>164(58.2)</td>
<td>85.8(16.7)</td>
<td>140(26.2)</td>
</tr>
</tbody>
</table>
5 Conclusion

We present the parametric simplex method. It can effectively solve a series of sparse statistical learning problems which can be written as linear programming problems with a tuning parameter. It is shown that by using the parametric simplex method, one can solve the problem for all values of the parameter in the same time as another method would require to solve for just one instance of the parameter. Another good feature of this method is that it maintains the feasibility while the algorithm iterates, and this feature guarantees the precision of the obtained solution path.

6 Proof

In this section, we present the supplementary proofs.

6.1 Proof of Theorem 3.1

Proof. In order to prove that the new dictionary is still optimal, we only need to show that new dictionary is still primal and dual feasible: \( \mathbf{x}_B^* + \lambda^* \tilde{\mathbf{x}}_B \geq \mathbf{0} \) and \( \mathbf{z}_{N^*}^* + \lambda^* \tilde{\mathbf{z}}_{N^*} \geq \mathbf{0} \).

Case I. When calculating \( \lambda^* \) given by (3.7), if the constraint corresponds to an index \( i \in B \), then \( \mathbf{z}_{N^*}^* + \lambda^* \tilde{\mathbf{z}}_{N^*} \geq \mathbf{0} \) is guaranteed by the way of choosing entering variable. It remains to show the primal solution is not changed: \( \mathbf{x}_B^* + \lambda^* \tilde{\mathbf{x}}_B = \mathbf{x}_B^* + \lambda^* \tilde{\mathbf{x}}_B^* \).

We observe that \( \mathbf{A}_B \) is obtained by changing one column of \( \mathbf{A}_{B^*} \) to another column vector from \( \mathbf{A}_{N^*} \), and we assume the difference of these two vectors are \( \mathbf{u} \). Without loss of generality, we assume that the \( k \)-th column of \( \mathbf{A}_B \) is replaced, now we have \( \mathbf{A}_B = \mathbf{A}_B + \mathbf{ue}_k^T \). Sherman-Morrison formula says that

\[
\mathbf{A}_B \mathbf{A}_B^{-1} = \mathbf{I} - \frac{\mathbf{ue}_k^T \mathbf{A}_B^{-1}}{1 + \mathbf{e}_k^T \mathbf{A}_B^{-1} \mathbf{u}} = \mathbf{I} - \beta \mathbf{ue}_k^T \mathbf{A}_B^{-1},
\]

where \( \beta = \frac{1}{1 + \mathbf{e}_k^T \mathbf{A}_B^{-1} \mathbf{u}} \). Now consider the following term:

\[
\mathbf{A}_B \left[ \mathbf{x}_B^* - \mathbf{x}_{B^*} + \lambda^* (\tilde{\mathbf{x}}_B - \tilde{\mathbf{x}}_{B^*}) \right]
= \mathbf{A}_B (\mathbf{A}_B^{-1} - \mathbf{A}_{B^*}^{-1}) \mathbf{b} + \lambda^* \mathbf{A}_B (\mathbf{A}_B^{-1} - \mathbf{A}_{B^*}^{-1}) \tilde{\mathbf{b}}
= \mathbf{b} - \mathbf{A}_B \mathbf{A}_B^{-1} \mathbf{b} + \lambda^* (\tilde{\mathbf{b}} - \mathbf{A}_B \mathbf{A}_{B^*}^{-1} \mathbf{b})
= (\beta \mathbf{ue}_k^T \mathbf{A}_B^{-1}) \mathbf{b} + \lambda^* (\beta \mathbf{ue}_k^T \mathbf{A}_B^{-1}) \tilde{\mathbf{b}}
= (\beta \mathbf{ue}_k^T \mathbf{A}_B^{-1} \mathbf{b} + \lambda^* \mathbf{ue}_k^T \mathbf{A}_B^{-1} \mathbf{b}).
\]

Recall in this case, we have

\[
\lambda^* = \max_{i \in B, \tilde{x}_i > 0} \frac{x_i}{\tilde{x}_i} = -\frac{\mathbf{e}_k^T \mathbf{A}_B^{-1} \mathbf{b}}{\mathbf{e}_k^T \mathbf{A}_B^{-1} \tilde{\mathbf{b}}}. \tag{6.3}
\]

Substitute the definition of \( \lambda^* \) from (6.3) into (6.2), we notice that the expression in (6.2) is 0. Since \( \mathbf{A}_B \) is invertible, we have \( \mathbf{x}_B^* + \lambda^* \tilde{\mathbf{x}}_B = \mathbf{x}_{B^*}^* + \lambda^* \tilde{\mathbf{x}}_{B^*} \), and thus the new dictionary is still optimal at \( \lambda^* \).
Case II. When calculating $\lambda^*$ given by (3.7), if, on the other hand, the constraint corresponds to an index $j \in N$, then $x^*_B + \lambda^*x_B \geq 0$ is guaranteed by the way we choose leaving variable. It remains to show that it is still dual feasible.

Again, we observe that $A_B$ is obtained by changing one column of $A_B$ (say, $a_i$) to another column vector from $A_N$ (say, $a_j$), and we denote $u = a_j - a_i$ as the difference of these two vectors. Without loss of generality, we assume the replacement occurs at the $k$-th column of $A_B$. Sherman-Morrison formula gives

$$ A_B^{-1}A_B = I - \frac{A_B^{-1}ue_k^T}{1 + e_k^TA_B^{-1}u} = I - \frac{pe_k^T}{1 + e_k^Tp} = \begin{pmatrix} 1 & -\frac{p_1}{1+p_k} & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \ddots & \cdots \\ \vdots & \cdots & \ddots & \ddots & \cdots \\ \vdots & \cdots & \cdots & \ddots & \cdots \\ -\frac{pm}{1+p_k} & \cdots & \cdots & \cdots & 1 \end{pmatrix}, \tag{6.4} $$

where $p = A_B^{-1}u$, and $p_l$ denotes the $l$-th entry of $p$. Observe that in (6.4), only the $k$-th column is different from the identity matrix.

Dual feasible requires that $z^*_N = (A_B^{-1}A_N)^Tc_B - c_N \geq 0$. Since $(A_B^{-1}A_B)^Tc_B - c_B = 0$, we slightly change the dual feasible condition to: $(A_B^{-1}A)^Tc_B - c \geq 0$. In the parametric linear programming sense, $c \leftarrow c + \lambda c$ and $c_B \leftarrow c_B + \lambda c_B$. We only need to show that $(A_B^{-1}A)^T(c_B + \lambda^*c) - (c + \lambda^*c) = (A_B^{-1}A)^T(c_B + \lambda^*c_B) - (c + \lambda^*c)$. Consider the following term:

$$ (A_B^{-1}A)^T(c_B) - c + \lambda^*[(A_B^{-1}A)^Tc_B - c] - \{(A_B^{-1}A)^Tc_B - c + \lambda^*[(A_B^{-1}A)^Tc_B - c] \}
$$

$$ = A^T(A_B^{-1})^T(c_B) - A^T(A_B^{-1})^T(c_B + \lambda^*c_B) - A^T(A_B^{-1})^T(c_B + \lambda^*c_B) - A^T(A_B^{-1})^T(c_B + \lambda^*c_B) \tag{6.6}$$

where $\alpha$ is a constant. According to (6.4), we have

$$ \alpha = \sum_{l \in B \setminus j} \frac{(c_i + \lambda^*c_i)p_l}{1 + p_k} - \frac{c_j + \lambda^*c_j}{1 + p_k} + c_i + \lambda^*c_i $$

$$ = \sum_{l \in B} \frac{(c_i + \lambda^*c_i)p_l}{1 + p_k} + \frac{c_i + \lambda^*c_i}{1 + p_k} - \frac{c_j + \lambda^*c_j}{1 + p_k} $$

$$ = \frac{(c_B + \lambda^*c_B)^TA_B^{-1}u + c_i + \lambda^*c_i - c_j - \lambda^*c_j}{1 + p_k} $$

$$ = \frac{(c_B + \lambda^*c_B)^TA_B^{-1}(a_j - a_i) + c_i + \lambda^*c_i - c_j - \lambda^*c_j}{1 + p_k} \tag{6.6}$$

$$ = \frac{(c_B + \lambda^*c_B)^T(A_B^{-1}a_j - e_k) + c_i + \lambda^*c_i - c_j - \lambda^*c_j}{1 + p_k} $$

$$ = \frac{(c_B + \lambda^*c_B)^T(A_B^{-1}a_j) - c_j - \lambda^*c_j}{1 + p_k} $$

$$ = \frac{(c_B + \lambda^*c_B)^T(A_B^{-1}a_j)}{1 + p_k} $$

$$ = \frac{c_B + \lambda^*c_B)^T(A_B^{-1}a_j - c_j)}{1 + p_k} $$

$$ = \frac{c_B^TA_B^{-1}a_j - c_j}{1 + p_k} + \lambda^*(c_B^TA_B^{-1}a_j - c_j) $$
where $c_i$ and $c_j$ are the entries in $c$, with indices corresponding to $a_i$ and $a_j$, and $\bar{c}_i$ and $\bar{c}_j$ are the entries in $\bar{c}$ and defined similarly.

Recall in this case

$$\lambda^* = \max_{j \in \mathcal{N}, \bar{z}_N^j > 0} \frac{\bar{z}_N^j}{\bar{z}_N^j} - \frac{(A^{-1}_G a_j)^T c_B - c_j}{(A^{-1}_G a_j)^T \bar{c}_B - \bar{c}_j}. \quad (6.7)$$

Substitute the definition of $\lambda^*$ from (6.7) into (6.6), we observe that $\alpha = 0$ and thus the dual feasible is guaranteed in the new dictionary. This proves Theorem 3.1. \qed
References


