Learning large graphical models with convex optimization

Terrence Alsup

May 18, 2020

Abstract

Graphical models are a powerful tool for understanding the dependencies between many variables. In the case where the joint distribution of the variables is Gaussian we can learn the graph structure of the dependencies by solving a semidefinite program (SDP) for the precision matrix. In section 2 we will explore three different algorithms for solving this SDP, two of which are based on coordinate descent. Section 3 briefly discusses several nice statistical properties of the computed solution. Finally, in section 4 we apply the algorithms described in section 2 to solve a real problem of determining the underlying graph structure of many different gene expressions in patients with tumors.

1 Introduction

1.1 Graphical models

A central problem in data science is to understand the relationships between multiple variables $x_1, \dots, x_p \in \mathbb{R}$. This is useful for prediction, anomaly detection, simulation, and other inference problems. These relationships are completely determined by the joint probability distribution of the random vector $\boldsymbol{x} = (x_1, \dots, x_p)^T \in \mathbb{R}^p$. A graphical model is simply a representation of this joint distribution in terms of a graph $\mathcal{G} = (V, E)$, where each node represents a variable and each edge between two nodes represents a statistical dependence between those two variables. Figure 1 shows an example for $p = 5$, where x_1, x_2, x_3 are dependent on each other and x_4, x_5 are dependent on each other. There are two important properties of graphical models that we will use (see [1], [2]):

1. Variables in disconnected components are statistically independent from each other. In the example, x_1, x_2, x_3 are independent of x_4, x_5.

2. Each variable is conditionally independent from the other variables given the values of its neighbors. Here x_1 and x_3 are conditionally independent given x_2. This is called the Markov property.

A special case is when the joint distribution is a multivariate Gaussian distribution, referred to as a Gaussian graphical model. A Gaussian distribution is completely characterized by its mean $\boldsymbol{\mu} \in \mathbb{R}^p$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$ and has the density

$$\phi(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}\sqrt{|\det \boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{1}$$

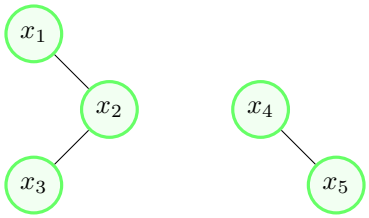

Figure 1: A simple graphical model with $p = 5$ covariates.

whenever $\mathbf{\Sigma} \succ 0$. If $\mathbf{\Sigma}$ has a zero eigenvalue, then the corresponding eigenvector indicates a direction in which the data does not vary. We could alternatively specify the precision matrix $\mathbf{\Theta} = \mathbf{\Sigma}^{-1}$, which is sometimes more useful. The graph $\mathcal{G}$ has an intuitive structure given either the matrix $\mathbf{\Sigma}$ or $\mathbf{\Theta}$. If $\mathbf{\Sigma}_{ij} = 0$, then x_i and x_j are independent, meaning they are a part of separate components. If $\mathbf{\Theta}_{ij} = 0$, then x_i and x_j are conditionally independent given the remaining variables, meaning that there is no edge between the nodes corresponding to variables x_i and x_j (see [3]). For the example in figure 1 the precision and covariance matrices would have the forms

$$\mathbf{\Sigma} = \begin{bmatrix} * & * & * & 0 & 0 \\ * & * & * & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}, \quad \mathbf{\Theta} = \begin{bmatrix} * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}$$

where $*$ represents a non-zero element. The block structure comes from the fact that there are two independent components in the graph. To learn a Gaussian graphical model we just need to estimate the mean and covariance.

1.2 Covariance estimation

Given a random sample of data $\{\boldsymbol{x}_i\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} N(\boldsymbol{\mu}, \mathbf{\Sigma})$, we can estimate the mean and covariance using maximum likelihood estimation. The maximum likelihood estimate of $\boldsymbol{\mu}$ is the sample mean $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i$ regardless of what $\mathbf{\Sigma}$ is. The likelihood function to be maximized is

$$\underset{\mathbf{\Sigma} \succeq 0}{\text{maximize}}\ L(\mathbf{\Sigma}) = \prod_{i=1}^n \frac{1}{(2\pi)^{p/2}\sqrt{|\det \mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \mathbf{\Sigma}^{-1}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})\right)$$

This is equivalent to minimizing the negative log-likelihood,

$$\underset{\mathbf{\Sigma} \succeq 0}{\text{minimize}}\ -\log L(\mathbf{\Sigma}) = \frac{np}{2}\log(2\pi) + \frac{n}{2}\log\det \mathbf{\Sigma} - \frac{1}{2}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \mathbf{\Sigma}^{-1}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})$$

We can ignore the constants and it will also help to replace $\mathbf{\Sigma}$ with $\mathbf{\Theta}^{-1}$ to avoid the inverse.

$$\underset{\mathbf{\Theta} \succeq 0}{\text{minimize}}\quad -\log\det \mathbf{\Theta} + \frac{1}{n}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \mathbf{\Theta}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})$$

Since the sum is a scalar,

$$\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T\boldsymbol{\Theta}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})=\operatorname{tr}\left[\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T\boldsymbol{\Theta}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})\right]=\frac{1}{n}\sum_{i=1}^{n}\operatorname{tr}\left[(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T\boldsymbol{\Theta}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})\right]$$

Using the trick that $\operatorname{tr}[AB]=\operatorname{tr}[BA]$, we simplify

$$\frac{1}{n}\sum_{i=1}^{n}\operatorname{tr}\left[(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T\boldsymbol{\Theta}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})\right]=\frac{1}{n}\sum_{i=1}^{n}\operatorname{tr}\left[(\boldsymbol{x}_i-\bar{\boldsymbol{x}})(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T\boldsymbol{\Theta}\right]=\operatorname{tr}\left[\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T\boldsymbol{\Theta}\right]$$

However, the sample covariance matrix $\boldsymbol{S}$ is given by

$$\boldsymbol{S}=\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T$$

which means that the maximum likelihood problem becomes equivalent to

$$\underset{\boldsymbol{\Theta}\succeq 0}{\text{minimize}} \quad -\log\det\boldsymbol{\Theta}+\operatorname{tr}[\boldsymbol{S}\boldsymbol{\Theta}] \tag{2}$$

This is a convex objective function (see [4]), but it is only defined whenever $\boldsymbol{\Theta}\succ 0$. As long as this is the case, the objective is actually differentiable and the KKT conditions for the optimal solution are $-\boldsymbol{\Theta}^{-1}+\boldsymbol{S}^T=0$ (see [5] for many useful matrix formulas). Since $\boldsymbol{S}$ is symmetric, we have the solution $\boldsymbol{\Theta}^{-1}=\boldsymbol{\Sigma}=\boldsymbol{S}$. Of course, this is problematic if $\boldsymbol{S}$ is not strictly positive definite. In particular, this will always happen whenever there are more variables than samples, $p>n$, meaning that maximum likelihood estimation is ill-suited for high-dimensional problems.

1.3 Sparsity and the graphical lasso

The covariance, and equivalently the precision, matrix has $\frac{p(p+1)}{2}$ total entries that need to be estimated since it is symmetric. In the high-dimensional setting when $p\gg n$, this is a huge number of parameters to estimate and we cannot hope to accurately estimate all of these given only a relatively small amount of data. Instead we will try to only estimate a few important entries for $\boldsymbol{\Theta}$. Similar to the lasso we will look for a sparse solution to the maximum likelihood problem, which is enforced by adding an ℓ_1 regularizer to the objective function.

$$\underset{\boldsymbol{\Theta}\succeq 0}{\text{minimize}}\ f(\boldsymbol{\Theta})=-\log\det\boldsymbol{\Theta}+\operatorname{tr}[\boldsymbol{S}\boldsymbol{\Theta}]+\lambda\|\boldsymbol{\Theta}\|_1 \tag{3}$$

We call the solution to this regularized problem the graphical lasso. Here $\lambda>0$ is the regularization parameter and the norm $\|\cdot\|_1$ is defined differently than the usual matrix 1-norm:

$$\|\boldsymbol{\Theta}\|_1=\sum_{i=1}^{p}\sum_{j=1}^{p}|\boldsymbol{\Theta}_{ij}|$$

This is the same as the ℓ_1 norm of the vectorized matrix $\operatorname{vec}(\boldsymbol{\Theta})$. Adding $\lambda\|\boldsymbol{\Theta}\|_1$ means that the objective function will remain convex, so this is still a semidefinite program (SDP) for the variable $\boldsymbol{\Theta}$. Also note that since the only constraint on $\boldsymbol{\Theta}$ is that it is positive definite, Slater's condition

holds and we have strong duality.

To derive the dual SDP we need to use the fact that the vector ℓ_1 and ℓ_∞ norms are dual. We use the notation $\|\cdot\|_\infty$ to refer to the ℓ_∞ norm of the vectorized matrix, not to the usual matrix ∞-norm.

$$\|\boldsymbol{Y}\|_\infty = \max_{i,j=1,\ldots,p} |\boldsymbol{Y}_{ij}|$$

Therefore, we can write

$$\min_{\boldsymbol{\Theta}\succeq 0} \; -\log\det\boldsymbol{\Theta} + \operatorname{tr}\left[\boldsymbol{S}\boldsymbol{\Theta}\right] + \lambda\|\boldsymbol{\Theta}\|_1 = \min_{\boldsymbol{\Theta}\succeq 0} \; -\log\det\boldsymbol{\Theta} + \operatorname{tr}\left[\boldsymbol{S}\boldsymbol{\Theta}\right] + \lambda \max_{\|\boldsymbol{Y}\|_\infty \leq 1} \operatorname{tr}\left[\boldsymbol{\Theta}\boldsymbol{Y}\right]$$

Of course we can rescale $\boldsymbol{Y}$ to obtain the equivalent min-max problem

$$\min_{\boldsymbol{\Theta}\succeq 0} \max_{\|\boldsymbol{Y}\|_\infty \leq \lambda} \; -\log\det\boldsymbol{\Theta} + \; \operatorname{tr}\left[\boldsymbol{\Theta}\left(\boldsymbol{S}+\boldsymbol{Y}\right)\right]$$

By strong duality, this is equivalent to

$$\max_{\|\boldsymbol{Y}\|_\infty \leq \lambda} \min_{\boldsymbol{\Theta}\succeq 0} \; -\log\det\boldsymbol{\Theta} + \; \operatorname{tr}\left[\boldsymbol{\Theta}\left(\boldsymbol{S}+\boldsymbol{Y}\right)\right]$$

We have already solved this minimization problem in 2, although with $\boldsymbol{S}+\boldsymbol{Y}$ replaced with $\boldsymbol{S}$ and the optimal solution is $\boldsymbol{\Theta} = \left(\boldsymbol{S}+\boldsymbol{Y}\right)^{-1}$. Plugging this in gives

$$\log\det\left(\boldsymbol{S}+\boldsymbol{Y}\right) + \; \operatorname{tr}\left[\boldsymbol{I}\right] = \log\det\left(\boldsymbol{S}+\boldsymbol{Y}\right) + p$$

meaning that the dual SDP of 3 is

$$\begin{aligned} \underset{\boldsymbol{Y}\succeq 0}{\text{maximize}} \; & g(\boldsymbol{Y}) = \log\det\left(\boldsymbol{S}+\boldsymbol{Y}\right) + p \\ \text{subject to} \quad & \|\boldsymbol{Y}\|_\infty \leq \lambda \end{aligned} \tag{4}$$

where the dual variable satisfies $\boldsymbol{Y} + \boldsymbol{S} = \boldsymbol{\Theta}^{-1} = \boldsymbol{\Sigma}$. The box constraint on the dual variable says that the estimated covariance matrix cannot be further than λ from the sample covariance $\boldsymbol{S}$ in this norm.

Since strong duality holds for the graphical lasso problem we can equivalently solve either the primal or dual SDP. For the primal SDP the variable is the precision matrix, whereas the variable in the dual SDP is $\boldsymbol{Y} = \boldsymbol{\Sigma} - \boldsymbol{S}$, meaning the variable is essentially the covariance matrix. In terms of our graphical model $\mathcal{G}$, a sparse precision matrix means that most pairs of variables x_i, x_j will be conditionally independent of each other given the remaining variables. That is, these nodes are not direct neighbors. A sparse precision matrix does not imply that the covariance is sparse, however, so the variables may all still be dependent on each other. A sparse precision matrix just means that the graph $\mathcal{G}$ does not have many large cliques. A practical benefit of using a sparse estimate of the precision matrix is that the model is more readily interpretable with clear dependencies between variables. In the next section we consider several algorithms that solve either the primal SDP 3 or the dual SDP 4.

2 Computing the graphical lasso solution

Since 3 and 4 are SDPs we can solve them using CVX in Matlab, which will then use the SDPT3 solver. We quickly find that this method becomes extremely slow once the dimension p is around 50. For large graphical models, this is insufficient and so we will now look at several algorithms designed specifically for the graphical lasso problem. For most of these algorithms the basic building block is coordinate descent.

2.1 Coordinate descent algorithms

Coordinate descent (CD) algorithms are extremely simple algorithms that minimize only along one coordinate at a time as opposed to methods like gradient descent, which updates all coordinates after every iteration.

Algorithm 1: Coordinate descent algorithm

Input: Objective function f, starting point $\boldsymbol{x}_0 \in \mathbb{R}^d$
Result: Optimal solution $\boldsymbol{x}^*$ and value $f(\boldsymbol{x}^*)$
Initialize $k = 0$, $\boldsymbol{x}^{(k)} = \boldsymbol{x}_0$;
while *not converged* **do**
  $i \leftarrow \mod(k, d) + 1$;
  $x_i^{(k+1)} \leftarrow \underset{x \in \mathbb{R}}{\operatorname{argmin}} f(x_1^k, \ldots, x_{i-1}^{(k)}, x, x_{i+1}^{(k)} \ldots, x_d^{(k)})$;
  $k \leftarrow k + 1$;
end

Algorithm 1 shows the basic coordinate descent algorithm where exact minimization is done along each coordinate. This is not always possible and in these cases one can use gradient descent with line search. For simple quadratic problems, though, we can find the exact minimizer at each step. Under some technical assumptions including strong convexity, continuously differentiable, and Lipschitz it can be shown (see Theorem 3 and Assumption 1 of [6]) that coordinate descent converges linearly.

$$f(\boldsymbol{x}^{(k)}) - f(\boldsymbol{x}^*) \leq (1 - c)^{k/d} (f(\boldsymbol{x}_0) - f(\boldsymbol{x}^*)) \tag{5}$$

for $k = nd$ with $n = 1, 2, 3, \ldots$ and where the constant $c > 0$ depends proportionally on the strong convexity constant and inverse proportionally on the dimension d and Lipschitz constant. This is a result on the objective function after every complete cycle through all coordinates, where $\lfloor k/d \rfloor$ is the number of total cycles that have been completed by iteration k. For the convergence criteria we can set a tolerance ϵ on either the relative change in the objective function value

$$\frac{f(\boldsymbol{x}^{((n-1)d)}) - f(\boldsymbol{x}^{(nd)})}{|f(\boldsymbol{x}^{((n-1)d)})|} < \epsilon$$

or the relative change in the solution

$$\frac{\|\boldsymbol{x}^{(nd)} - \boldsymbol{x}^{((n-1)d)}\|}{\|\boldsymbol{x}^{((n-1)d)}\|} < \epsilon$$

where $k = nd$ for $n = 1, 2, 3, \ldots$ meaning we only check the convergence criteria after each complete cycle over all of the coordinates.

Two important problems that can be solved with coordinate descent and are useful for the algorithms presented in the next section are the lasso problem:

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}} \ \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + \lambda\|\boldsymbol{x}\|_1 \tag{6}$$

and a quadratic program (QP) with box constraints:

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{\ell} \leq \boldsymbol{x} \leq \boldsymbol{u} \end{aligned} \tag{7}$$

Note that equation 6 is written differently from how the lasso problem is usually defined. In both problems, $\boldsymbol{A}$ is a symmetric positive semidefinite matrix.

For the lasso problem, the optimality condition for the update to variable x_i can be found by setting the subgradient of the function $x \mapsto f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_d)$ to zero.

$$A_{ii}x_i + \sum_{j\neq i} A_{ij}x_j + b_i + \lambda\gamma_i = 0$$

where the subgradient $\gamma_i \in \partial|x|$

$$\gamma_i \in \begin{cases} \{-1\} & x_i < 0 \\ \{1\} & x_i > 0 \\ [-1, 1] & x_i = 0 \end{cases}$$

In vector form this is

$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} + \lambda\boldsymbol{\gamma} = \boldsymbol{0} \tag{8}$$

Since $\boldsymbol{A} \succ 0$ we know that $A_{ii} > 0$ and $\operatorname{sgn}(x_i) = \operatorname{sgn}(A_{ii}x_i)$. Now consider three cases. If $x_i > 0$, then $\gamma_i = 1$ and

$$A_{ii}x_i = -\sum_{j\neq i} A_{ij}x_j - b_i - \lambda$$

Since $\lambda > 0$ we know that $-\sum_{j\neq i} A_{ij}x_j - b_i > 0$ as well so we can write

$$x_i = \frac{\left|-\sum_{j\neq i} A_{ij}x_j - b_i\right| - \lambda}{A_{ii}}$$

assuming $x_i > 0$. For $x_i < 0$ we can follow similar steps with $\gamma_i = -1$ now.

$$x_i = \frac{-\sum_{j\neq i} A_{ij}x_j - b_i + \lambda}{A_{ii}} = -\left(\frac{\left|-\sum_{j\neq i} A_{ij}x_j - b_i\right| - \lambda}{A_{ii}}\right)$$

since $-\sum_{j\neq i} A_{ij}x_j - b_i < 0$ in this case. In the last case we just have $x_i = 0$. We can now write the formula for the update to the variable x_i

$$x_i \leftarrow \operatorname{sgn}\left(-\sum_{j\neq i} A_{ij}x_j - b_i\right)\left(\frac{\left|-\sum_{j\neq i} A_{ij}x_j - b_i\right| - \lambda}{A_{ii}}\right)^+ \tag{9}$$

with $z^+ = \max\{0, z\}$. This is just soft thresholding and we use this update formula in algorithm 1 to solve the lasso problem 6.

For the quadratic program with box constraints the update is even simpler since the objective function is quadratic. Specifically, $x \mapsto f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_d)$ is a quadratic function with positive leading coefficient since $\boldsymbol{A} \succ 0$. Setting the derivative to zero gives

$$x^* = \frac{-\sum_{j\neq i} A_{ij}x_j - b_i}{A_{ii}}$$

as the optimal point. However, we have to enforce the constraint that $\ell_i \leq x_i \leq u_i$. Because the leading coefficient is positive, the function is decreasing on $(-\infty, x^*]$ and increasing on $[x^*, \infty)$. Therefore, the update is given by taking x_i to be the point in the box $[\ell_i, u_i]$ that is as close to x^* as possible

$$x_i \leftarrow \begin{cases} \ell_i & x^* \leq \ell_i \\ x^* & \ell_i \leq x_i \leq u_i \\ u_i & x^* \geq u_i \end{cases}, \quad \text{with} \quad x^* = \frac{-\sum_{j\neq i} A_{ij}x_j - b_i}{A_{ii}} \tag{10}$$

Using this update formula in algorithm 1 will solve the quadratic program with box constraints 7.

These special cases are implemented in the Matlab functions `lasso.m` and `QP_box.m`, respectively [1]. The implementations were verified against CVX for relatively small random problems of dimension around $d = 20$. Figure 2 shows that for both problems we have the theoretical linear convergence given by equation 5. We also see from comparing the top and bottom rows that increasing the eigenvalues of $\boldsymbol{A}$ makes the problem more convex and we require fewer iterations in both algorithms. For the lasso problem, increasing λ has a similar effect in that fewer iterations are needed for convergence, but also that the computed solution is more sparse. Of course, neither of these two algorithms can be used directly to solve the original problem 3 that we are interested in, but they can be used as building blocks to compute the solution one row and column at a time.

2.2 Glasso and DP-Glasso

The most common approaches to solving either 3 or 4 revolve around partitioning the covariance and precision matrices and updating them one row and column at a time in a coordinate descent-like fashion (see [3], [7], and [8]). We will use notation consistent with these papers. Let $\boldsymbol{\Theta}_{-i,-i}$ denote the $(p-1)\times(p-1)$ submatrix obtained by deleting the i-th row and column from $\boldsymbol{\Theta}$, let $\boldsymbol{\theta}_{-i,i}$ be the $(p-1)\times 1$ column vector obtained by deleting the i-th entry from the i-th column of

[1] See the end of this report for the code listings.

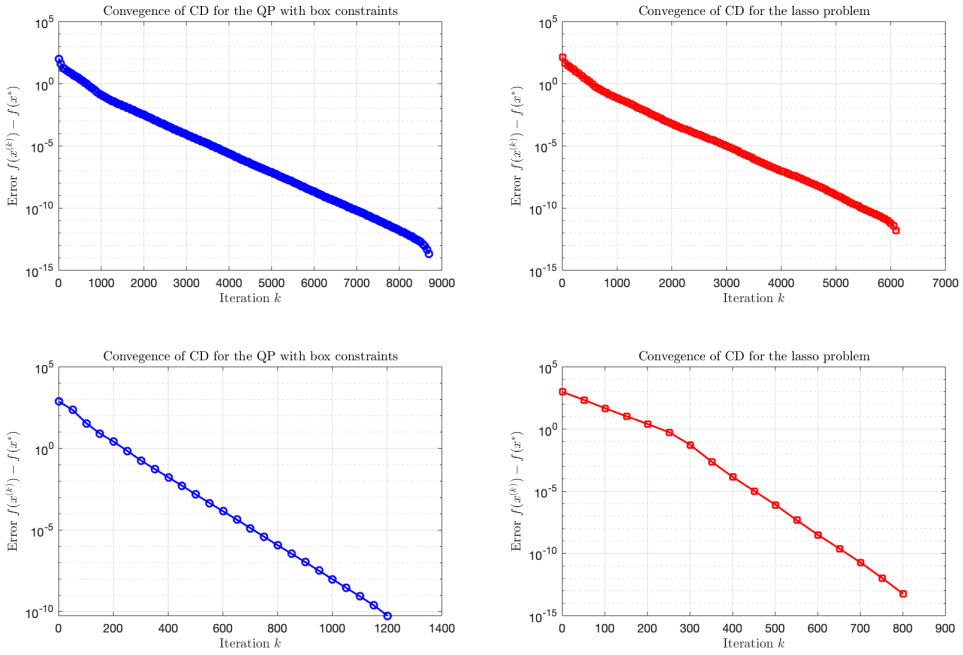

Figure 2: Performance of the CD algorithm 1 on the QP with box constraints 7 (left column in blue) and the lasso problem 6 (right column in red) on random problems of dimension $d = 50$. In all cases we set a tolerance of $\epsilon = 10^{-12}$ on the relative difference of objective function values and for the lasso problem we set $\lambda = 1$. The top row shows the performance on a random problem where $\boldsymbol{A} = \boldsymbol{B}^T\boldsymbol{B} + \boldsymbol{I}$, where $\boldsymbol{B} \in \mathbb{R}^{d\times d}$ is a random matrix and $\boldsymbol{I}$ is the identity, ensuring that the smallest eigenvalue of $\boldsymbol{A}$ is at least 1. For the bottom row we use $\boldsymbol{A} = \boldsymbol{B}^T\boldsymbol{B} + 10 \times \boldsymbol{I}$, so that the smallest eigenvalues is now at least 10.

$\boldsymbol{\Theta}$, and let $\theta_{i,i} = \boldsymbol{\Theta}_{i,i}$. If we rearrange the rows and columns of $\boldsymbol{\Theta}$ so that the i-th row and column is now last, then we can write

$$\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\Theta}_{-i,-i} & \boldsymbol{\theta}_{-i,i} \\ \boldsymbol{\theta}_{-i,i}^T & \theta_{i,i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Theta}_{11} & \boldsymbol{\theta}_{12} \\ \boldsymbol{\theta}_{12}^T & \theta_{22} \end{bmatrix}$$

We will use the simpler notation on the right with $\boldsymbol{\Theta}_{11}, \boldsymbol{\theta}_{12}, \theta_{22}$ where the coordinate is now implicit and will be clear from context. We define $\boldsymbol{\Sigma}_{11}, \boldsymbol{\sigma}_{12}, \sigma_{22}$ similarly for the covariance matrix as well as $\boldsymbol{s}_{12}$ for the sample covariance matrix. Any other block matrices in the following will also use this notation. The relation between the blocks of $\boldsymbol{\Theta}$ and $\boldsymbol{\Sigma}$ is given by the Schur complement since $\boldsymbol{\Theta}^{-1} = \boldsymbol{\Sigma}$.

$$\begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\sigma}_{12} \\ \boldsymbol{\sigma}_{12}^T & \sigma_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Theta}_{11}^{-1} + \frac{\boldsymbol{\Theta}_{11}^{-1}\boldsymbol{\theta}_{12}\boldsymbol{\theta}_{12}^T\boldsymbol{\Theta}_{11}^{-1}}{\theta_{22} - \boldsymbol{\theta}_{12}^T\boldsymbol{\Theta}_{11}^{-1}\boldsymbol{\theta}_{12}} & -\frac{\boldsymbol{\Theta}_{11}^{-1}\boldsymbol{\theta}_{12}}{\theta_{22} - \boldsymbol{\theta}_{12}^T\boldsymbol{\Theta}_{11}^{-1}\boldsymbol{\theta}_{12}} \\ -\frac{\boldsymbol{\theta}_{12}^T\boldsymbol{\Theta}_{11}^{-1}}{\theta_{22} - \boldsymbol{\theta}_{12}^T\boldsymbol{\Theta}_{11}^{-1}\boldsymbol{\theta}_{12}} & \frac{1}{\theta_{22} - \boldsymbol{\theta}_{12}^T\boldsymbol{\Theta}_{11}^{-1}\boldsymbol{\theta}_{12}} \end{bmatrix} \tag{11}$$

We can derive another useful relation by using the fact that $\boldsymbol{\Theta}\boldsymbol{\Sigma} = \boldsymbol{I}$, which gives a system of four equations:

$$\begin{aligned} \boldsymbol{\Sigma}_{11}\boldsymbol{\Theta}_{11} + \boldsymbol{\sigma}_{12}\boldsymbol{\theta}_{21} &= \boldsymbol{I} \\ \boldsymbol{\sigma}_{21}\boldsymbol{\Theta}_{11} + \sigma_{22}\boldsymbol{\theta}_{21} &= \boldsymbol{0} \\ \boldsymbol{\Sigma}_{11}\boldsymbol{\theta}_{12} + \boldsymbol{\sigma}_{12}\theta_{22} &= \boldsymbol{0} \\ \boldsymbol{\sigma}_{21}\boldsymbol{\theta}_{12} + \sigma_{22}\theta_{22} &= 1 \end{aligned}$$

By rearranging these equations we obtain the relation[2]

$$\begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\sigma}_{12} \\ \boldsymbol{\sigma}_{12}^T & \sigma_{22} \end{bmatrix} = \begin{bmatrix} \left(\boldsymbol{\Theta}_{11} - \frac{\boldsymbol{\theta}_{12}\boldsymbol{\theta}_{21}}{\theta_{22}}\right)^{-1} & -\boldsymbol{\Sigma}_{11}\frac{\boldsymbol{\theta}_{12}}{\theta_{22}} \\ -\boldsymbol{\Sigma}_{11}\frac{\boldsymbol{\theta}_{12}}{\theta_{22}} & \frac{1}{\theta_{22}}\left(1 + \frac{\boldsymbol{\theta}_{21}\boldsymbol{\Sigma}_{11}\boldsymbol{\theta}_{12}}{\theta_{22}}\right) \end{bmatrix} \tag{12}$$

The relations 11 and 12 will be essential for switching between $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$ whenever the rows and columns are updated.

2.2.1 Glasso

The first of these types of algorithms is the original Glasso algorithm introduced in [3] and largely based on the work done in [7]. As a starting point consider the optimality conditions for 3 and finding a zero subgradient. The matrix derivative of the objective function is

$$-\boldsymbol{\Theta}^{-1} + \boldsymbol{S} + \lambda\boldsymbol{\Gamma} = \boldsymbol{0} \tag{13}$$

where each entry $\gamma_{ij} = \boldsymbol{\Gamma}_{ij}$ is in the subdifferential $\partial|x|$ at $x = \boldsymbol{\Theta}_{ij}$ just as in the lasso problem 6. Instead of trying to solve for $\boldsymbol{\Theta}$ the Glasso algorithm solves for $\boldsymbol{\Sigma}$ which satisfies

$$\boldsymbol{\Sigma} - \boldsymbol{S} - \lambda\boldsymbol{\Gamma} = \boldsymbol{0}$$

[2]The bottom-right entry is actually slightly wrong in the paper [8], although this particular relation is not used anywhere else.

By looking at the upper-right block and using equation 12 for $\boldsymbol{\sigma}_{12}$ we obtain

$$\boldsymbol{\Sigma}_{11}\frac{\boldsymbol{\theta}_{12}}{\theta_{22}} + \boldsymbol{s}_{12} + \lambda\boldsymbol{\Gamma}_{12} = \mathbf{0}$$

Because $\boldsymbol{\Theta} \succ 0$, $\theta_{22} > 0$ and $\text{sgn}\,(\boldsymbol{\theta}_{12}/\theta_{22}) = \boldsymbol{\Gamma}_{12}$, so this is exactly the optimality condition we derived for the lasso problem 6 whenever $\boldsymbol{A} = \boldsymbol{\Sigma}_{11}$, $\boldsymbol{b} = \boldsymbol{s}_{12}$ and $\boldsymbol{\gamma} = \boldsymbol{\Gamma}_{12}$. The solution of this lasso problem is $\boldsymbol{\theta}_{12}/\theta_{22}$, which we can then use to update $\boldsymbol{\sigma}_{12}$ and build an estimate of the inverse $\boldsymbol{\Theta}$ using equations 11 and 12. We repeat this procedure, cycling over all of the rows and columns of $\boldsymbol{\Sigma}$ until convergence. Algorithm 2 describes the procedure, which is implemented in the Matlab function `glasso.m`.

Algorithm 2: Glasso

Input: Sample covariance $\boldsymbol{S}$, regularization parameter λ
Result: Estimated precision $\boldsymbol{\Theta}$ and covariance $\boldsymbol{\Sigma}$

Initialize $\boldsymbol{\Sigma} = \boldsymbol{S} + \lambda\boldsymbol{I}$, $\boldsymbol{\Theta} = \boldsymbol{I}$, $k = 0$;
while *not converged* **do**
  $i \leftarrow \mod(k, p) + 1$;
  $\boldsymbol{x} \leftarrow \texttt{lasso}(\boldsymbol{\Sigma}_{11}, \boldsymbol{s}_{12}, \lambda)$; `// x = θ12/θ22`
  $\boldsymbol{\sigma}_{12} \leftarrow -\boldsymbol{\Sigma}_{11}\boldsymbol{x}$; `// update covariance`
  $\theta_{22} \leftarrow 1/(\sigma_{22} + \boldsymbol{x}^T\boldsymbol{\sigma}_{12})$; `// update precision`
  $\boldsymbol{\theta}_{12} \leftarrow \theta_{22}\boldsymbol{x}$;
  $k \leftarrow k + 1$;
end

If λ is large, then the solution $\boldsymbol{x}$ returned by `lasso` should be sparse, which means that we are updating the rows and columns of the matrix $\boldsymbol{\Theta}$ with sparse vectors resulting in a sparse matrix.

There are several problems with the Glasso algorithm, which are discussed in detail in [8]. Perhaps the biggest is that the primal objective function 3 may not decrease monotonically. Note that the matrix $\boldsymbol{\Sigma}_{11}$ and variable $\boldsymbol{\theta}_{12}$ in the lasso problem are actually coupled. This means after that each iteration we may not have $\boldsymbol{\Theta}^{-1} = \boldsymbol{\Sigma}$, so the dual variable $\boldsymbol{Y} \neq \boldsymbol{\Sigma} - \boldsymbol{S}$. However, it turns out that algorithm 2 actually does coordinate ascent on the dual problem 4 (Theorem 4.1 of [8]). This is illustrated in figure 3 where the dual objective is seen to be monotonically increasing. Using the starting point $\boldsymbol{\Sigma} = \boldsymbol{S} + \lambda\boldsymbol{I}$ ensures that $\boldsymbol{\Sigma}$ is dual feasible and positive definite. This will ensure that $\boldsymbol{\Sigma}$ remains positive definite throughout. However, because $\boldsymbol{\Theta} \neq \boldsymbol{\Sigma}^{-1}$ we are not guaranteed that $\boldsymbol{\Theta}$ is also positive definite at any given step. Despite this, once the algorithm converges all rows and columns of $\boldsymbol{\Theta}$ are updated and it will be the inverse $\boldsymbol{\Sigma}^{-1}$. Since our goal is to uncover the structure of the graphical model through $\boldsymbol{\Theta}$ it may be better to ensure that $\boldsymbol{\Theta}$ is always positive definite. This is what the next algorithm DP-Glasso attempts to do.

2.2.2 DP-Glasso

In many ways the DP-Glasso algorithm is opposite to the Glasso algorithm. The Glasso algorithm starts from the KKT optimality conditions 13 of the primal problem 3 to derive a condition on

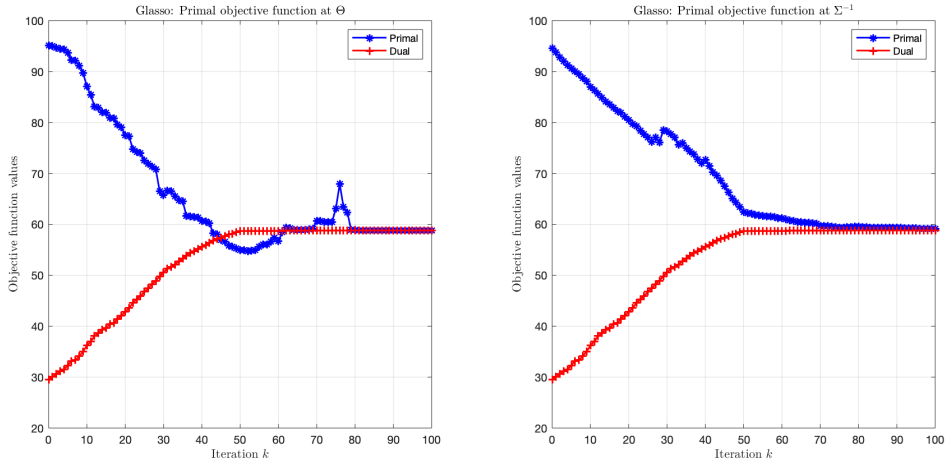

Figure 3: The primal (blue) and dual (red) objective values at each iteration of the Glasso algorithm on a random problem of dimension $p = 50$. Here $\boldsymbol{S}$ is the sample covariance of $n = 5$ data points drawn from a $N(\mathbf{0}, 1.2^2 \times \boldsymbol{I})$ distribution and $\lambda = 0.5$. The left plot shows the primal objective values $f(\boldsymbol{\Theta})$ and the dual objective values $g(\boldsymbol{\Sigma})$ while the right plot shows the primal objective values $f(\boldsymbol{\Sigma}^{-1})$ using the true primal variable as well as the dual objective values $g(\boldsymbol{\Sigma})$. In the left plot the two curves cross indicating that $\boldsymbol{\Theta}$ is not primal feasible (i.e. not positive definite).

the block $\boldsymbol{\sigma}_{12}$. This condition is translated using 11 and 12 to a condition on the primal variable $\boldsymbol{\theta}_{12}$ and is recognized to be the optimality conditions for the lasso problem 8, which is then solved to update $\boldsymbol{\Theta}$ and $\boldsymbol{\Sigma}$. The DP-Glasso does all of this except from the point of view of the dual problem. Rather than go through the tedious derivations we refer the reader to [8] and instead present a diagram 4 that summarizes the relationship nicely. The DP-Glasso algorithm itself is outlined in algorithm 3 and implemented in the Matlab function `dpglasso.m`.

Based on figure 4 we expect that DP-Glasso will have the same problem that Glasso has only for the covariance matrix $\boldsymbol{\Sigma}$. Figure 5 shows that this is indeed the case where the dual objective is now the one that is not monotone increasing. This is to be expected because the box-constrained QP that solves for $\boldsymbol{y}_{12}$ depends on $\boldsymbol{\Theta}_{11}$, which is coupled with $\boldsymbol{y}_{12}$. In fact the authors of [8] propose a corrected version called P-Glasso, which uses equation 11 to enforce that $\boldsymbol{\Theta} = \boldsymbol{\Sigma}^{-1}$ after each iteration in Glasso. In the diagram 4 this corresponds to changing the expression in terms of the primal variable. The same could in theory be done for DP-Glasso. Each correction however is a rank one update and costs $O(p^2)$ operations. Since Glasso returns $\boldsymbol{\Sigma}$ and DP-Glasso returns $\boldsymbol{\Theta}$ there is really no need to use P-Glasso unless one needs both matrices at once. Moreover, upon convergence we have that $\boldsymbol{\Theta}^{-1} = \boldsymbol{\Sigma}$ for both Glasso and DP-Glasso (P-Glasso guarantees this pre-asymptotically).

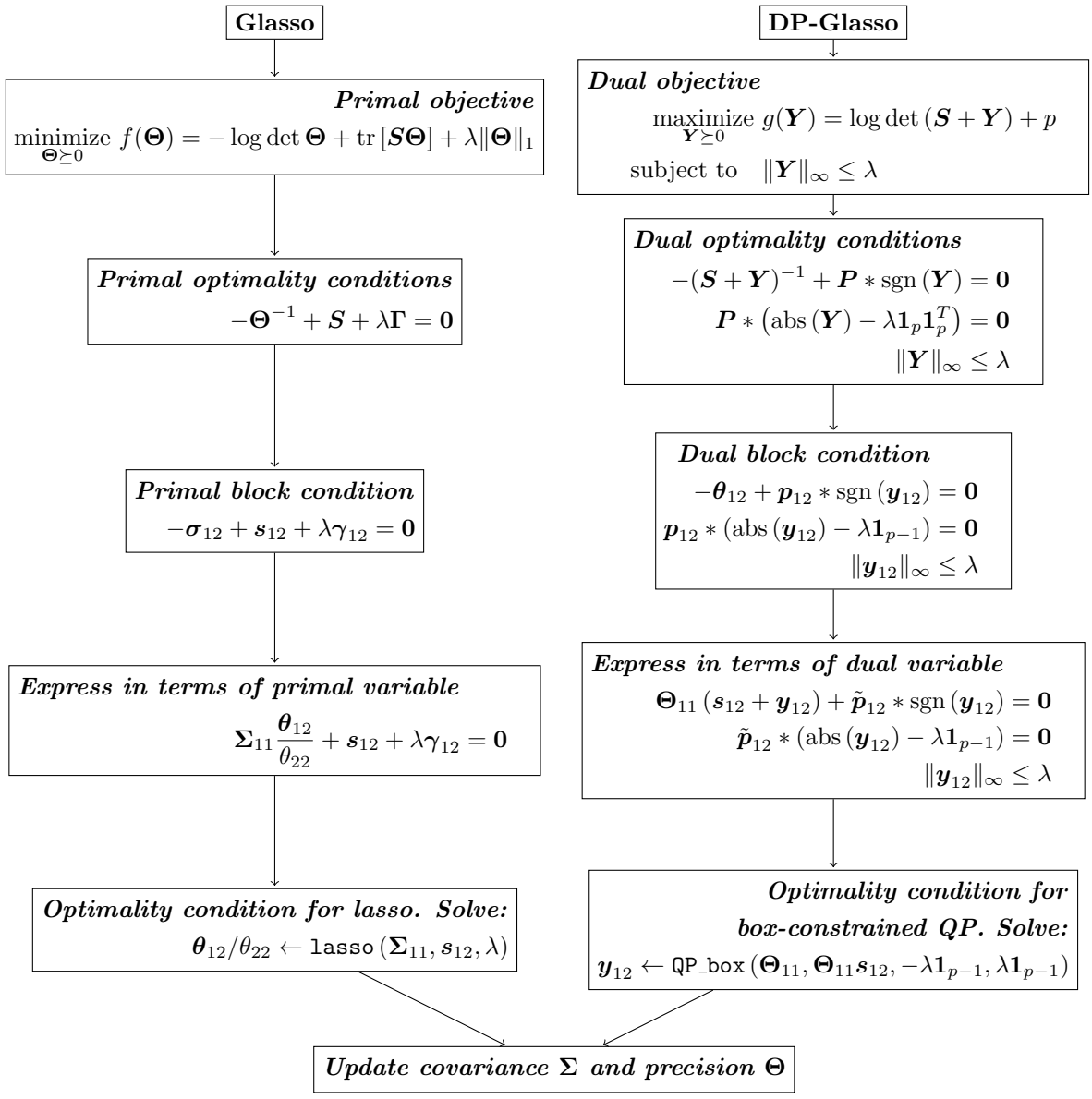

Figure 4: A summary of the derivations of the Glasso and DP-Glasso algorithms presented in [8] and [3]. The matrix $\boldsymbol{P} = \operatorname{abs}(\boldsymbol{\Theta})$ has non-negative entries, $\tilde{\boldsymbol{p}}_{12} = \boldsymbol{p}_{12}/\sigma_{22}$ (recall $\sigma_{22} = s_{22} + \lambda$), and $\boldsymbol{1}_p$ is a $p \times 1$ vector of all 1's.

Algorithm 3: DP-Glasso

Input: Sample covariance $\boldsymbol{S}$, regularization parameter λ

Result: Estimated precision $\boldsymbol{\Theta}$ and covariance $\boldsymbol{\Sigma}$

Initialize $\boldsymbol{\Sigma} = \boldsymbol{S} + \lambda \boldsymbol{I}$, $\boldsymbol{\Theta} = \boldsymbol{I}$, $k = 0$;

while *not converged* **do**

  $i \leftarrow \mod(k, p) + 1$;

  $\boldsymbol{y} \leftarrow \texttt{QP\_box}(\boldsymbol{\Theta}_{11}, \boldsymbol{\Theta}_{11}\boldsymbol{s}_{12}, -\lambda \mathbf{1}_{p-1}, \lambda \mathbf{1}_{p-1})$; // $\boldsymbol{\sigma}_{12} = \boldsymbol{y}_{12} + \boldsymbol{s}_{12}$

  $\boldsymbol{\sigma}_{12} \leftarrow \boldsymbol{y}_{12} + \boldsymbol{s}_{12}$; // update covariance

  $\boldsymbol{\theta}_{12} \leftarrow -\boldsymbol{\Theta}_{11}\boldsymbol{\sigma}_{12}/\sigma_{22}$; // update precision

  $\theta_{22} \leftarrow (1 - \boldsymbol{\theta}_{12}^T \boldsymbol{\sigma}_{12})/\sigma_{22}$;

  $k \leftarrow k + 1$;

end

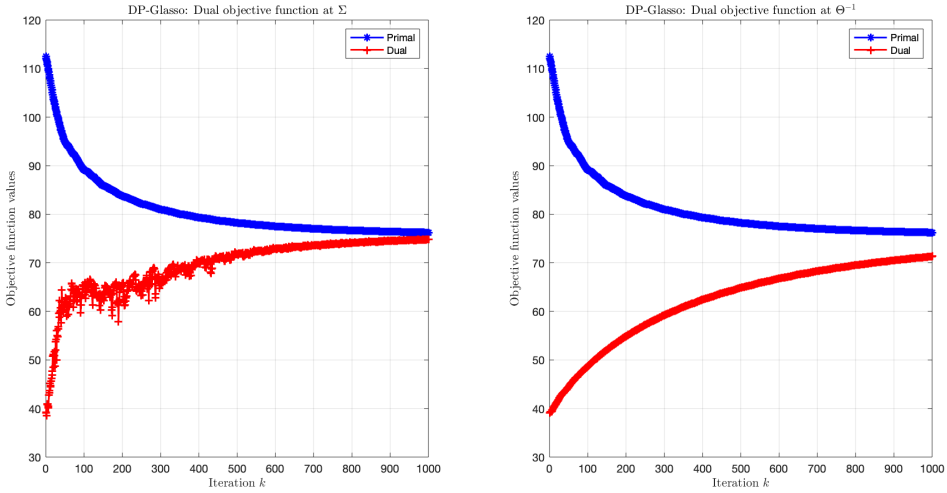

Figure 5: The primal (blue) and dual (red) objective values at each iteration of the DP-Glasso algorithm on a random problem of dimension $p = 50$. Here $\boldsymbol{S}$ is the sample covariance of $n = 5$ data points drawn from a $N(\mathbf{0}, 16 \times \boldsymbol{I})$ distribution and $\lambda = 0.5$. The left plot shows the primal objective values $f(\boldsymbol{\Theta})$ and the dual objective values $g(\boldsymbol{\Sigma})$ while the right plot shows the primal objective values $f(\boldsymbol{\Theta})$ and the dual objective values $g(\boldsymbol{\Theta}^{-1} - \boldsymbol{S})$ using the true dual variable.

2.3 ADMM

A completely different approach for solving the primal problem 3, which is not based on coordinate descent, is the alternating direction method of multipliers (ADMM) (see [9] and [10]). The problem is equivalent to

$$\begin{aligned} \underset{\boldsymbol{\Theta} \succeq 0}{\text{minimize}} \quad & -\log\det \boldsymbol{\Theta} + \operatorname{tr}\left[\boldsymbol{S}\boldsymbol{\Theta}\right] + \lambda \|\boldsymbol{Z}\|_1 \\ \text{subject to} \quad & \boldsymbol{\Theta} = \boldsymbol{Z} \end{aligned}$$

If μ is the penalty parameter, then the augmented Lagrangian is

$$L(\boldsymbol{\Theta}, \boldsymbol{Z}, \boldsymbol{Y}) = -\log\det \boldsymbol{\Theta} + \operatorname{tr}\left[\boldsymbol{S}\boldsymbol{\Theta}\right] + \lambda \|\boldsymbol{Z}\|_1 + \operatorname{tr}\left[\boldsymbol{Y}\left(\boldsymbol{\Theta} - \boldsymbol{Z}\right)\right] + \frac{\mu}{2}\|\boldsymbol{\Theta} - \boldsymbol{Z}\|_F^2$$

Note that the dual variable $\boldsymbol{Y}$ is exactly the dual variable we have seen before, $\boldsymbol{Y} + \boldsymbol{S} = \boldsymbol{\Theta}^{-1}$, meaning at the end of the algorithm we will easily be able to recover the exact inverse $\boldsymbol{\Sigma}$. ADMM alternates between minimizing over $\boldsymbol{\Theta}$ and $\boldsymbol{Z}$. At iteration k the minimization over $\boldsymbol{\Theta}$ is given by

$$\boldsymbol{\Theta}^{(k+1)} = \underset{\boldsymbol{\Theta} \succeq 0}{\operatorname{argmin}}\ L(\boldsymbol{\Theta}, \boldsymbol{Z}^{(k)}, \boldsymbol{Y}^{(k)})$$

Now the objective is differentiable with respect to $\boldsymbol{\Theta}$ so setting the matrix derivative to zero gives the optimality condition

$$\boldsymbol{\Theta} - \frac{1}{\mu}\boldsymbol{\Theta}^{-1} = \boldsymbol{Z}^{(k)} - \frac{1}{\mu}\boldsymbol{S} - \frac{1}{\mu}\boldsymbol{Y}^{(k)}$$

Therefore, the eigenvalues of the matrix on the left and right must agree (see [10]). Let the eigendecomposition of the right-hand side (which is a symmetric matrix) be

$$\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T = \boldsymbol{Z}^{(k)} - \frac{1}{\mu}\boldsymbol{S} - \frac{1}{\mu}\boldsymbol{Y}^{(k)} = \boldsymbol{\Theta} - \frac{1}{\mu}\boldsymbol{\Theta}^{-1}$$

The eigenvalues ρ_i of $\boldsymbol{\Theta}$ must satisfy

$$\rho_i - \frac{1}{\rho_i \mu} = \lambda_i$$

where λ_i is an eigenvalue on the diagonal of $\boldsymbol{\Lambda}$. Solving the quadratic equation and using the fact that $\rho_i > 0$ gives

$$\boldsymbol{\Theta}^{(k+1)} = \frac{1}{2}\boldsymbol{V}\operatorname{diag}\left(\lambda_i + \sqrt{\lambda_i^2 + \frac{4}{\mu}}\right)\boldsymbol{V}^T$$

The next step in ADMM is minimization over $\boldsymbol{Z}$:

$$\boldsymbol{Z}^{(k+1)} = \underset{\boldsymbol{Z} \succeq 0}{\operatorname{argmin}}\ L(\boldsymbol{\Theta}^{(k+1)}, \boldsymbol{Z}, \boldsymbol{Y}^{(k)})$$

The optimality conditions are given by setting the subgradient of the augmented Lagrangian to zero

$$\boldsymbol{Z} = \boldsymbol{\Theta}^{(k+1)} + \frac{1}{\mu}\boldsymbol{Y}^{(k)} - \frac{\lambda}{\mu}\boldsymbol{\Gamma}$$

Just as in the lasso problem 6 and 8, the solution $\boldsymbol{Z}^{(k+1)}$ is given by soft thresholding. The last step of ADMM is updating the dual variable $\boldsymbol{Y}$, which is trivial. The entire procedure is summarized

Algorithm 4: ADMM for the graphical lasso

Input: Sample covariance $\boldsymbol{S}$, regularization parameter λ, penalty parameter μ
Result: Estimated precision $\boldsymbol{\Theta}$ and covariance $\boldsymbol{\Sigma}$

Initialize $k = 0$, $\boldsymbol{\Theta}^{(0)} = \boldsymbol{I}$, $\boldsymbol{Z}^{(0)} = \boldsymbol{I}$, $\boldsymbol{Y}^{(0)} = \boldsymbol{I}$;
while *not converged* **do**
  $\boldsymbol{V}\text{diag}\left(\lambda_i\right)\boldsymbol{V}^T \leftarrow \texttt{eig}\left(\boldsymbol{Z}^{(k)} - \frac{1}{\mu}\boldsymbol{S} - \frac{1}{\mu}\boldsymbol{Y}^{(k)}\right)$;
  $\boldsymbol{\Theta}^{(k+1)} \leftarrow \boldsymbol{V}\text{diag}\left(\lambda_i + \sqrt{\lambda_i^2 + \frac{4}{\mu}}\right)\boldsymbol{V}^T$;
  $\boldsymbol{Z}^{(k+1)} \leftarrow \text{sgn}\left(\boldsymbol{\Theta}^{(k+1)} + \frac{1}{\mu}\boldsymbol{Y}^{(k)}\right)\left(\left|\boldsymbol{\Theta}^{(k+1)} + \frac{1}{\mu}\boldsymbol{Y}^{(k)}\right| - \frac{\lambda}{\mu}\right)^+$; `// entry-wise update`
  $\boldsymbol{Y}^{(k+1)} \leftarrow \mu\left(\boldsymbol{\Theta}^{(k+1)} - \boldsymbol{Z}^{(k+1)}\right)$;
  $k \leftarrow k + 1$;
end
Set $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}^{(k)}$ and $\boldsymbol{\Sigma} \leftarrow \boldsymbol{S} + \boldsymbol{Y}^{(k)}$;

in algorithm 4, which is implemented in the Matlab function `glasso_admm.m`.

Just like Glasso and DP-Glasso, figure 6 shows the difference between the dual objective values at the dual variable $\boldsymbol{Y}$ and $\boldsymbol{\Theta}^{-1} - \boldsymbol{S}$, which are only equal upon convergence. The only real noticeable difference between the dual objective values $g(\boldsymbol{Y})$ and $g(\boldsymbol{\Theta}^{-1} - \boldsymbol{S})$ is near the beginning and due to the initialization. After just a few iterations the difference becomes negligible. Also notice that after the first few iterations the primal objective decreases monotonically while the dual objective increases monotonically. Next we will explore this behavior in more detail as well as compare against Glasso and DP-Glasso.

2.4 Comparison of algorithms

Since we have mentioned it several times already we start by comparing how close $\boldsymbol{\Theta}^{-1}$ is to $\boldsymbol{\Sigma}$ for these three algorithms[3]. Specifically, figure 7 shows the Frobenius norm $\|\boldsymbol{\Sigma}\boldsymbol{\Theta} - \boldsymbol{I}\|_F$ at each iteration. We see that Glasso and DP-Glasso are comparable in this regard, but ADMM is noticeably better. However, this is far from the complete picture since ADMM does a significant amount of work each iteration to compute an eigenvalue decomposition and update the entire matrix, whereas Glasso and DP-Glasso only update a single row and column vector.

To get a better sense of the amount of work done by each algorithm in iteration we time the results for different values of λ. These results are shown in figure 8. Surprisingly ADMM's performance deteriorates rapidly for larger λ even though Glasso and DP-Glasso perform better. The results for Glasso and DP-Glasso are explainable at least. For DP-Glasso more iterations are required than Glasso when λ is small but the iterations are much cheaper. Whenever λ is small the box constraints are very tight and so fewer iterations are required within `QP_box`. On the other hand, when λ is small, the updates are unlikely to be very sparse which means more coordinate descent iterations will be needed within `lasso`. The poor performance of ADMM can partially be

[3]All algorithms were tested against CVX for small dimensional problems.

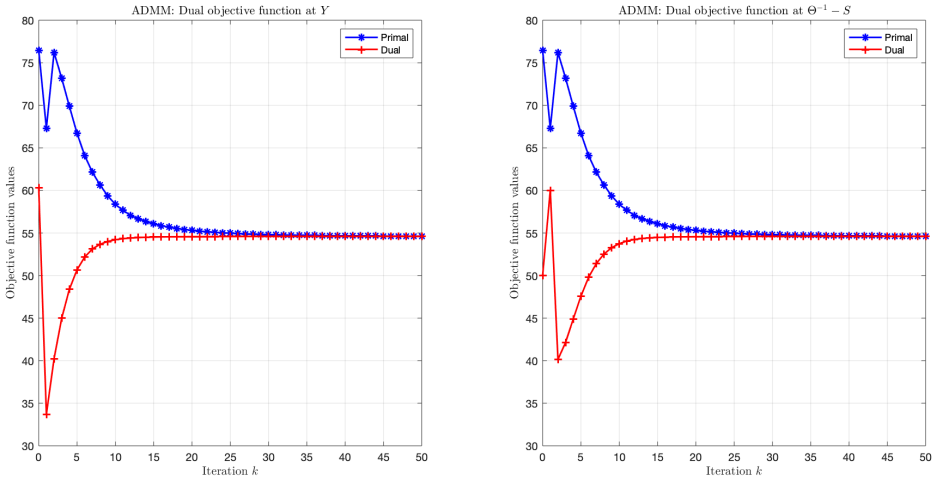

Figure 6: The primal (blue) and dual (red) objective values at each iteration of ADMM on a random problem of dimension $p = 50$. Here $\boldsymbol{S}$ is the sample covariance of $n = 5$ data points drawn from a $N(\boldsymbol{0}, \boldsymbol{I})$ distribution, $\lambda = 0.5$, and $\mu = 1.0$. The left plot shows the primal objective values $f(\boldsymbol{\Theta})$ and the dual objective values $g(\boldsymbol{Y})$ while the right plot shows the primal objective values $f(\boldsymbol{\Theta})$ and the dual objective values $g(\boldsymbol{\Theta}^{-1} - \boldsymbol{S})$ using the true dual variable.

explained by figure 9. For ADMM, we see that when λ is large essentially all entries have absolute value above 10^{-8}, but very few (just the diagonal elements) have absolute value larger than 10^{-4}. It is extremely unlikely for a random problem that almost all entries would be in the range $[10^{-8}, 10^{-4}]$, which suggests there is a numerical stability issue, one that Glasso and DP-Glasso do not experience. This means that in each iteration of ADMM $\boldsymbol{\Theta}$ is a dense matrix when in principle it should be sparse. This is what is likely causing the slowdown for large λ as seen in figure 8.

The last comparison we make between the three methods is comparing how fast they converge to the optimal objective value. So far, there does not seem to be result in the literature about the rate of convergence for Glasso and DP-Glasso since these are not strict coordinate descent algorithms (remember the variables were coupled). The results on a random problem are shown in figure 10. For small λ we see what appears to be linear convergence which aligns with the theoretical coordinate descent rate 5. For larger λ, Glasso and DP-Glasso converge very fast, needing at most a couple of cycles over all of the coordinates. For all choices of λ, ADMM appears to converge slowly but steadily. In terms of both cost (runtime) and accuracy, the coordinate descent-based algorithms Glasso and DP-Glasso appear to be comparable to each other and superior to ADMM. Of course, this seems reasonable because ADMM is very general while Glasso and DP-Glasso are designed specifically to handle the graphical lasso problem.

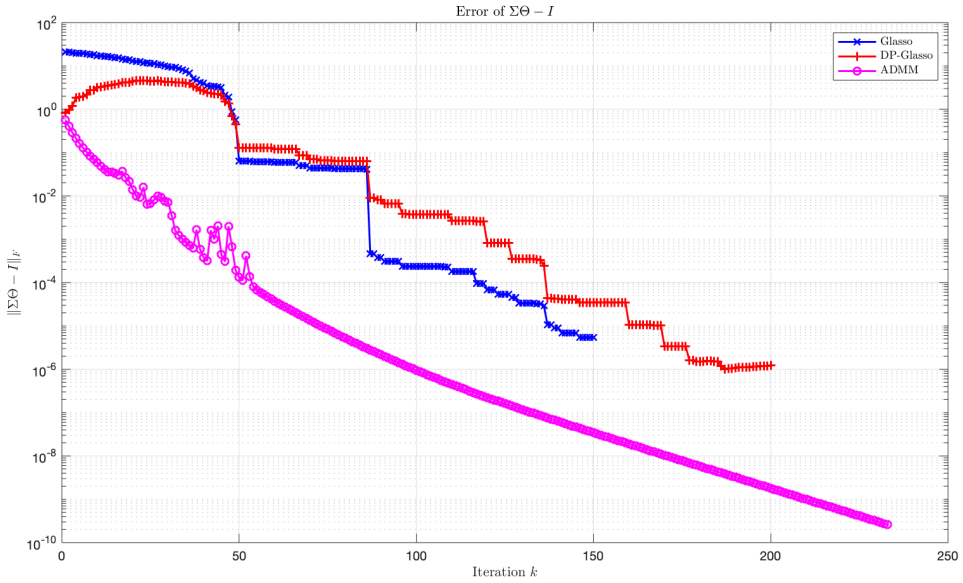

Figure 7: The Frobenius norms $\|\boldsymbol{\Sigma\Theta} - \boldsymbol{I}\|_F$ at each iteration in Glasso, DP-Glasso, and ADMM on a random $p = 50$ dimensional problem. We set $\lambda = 1$, $\mu = 1$, and drew $n = 5$ data points from $N(\mathbf{0}, \boldsymbol{I})$ to compute the sample covariance $\boldsymbol{S}$.

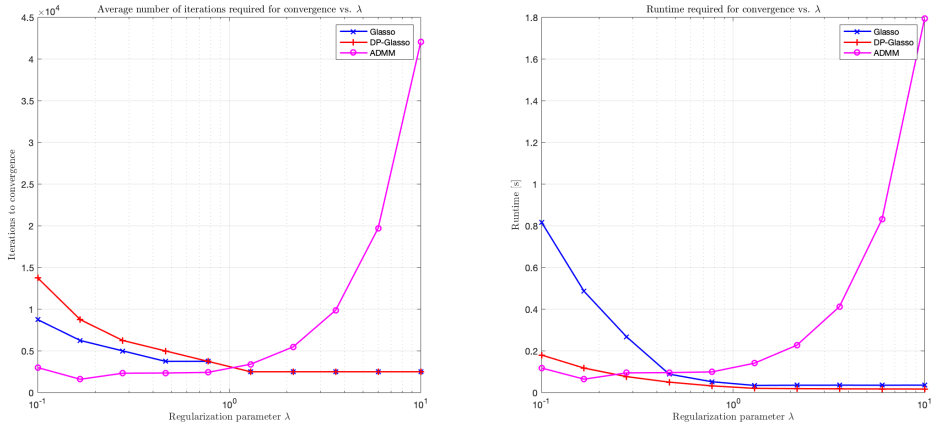

Figure 8: We set a tolerance of $\epsilon = 10^{-9}$ on the relative change in the primal objective function value and let all three algorithms run until convergence for 10 different values of λ (0.1 to 10 log-spaced) on the same random problem with $p = 50$ and $n = 20$. For ADMM we set $\mu = 1$. The left plot shows the number of iterations required for convergence (averaged over 25 trials) while the right plot shows the actual run time in seconds. For all timings the processor was 1.6 GHz Dual-Core Intel Core i5.

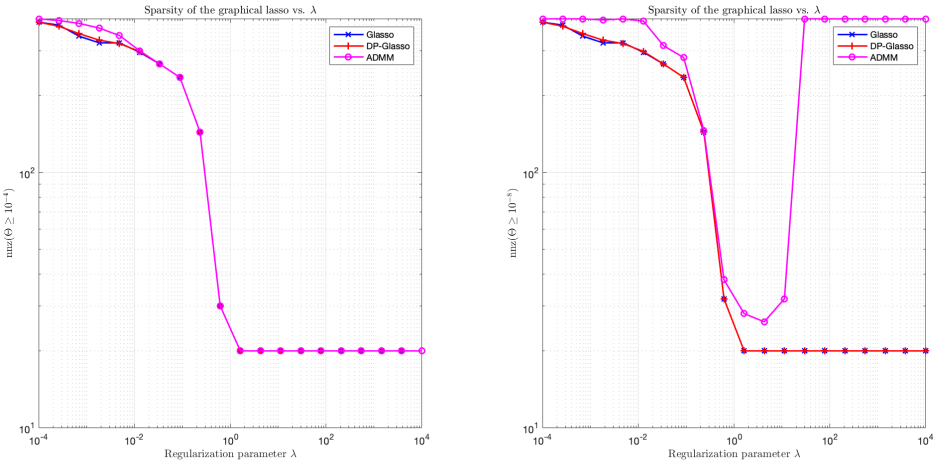

Figure 9: The number of non-zero (nnz) elements of the computed precision matrix $\boldsymbol{\Theta}$ for 20 different values of λ (log-spaced) on a random $p = 20$ dimensional problem with $n = 10$. In the left plot we set a threshold of 10^{-4} so any entry with absolute value larger than 10^{-4} is counted as a zero. For the plot on the right the threshold is set to 10^{-8}.

3 Statistical guarantees of the graphical lasso

In this section we will assume that we have computed the optimal solution to the graphical lasso problem 3 so as not to worry about any convergence or numerical issues. We will also now distinguish between the computed solution $\boldsymbol{\Theta}$ and the true underlying precision matrix $\boldsymbol{\Theta}_*$ that determines the distribution from which the data was drawn: $\{\boldsymbol{x}\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} N(\mathbf{0}, \boldsymbol{\Theta}_*^{-1})$. From a statistical point of view the solution $\boldsymbol{\Theta}$ has several attractive properties related to the regularization parameter λ, which can be found in [11]. As a disclaimer we note that these results are for a slightly different formulation of the problem 3 where the diagonal entries are not penalized, but we proceed nonetheless.

As we increase the number of samples n relative to the dimension p we expect to get a more accurate answer on average. The choice of λ determines the trade-off between the bias and variance of our estimator. When λ is large the problem is heavily regularized so we obtain an estimator with very low variance at the price of potentially being very inaccurate. Similarly, when λ is small we obtain an estimator that is correct on average, but by itself unreliable due to having a high variance. The best choice for the regularization parameter is on the order of $\sqrt{\frac{\log p}{n}}$. Under this choice for λ it can be shown (see Proposition 11.9 of [11]) that with high probability

$$\|\boldsymbol{\Theta} - \boldsymbol{\Theta}_*\|_F \leq c\lambda$$

where c is a constant depending on the dimension p, the number of zeros per row of $\boldsymbol{\Theta}_*$, and the matrix 2-norm of $\|\boldsymbol{\Theta}_*\|_2$. This result says that when we have more samples it is better to take λ smaller since the sample covariance $\boldsymbol{S}$ is a more reliable of $\boldsymbol{\Theta}_*^{-1}$. Figure 11 shows the results for two different choices of λ as we increase the sample size. The toy problem used here was set up

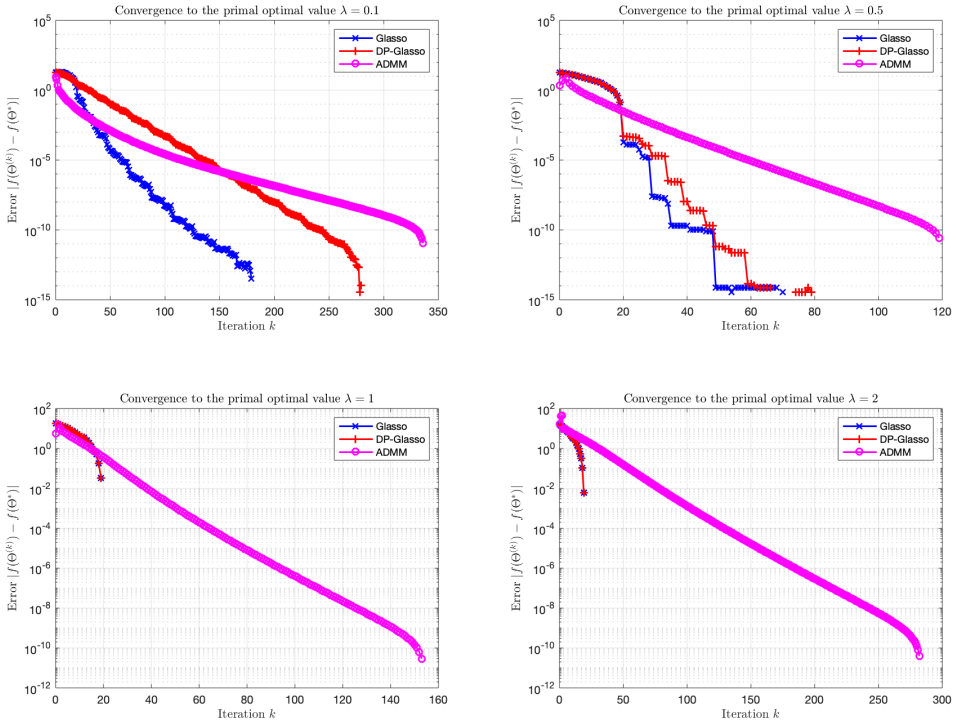

Figure 10: The convergence of Glasso, DP-Glasso, and ADMM on a random problem for $\lambda = 0.1$ (top-left), $\lambda = 0.5$ (top-right), $\lambda = 1$ (bottom-left), and $\lambda = 2$ (bottom-left). The dimension is $p = 20$ and $n = 10$ data points were drawn from $N(\mathbf{0}, \boldsymbol{I})$. We set a tight tolerance of $\epsilon = 10^{-12}$ on the relative error of the primal objective function f between cycles. For ADMM we kept $\mu = 1$.

with the following steps:

1. Use the Matlab function $\boldsymbol{A} = \texttt{sprandn}(p, p, 1/p)$ to generate a sparse random matrix.
2. Set $\boldsymbol{\Theta}_* = \boldsymbol{A}^T\boldsymbol{A} + 0.1 \times \boldsymbol{I}$, which ensures that the resulting matrix will be positive definite (taking the constant 0.1 to instead be larger will result in faster runtimes).
3. Compute the true covariance matrix $\boldsymbol{\Sigma}_* = \boldsymbol{\Theta}_*^{-1}$.
4. Draw random samples $\{\boldsymbol{x}\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} N(\boldsymbol{0}, \boldsymbol{\Sigma}_*)$ for different $n \leq p$ and compute the sample covariance $\boldsymbol{S}$. In our case $p = 500$ and we set $n = 5, 10, 50, 100$, and 500.

Taking $\lambda \sim \sqrt{\frac{\log p}{n}}$ gives a much more accurate answer than when λ is a fixed constant once the sample size is large enough relative to the dimension. Presumably the difference would be more clear for much larger dimensions p since the number of samples needed is $O(\log p)$. Even in 11 when $p = 500$, $\log p \approx 6$ which is between the second and third points on the plot, exactly where the curves cross.

Another interesting property of the graphical lasso solution, which is more relevant to the problem of uncovering the underlying structure of graphical model, is that for the same choice of $\lambda \sim \sqrt{\frac{\log p}{n}}$ the solution has no false inclusions with high probability. This means that if $(\boldsymbol{\Theta}_*)_{ij} = 0$, then $\boldsymbol{\Theta}_{ij} = 0$ (see Proposition 11.10 of [11]). However, the pre-factor for the choice of λ depends upon a complicated parameter relating to an assumption called α-incoherence. This makes the result difficult to check numerically. Although there are lots of other variables involved, this at least gives us some idea of how to choose λ.

4 Modeling gene expression data

We now attempt to solve a large real problem using a data set ([12]) of gene expressions in patients with different types of tumors. The original data set has $n = 801$ patients with $p = 20,531$ measured gene expressions per patient. Because the original data set is so large we will truncate it to instead just look at the first $p = 1000$ genes (even just loading the original data set in Matlab takes a non-trivial amount of time). We solve this problem using our DP-Glasso implementation for three different choices of λ. In all three cases we set a tolerance of $\epsilon = 10^{-4}$ for the relative change in the solution $\boldsymbol{\Theta}$ and set the maximum number of iterations to be 10,000 (10 full cycles over all coordinates). Figures 12, 13, and 14 show the results for $\lambda = c\sqrt{\frac{\log p}{n}}$ with $c = 10, 50$, and 100, respectively. All of the computed solutions are sparse and become more sparse as we increase λ, however $c = 50$ appears to be a good middle ground. From figure 13 we can see the direct dependencies of each gene on all of the others (out of the first 1000). In particular, there appear to be several rows containing many non-zero entries. This corresponds to a gene that is directly related to many other genes. It would be interesting to see if this is a known gene with certain medical or biological implications. However, we do not have access to this information from the data set.

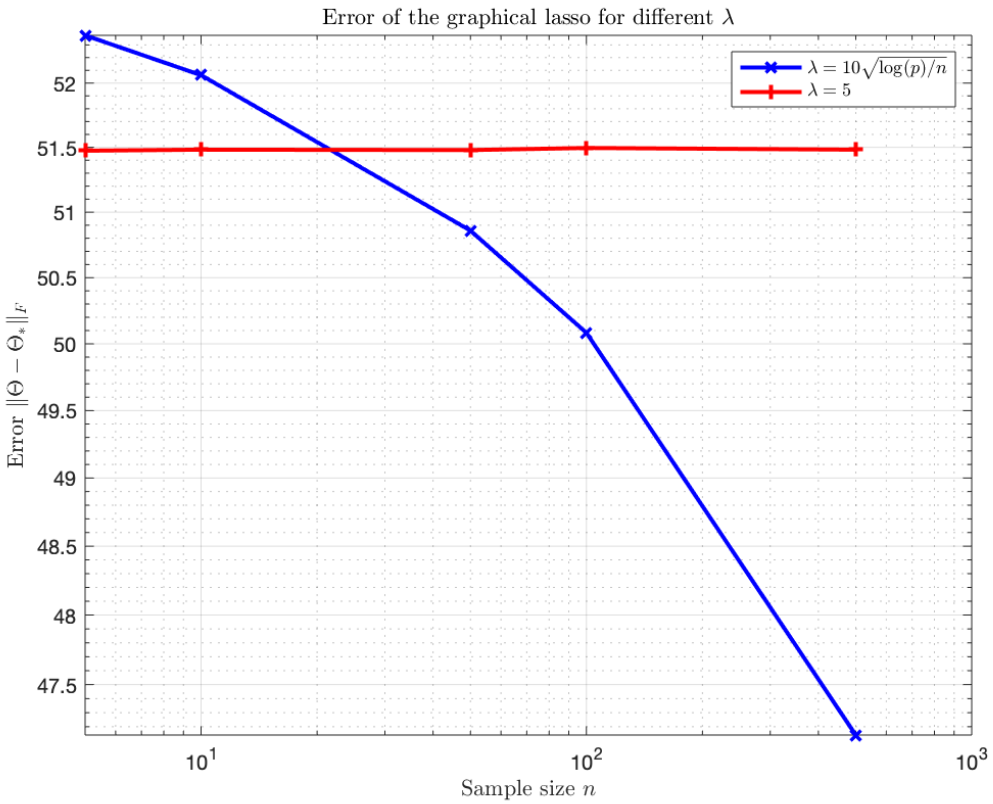

Figure 11: The error of the graphical lasso on the toy problem for $\lambda = 10\sqrt{\frac{\log p}{n}}$ and $\lambda = 5$ fixed as the sample size n varies. For computing the solution we used the `dpglasso.m` function and set a tolerance of $\epsilon = 10^{-4}$ on the Frobenius norm of the relative change in the precision matrix after each cycle.

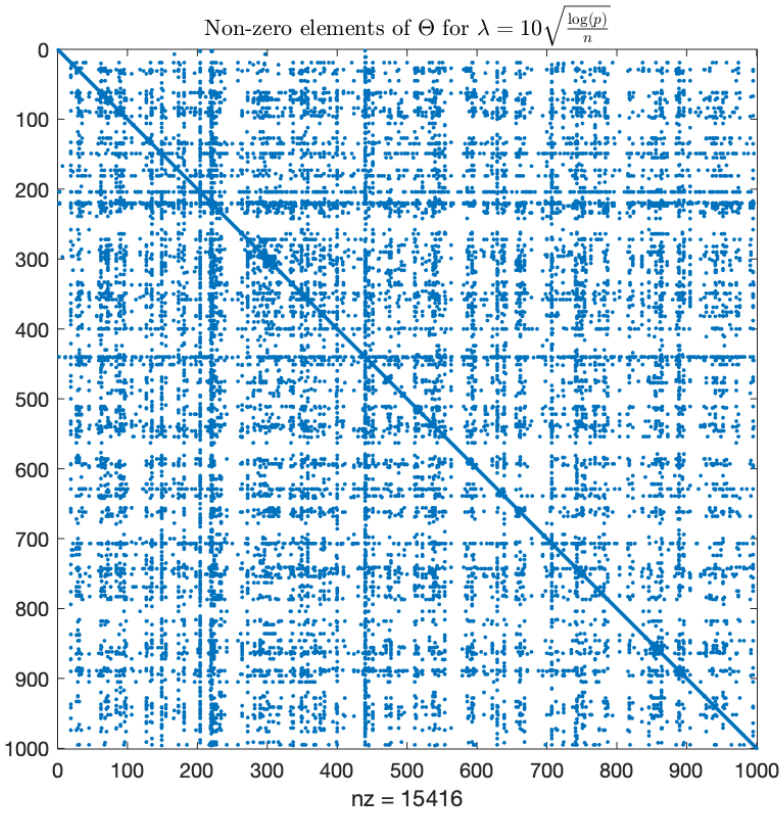

Figure 12: The non-zero elements of the graphical lasso for $\lambda = 10\sqrt{\frac{\log p}{n}} \approx 0.93$. The algorithm reached the maximum number of iterations (roughly 1 hour of CPU time) before converging.

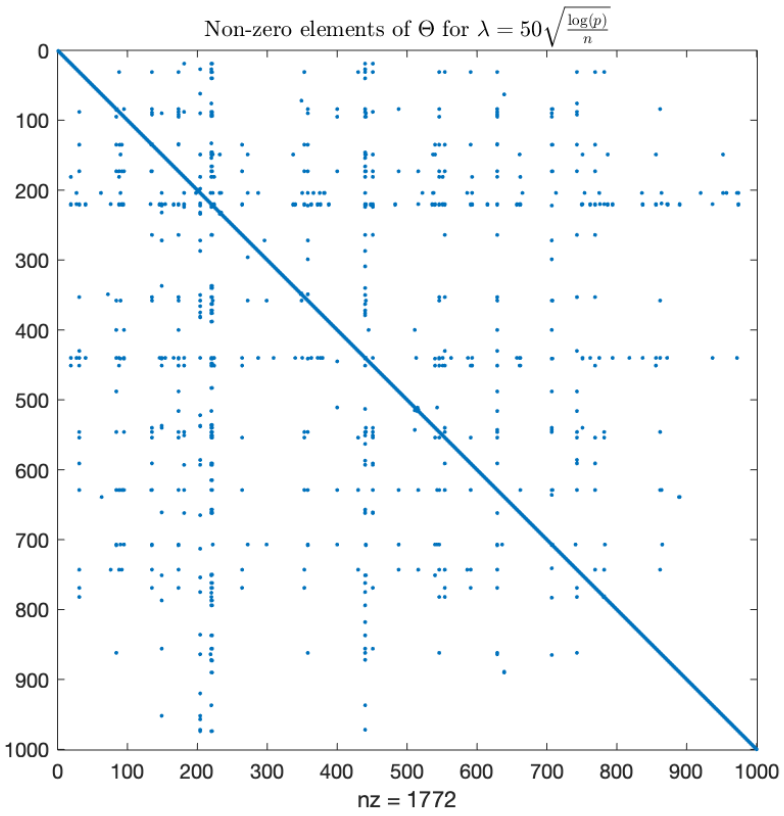

Figure 13: The non-zero elements of the graphical lasso for $\lambda = 50\sqrt{\frac{\log p}{n}} \approx 4.64$.

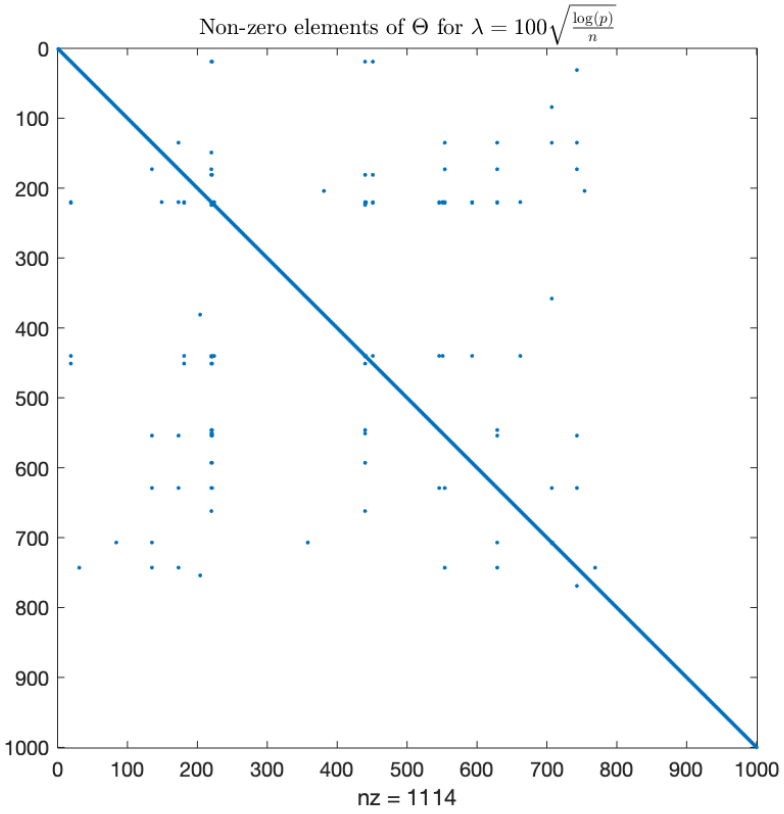

Figure 14: The non-zero elements of the graphical lasso for $\lambda = 100\sqrt{\frac{\log p}{n}} \approx 9.23$.

5 Conclusion

We have seen that convex optimization plays an essential role in being able to analyze data with a graphical structure to it. This has many real-world applications, such as in biostatistics where we analyzed a data set of gene expressions for patients with tumors. The best performing algorithms for this problem are block coordinate descent-type algorithms such as Glasso and DP-Glasso, which work by updating one row and column of the precision and covariance matrices at a time. There is still a lot that is unknown about these algorithms such as guaranteed rates of convergence. It would be interesting to see a more in-depth analysis of these types of algorithms and if there are better ways to solve the lasso problem and box-constrained QP during the block updates. While these implementations can already handle a problem of $p = 1000$ dimensions within a reasonable amount of time, it would be nice to have faster implementations that scale better to allow larger problems to be analyzed such as the entire original gene expression data set. At the moment this seems out of reach however without serious high-performance computing, but it will be interesting to see how these algorithms develop, both in terms of theory and scalability, in the future.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009.

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

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. 2012.

[6] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151:3–34, 2015.

[7] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *Journal of Machine Learning Research*, 9:485–516, 2008.

[8] R. Mazumder and T. Hastie. The graphical lasso: New insights and alternatives. *Electronic Journal of Statistics*, 6:2125–2149, 2012.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[10] Yuxin Chen. Large-scale optimization for data science: Alternating direction method of multipliers ELE 522 lecture notes, 2019.

[11] M. J. Wainwright. *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge University Press, 2019.

[12] UCI Machine Learning Repository. gene expression cancer RNA-Seq data set, 2016.

```
function [x, fvals] = lasso(A, b, r, x0, maxiter, tol, ret_obj)
% lasso Solve the lasso problem using coordinate descent.
%
%   min 0.5*x'Ax + b'x + r|x|_1]
%
% where A is a positive definite p-by-p matrix.
% x0, b are p-by-1 vectors and r > 0 is the regularization.
% Returns the optimal solution x.
% Returns the function values fvals if ret_obj is true.
%
% Author: Terrence Alsup
% Date: May 18, 2020
% File: lasso.m

% By default do not return the objective function values.
if nargin < 7
    ret_obj = false;
end

% Get the dimension of the problem.
p = size(x0, 1);

% Set the starting value.
x = x0;

% Return the objective function values after each cycle.
if ret_obj
    obj = @(x) 0.5*x'*A*x + b'*x + r*norm(x, 1);
    fvals(1) = obj(x);
    fprev = fvals(1);
else
    xprev = x;
end

% Set the initial difference to ensure at least 1 iteration.
rel_diff = 2*tol + 1;

k = 0; % Track the iteration.
while k < maxiter && rel_diff > tol

    % The coordinate to update.
    i = mod(k, p) + 1;
        % Perform the update to coordinate i.
        temp = (-b(i) - A(i,:)*x + A(i,i)*x(i));
        % Soft thresholding.
        x(i) = sign(temp)*max(abs(temp) - r, 0)/A(i,i);
```

```
    % Increment the iteration.
    k = k + 1;

    % Compute the relative difference for the stopping criteria after
    % every cycle over all coordinates.
    if ret_obj
        % Use relative change in objective function value.
        fvals(k + 1) = obj(x);
        if i == p
            rel_diff = (fprev - fvals(k+1))/abs(fprev);
            fprev = fvals(k + 1);
        end
    else
        % Use relative change in solution.
        if i == p
            rel_diff = norm(x - xprev)/norm(xprev);
            xprev = x;
        end
    end
end
end
```

```
function [x, fvals] = QP_box(Q, b, l, u, x0, maxiter, tol, ret_obj)
% QP_box Solve the QP with box constraints using coordinate descent.
%
% min 0.5*x'Qx + b'x
% s.t. l <= x <= u
%
% where Q is a positive definite p-by-p matrix.
% x0,b,l,u are column vectors of dimension p-by-1.
% Returns the optimals solution x.
% Returns the function values fvals if ret_obj is true.
%
% Author: Terrence Alsup
% Date: May 18, 2020
% File: QP_box.m


% By default do not return the objective function values.
if nargin < 8
    ret_obj = false;
end

% Get the dimension of the problem.
p = size(x0, 1);

% Set the starting value.
x = x0;

% Return the objective function values after each cycle.
if ret_obj
    obj = @(x) 0.5*x'*Q*x + b'*x;
    fvals(1) = obj(x);
    fprev = fvals(1);
else
    xprev = x;
end

% Set the initial difference to ensure at least 1 iteration.
rel_diff = 2*tol + 1;

k = 0; % Track the iteration.
while k < maxiter && rel_diff > tol

    % Get the coordinate to update.
    i = mod(k, p) + 1;
        % Perform update on coordinate i.
    temp = (Q(i,i)*x(i) - b(i) - Q(i,:)*x)/Q(i,i);
```

```
        % Threshold to meet box constraints.
        x(i) = max(min(temp, u(i)), l(i));

        % Increment the iteration
        k = k + 1;
        % Compute the relative difference for the stopping criteria after
        % every cycle over all coordinates.
        if ret_obj
            % Use relative change in objective function value.
            fvals(k + 1) = obj(x);
            if i == p
                rel_diff = (fprev - fvals(k+1))/abs(fprev);
                fprev = fvals(k + 1);
            end
        else
            % Use relative change in solution.
            if i == p
                rel_diff = norm(x - xprev)/norm(xprev);
                xprev = x;
            end
        end
    end
end
```

```
function [X, W, fvals, dvals] = glasso(S, r, maxiter, tol, ret_obj)
% glasso Solve the graphical lasso problem using the glasso algorithm.
%
% S is the p-by-p sample covariance matrix.
% r > 0 is the regularization parameter.
% Returns precision matrix X and covariance matrix W.
% Returns primal and dual objective function values if ret_obj is true.
%
% Author: Terrence Alsup
% Date: May 18, 2020
% File: glasso.m


% By default do not return objective function values.
if nargin < 5
    ret_obj = false;
end

% Get the dimension of the problem.
p = size(S, 1);

% Initialize the covariance and precision matrices.
W = S + r*eye(p);
X = speye(p); % Do not need to start with X^-1 = W.

% Return the objective function values after each iteration.
if ret_obj
    % Define the primal objective function.
    obj = @(X) -log(det(X)) + trace(X*S) + r*sum(abs(X), 'all');
    dobj = @(W) log(det(W)) + p;

    % Initial point is guaranteed to be postive definite (i.e. feasible).
    fvals(1) = obj(X);
    dvals(1) = dobj(W);
    fprev = fvals(1);
else
    % Track the covariance matrix.
    Wprev = W;
end

% Set the initial difference to ensure at least 1 iteration.
rel_diff = 2*tol + 1;

k = 0; % Track the iteration.
while k < maxiter && rel_diff > tol
```

```
% Get the coordinate to update.
i = mod(k, p) + 1;

% Get the indices for the block submatrix.
idx = cat(2, 1:i-1, i+1:p);

W11 = W(idx, idx);
w12 = W(idx, i);
w22 = W(i, i);
s12 = S(idx, i);
x12 = X(idx, i);
x22 = X(i, i);

% Solve the lasso problem with coordinate descent.
% Starting point is previous optimal value of b.
b = lasso(W11, s12, r, x12/x22, 100*p, 1e-6);

% Update the covariance matrix.
w12 = -W11*b;
W(idx, i) = w12;
W(i, idx) = w12';

% Update the precision matrix.
x22 = 1/(w22 + b'*w12);
x12 = x22*b;
X(i, i) = x22;
X(idx, i) = x12;
X(i, idx) = x12';


% Increment the iteration.
k = k + 1;

% Compute the relative difference for the stopping criteria after
% every cycle over all coordinates.
if ret_obj
    % Use relative change in objective function value.
    % Note that we use inv(W) instead of X because WX != I.
    fvals(k + 1) = obj(X);
    dvals(k + 1) = dobj(W);
    if i == p
        rel_diff = (fprev - fvals(k+1))/abs(fprev);
        fprev = fvals(k + 1);
    end
else
    if i == p
```

```
            % Use relative change in covariance matrix.
            rel_diff = norm(W - Wprev, 'Fro')/norm(Wprev, 'Fro');
            Wprev = W;
        end
    end
end
end
```

```
function [X, W, fvals, dvals] = dpglasso(S, r, maxiter, tol, ret_obj)
% dpglasso Solve the graphical lasso problem using the DP-glasso algorithm.
%
% S is the p-by-p sample covariance matrix.
% r > 0 is the regularization parameter.
% Returns precision matrix X and covariance matrix W.
% Returns primal and dual objective function values if ret_obj is true.
%
% Author: Terrence Alsup
% Date: May 18, 2020
% File: dpglasso.m

% By default do not return objective function values.
if nargin < 5
    ret_obj = false;
end

% Get the dimension of the problem.
p = size(S, 1);

% Initialize the precision and covariance matrices.
dia = diag(S) + r;
X = spdiags(1./dia, 0, p, p);
W = diag(dia); % Covariance estimate will not be sparse in general.

% Dual feasible point.
%W = S + r*eye(p);
%X = inv(W);

% Return the objective function values after each iteration.
if ret_obj
    % Define the primal objective function.
    obj = @(X) -log(det(X)) + trace(X*S) + r*sum(abs(X), 'all');
    dobj = @(W) log(det(W)) + p;

    % Initial point is guaranteed to be postive definite (i.e. feasible).
    fvals(1) = obj(X);
    dvals(1) = dobj(W);
    fprev = fvals(1);
else
    % Track the precision matrix.
    Xprev = X;
end

% Set the initial difference to ensure at least 1 iteration.
rel_diff = 2*tol + 1;
```

```
k = 0; % Track the iteration.
while k < maxiter && rel_diff > tol

    % Get the coordinate to update.
    i = mod(k, p) + 1;

    % Get the indices for the block submatrix.
        idx = cat(2, 1:i-1, i+1:p);

        X11 = X(idx, idx);
        x12 = X(idx, i);
        s12 = S(idx, i);

        % Solve the QP with box constraints.  Do a maximum of 100 cycles.
        % Initial point is previous optimal value.
        cons = r*ones(p - 1, 1);
        dy = QP_box(X11, X11*s12, -cons, cons, x12, 100*p, 1e-6);

        % Update the covariance matrix.
        w12 = s12 + dy;
       W(idx, i) = w12;
   W(i, idx) = w12';

        % Update the precision matrix.
        % Use thresholding to ensure that the update will be sparse.
        x12 = -X11*w12/W(i,i) .* (r - abs(dy) < 1e-6);
        x22 = (1 - x12'*w12)/W(i,i);
        X(idx, i) = x12;
        X(i, idx) = x12';
        X(i, i) = x22;

    % Increment the iteration.
    k = k + 1;

    % Compute the relative difference for the stopping criteria after
    % every cycle over all coordinates.
    if ret_obj
        % Use relative change in objective function value.
        fvals(k + 1) = obj(X);
        dvals(k + 1) = dobj(W);
        if i == p
            rel_diff = (fprev - fvals(k+1))/abs(fprev);
            fprev = fvals(k + 1);
        end
    else
```

```
        % Use relative change in precision matrix.
        if i == p
            rel_diff = norm(X - Xprev, 'Fro')/norm(Xprev, 'Fro');
            Xprev = X;
        end
    end
end
end
```

```
function [X, W, fvals, dvals] = glasso_admm(S, r, mu, maxiter, tol, ret_obj)
% glasso_admm Solve the graphical lasso problem using ADMM.
%
% S is the p-by-p sample covariance matrix.
% r,mu > 0 are the regularization and penality parameters.
% Returns precision matrix X and covariance matrix W.
% Returns primal and dual objective function values if ret_obj is true.
%
% Author: Terrence Alsup
% Date: May 18, 2020
% File: glasso_admm.m

% By default do not return objective function values.
if nargin < 6
    ret_obj = false;
end

% Get the dimension of the problem.
p = size(S, 1);

% Initialization.
X = eye(p); % Precision matrix.
Z = eye(p); % Z variable from ADMM
Y = eye(p); % Dual variable

% Return the objective function values after each iteration.
if ret_obj
    % Define the primal objective function.
    obj = @(X) -log(det(X)) + trace(X*S) + r*sum(abs(X), 'all');
    dobj = @(Y) log(det(S + Y)) + p;

    % Initial point is guaranteed to be postive definite (i.e. feasible).
    fvals(1) = obj(X);
    dvals(1) = dobj(Y);
    fprev = fvals(1);
else
    Xprev = X;
end

% Set the initial difference to ensure at least 1 iteration.
rel_diff = 2*tol + 1;

k = 0; % Track the iteration.
while k < maxiter && rel_diff > tol

    % Update precision matrix X.
```

```
    [V, D] = eig(Z - Y/mu - S/mu);
    d = diag(D);
    D = diag(0.5*(d + sqrt(d.^2 + 4/mu)));
    X = V*D*V';

    % Update Z variable with soft thresholding.
    Z = sign(X + Y/mu).*(max(abs(X + Y/mu) - r/mu, 0));

    % Update dual variable X.
    Y = Y + mu*(X - Z);

    % Increment the iteration.
    k = k + 1;

    % Compute the relative difference for the stopping criteria after
    % every iteration.
    if ret_obj
        % Use relative change in objective function value.
        fvals(k + 1) = obj(X);
        dvals(k + 1) = dobj(Y);
        rel_diff = abs(fprev - fvals(k+1))/abs(fprev);
        fprev = fvals(k + 1);
    else
        % Use relative change in precision matrix.
        rel_diff = norm(X - Xprev, 'Fro')/norm(Xprev, 'Fro');
        Xprev = X;
    end
end

% Recover the covariance matrix from the dual variable.
W = S + Y;

end
```