Large Scale Optimization for Machine Learning:
Lecture 22

Lecturer: Meisam Razaviyayn Scribe: Max Gaungyu Li

Nov 14, 2017

1 Recap: BCD Method

When we use BCD Method to solve the multi-block structure problem, there will be four main drawbacks:

1. Analysis of differentiable function or relaxed assumption of regular function.
2. Separable constraints.
3. Uniqueness of Minimizer.
4. Computation of an exact minimizer for the sub-problem at each iteration.

Since the multi-block structure requires separable constraints, like we have discussed before, it is necessary. Let's take a simple example to show the necessity here. Suppose we want to minimize $x_1^2+x_2^2$, subject to $x_1+x_2 = 0$. If we start at the point $x_1 = 1, x_2 = -1$. then it will get stuck and won't change its value using BCD Method. Thus, we can't have coupled constraints. But if we use ADMM, we can handle some simple coupled constraints, like linear coupled constraints our example shown above.
Before we go further on ADMM, we need to take a look at this simple case.

2 ADMM

2.1 Simple Case: One-Block

Consider this simple case of one-block problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ s.t. \quad & \mathbf{Ax} = \mathbf{b} \end{aligned} \tag{1}$$

Can we use gradient method on the dual function to solve this problem? Yes.

- Lagrangian function: $L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \langle \boldsymbol{\lambda}, \mathbf{Ax\text{-}b} \rangle$
- Dual function: Assume we know the existence of minimizer, then the infimum of L is equivalent to minimum of L, thus, $g(\boldsymbol{\lambda}) = \min_{\mathbf{x}} \quad f(\mathbf{x}) + \langle \boldsymbol{\lambda}, \mathbf{Ax\text{-}b} \rangle$
- $\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \quad g(\boldsymbol{\lambda})$

Algorithm 1 Gradient descent method implemented to dual function

1: **procedure** UPDATE $\boldsymbol{\lambda}$
2: *loop*: $r = 1, 2, 3, ...$
3: $\quad \boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \alpha \nabla g(\boldsymbol{\lambda}^r)$.
4: *end loop*
5: get $\boldsymbol{\lambda}^*$.
6: **procedure** UPDATE $\mathbf{x}$
7: *loop*: $r = 1, 2, 3, ...$
8: $\quad \mathbf{x}^{r+1} \leftarrow \mathbf{x}^r - \beta \nabla L(\mathbf{x}^r, \boldsymbol{\lambda}^*)$.
9: *end loop*
10: get $\mathbf{x}^*$.

It seems to be good, but there are some other issues when we implement this method. Firstly, we don't know if $g(\boldsymbol{\lambda}^r)$ has closed form, actually in many cases, there is no close form. Secondly, there is no guarantee that $g(\boldsymbol{\lambda}^r)$ is differentiable. So we can't compute gradient of $g(\boldsymbol{\lambda}^r)$. But some might say, we can compute the subgradient of $g(\boldsymbol{\lambda}^r)$ instead. Yes, that is true. Actually, we don't need to know $g(\boldsymbol{\lambda}^r)$ and we can compute the gradient or the subgradient of $g(\boldsymbol{\lambda}^r)$. Let's introduce Danskin's Theorem first.

Theorem 1 ***Danskin's Theorem*** *[1]*
Suppose $\phi(x, z)$ is a continuous function of two arguments,

$\phi : \mathbb{R}^n \times Z \to \mathbb{R}$

where $Z \subset \mathbb{R}^m$ is a compact set. Further assume that $\phi(x, z)$ is convex in x for every $z \in Z$.

Under these conditions, Danskin's theorem provides conclusions regarding the differentiability of the function $f(x) = \max_{z \in Z} \phi(x, z)$.

To state these results, we define the set of maximizing points $Z_0(x)$ as $Z_0(x) = \{\overline{z} : \phi(x, \overline{z}) = \max_{z \in Z} \phi(x, z)\}$. *Danskin's theorem then provides the following results.*

Convexity *$f(x)$ is convex.*

Directional derivatives *The directional derivative of $f(x)$ in the direction y, denoted $D_y f(x)$, is given by $D_y f(x) = \max_{z \in Z_0(x)} \phi'(x, z; y)$, where $\phi'(x, z; y)$ is the directional derivative of the function $\phi(\cdot, z)$ at x in the direction y.*

Derivative *$f(x)$ is differentiable at x if $Z_0(x)$ consists of a single element $\overline{z}$. In this case, the derivative of $f(x)$ (or the gradient of $f(x)$ if x is a vector) is given by* $\dfrac{\partial f}{\partial x} = \dfrac{\partial \phi(x, \overline{z})}{\partial x}$.

To put this theorem in simple words, let's get back to the one-block problem for example. If $\mathbf{x}$ is fixed, then the Lagrangian function, $L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \langle \boldsymbol{\lambda}, \mathbf{Ax\text{-}b} \rangle$ is linear on $\boldsymbol{\lambda}$. Since $g(\boldsymbol{\lambda}) = \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda})$ if we assume the existence of a unique minimizer, then we can have $\nabla g(\boldsymbol{\lambda}) = \mathbf{A}\overline{\mathbf{x}} - \mathbf{b}$, where $\overline{\mathbf{x}}$ is the minimizer $\overline{\mathbf{x}} = \arg\min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda})$. This is because $\nabla g(\boldsymbol{\lambda})$ can be viewed as the minimum of (infinitely many)$L(\mathbf{x}, \boldsymbol{\lambda})$ varied on any arbitrary $\mathbf{x}$, Danskin's Theorem states that $\nabla g(\boldsymbol{\lambda})$ is exactly the smallest single function of L on all arbitrary $\mathbf{x}$.

Thus, we derive the dual ascent algorithm below:

Algorithm 2 Dual Ascent Algorithm

1: **procedure** UPDATE $\mathbf{x}, \boldsymbol{\lambda}$
2: *loop*: $r = 1, 2, 3, ...$
3: $\quad \mathbf{x}^{r+1} \leftarrow \arg\min_{\mathbf{x}} \quad L(\mathbf{x}, \boldsymbol{\lambda}^r)$.
4: $\quad \boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \alpha(\mathbf{A}\mathbf{x}^{r+1} - \mathbf{b})$.
5: *end loop*
6: get $\mathbf{x}^*, \boldsymbol{\lambda}^*$.

In this Dual Ascent Algorithm, given a fixed $\boldsymbol{\lambda}^r$, we can find a minimizer $\mathbf{x}^{r+1}$, then we can get the gradient ascent direction if we use this minimizer $\mathbf{x}^{r+1}$ in this iteration and get a $\boldsymbol{\lambda}^{r+1}$.

Though this Dual Ascent Algorithm is popular, there are some drawbacks about this algorithm. Firstly, it has a strong assumption requirement. It required to have a compact set, differentiable and strong convex function, and need unique minimizer.. Moreover, we don't know how to choose a proper step size in this algorithm because we don't have a lipschitz constant here, and the convergence might be slow.

But this dual ascent algorithm also have some benefits, if we can separate the function into sum of multiple functions and also the constraints can be separable, then we can solve this problem in parallel. This is multi-block separable case.

2.2 Multi-block Separable Case

Consider this Multi-block Separable problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_1(\mathbf{x}_1) + ... + f_m(\mathbf{x}_m) \\ s.t. \quad & \mathbf{A}_1\mathbf{x}_1 + ... + \mathbf{A}_m\mathbf{x}_m = \mathbf{b} \end{aligned} \tag{2}$$

Algorithm 3 Dual Decomposition Algorithm

1: **procedure** UPDATE $\mathbf{x}_i, \boldsymbol{\lambda}$
2: *loop*: $r = 1, 2, 3, ...$
3: $\quad \mathbf{x}_i^{r+1} \leftarrow \arg\min_{\mathbf{x}_i} \sum_{i=1}^m f_i(\mathbf{x}_i) + \boldsymbol{\lambda}^{rT}(\sum_{i=1}^m \mathbf{A}_i\mathbf{x}_i - \mathbf{b})$, $i = 1, ..., m$.
4: $\quad \boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \alpha \sum_{i=1}^m \mathbf{A}_i\mathbf{x}_i^{r+1} - \mathbf{b}$.
5: *end loop*
6: get $\mathbf{x}^*, \boldsymbol{\lambda}^*$.

The property of Dual Decomposition Algorithm is the same as dual ascent algorithm.

2.3 Method of Multipliers

Consider the one-block problem 1, if we add a penalty term $\frac{\rho}{2}\|\mathbf{Ax} - \mathbf{b}\|^2$ in L, we can have an augmented Lagrangian Function: $L_\rho(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \langle \boldsymbol{\lambda}, \mathbf{Ax\text{-}b} \rangle + \frac{\rho}{2}\|\mathbf{Ax} - \mathbf{b}\|^2$ where ρ is some positive constant. If we use this L_ρ

to solve the problem 1 like Algorithm 1, we will get this new method, called Method of Multipliers:

Algorithm 4 Method of Multipliers

1: **procedure** UPDATE $\mathbf{x}, \boldsymbol{\lambda}$
2: *loop*: $r = 1, 2, 3, ...$
3: $\quad \mathbf{x}^{r+1} \leftarrow \arg\min_{\mathbf{x}} \quad L_\rho(\mathbf{x}, \boldsymbol{\lambda}^r)$.
4: $\quad \boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \rho(\mathbf{A}\mathbf{x}^{r+1} - \mathbf{b})$.
5: *end loop*
6: get $\mathbf{x}^*, \boldsymbol{\lambda}^*$.

this Method of Multipliers is robust and suitable for more relaxed conditions. It doesn't require the objective function to be smooth. But the drawback is that it can't be decomposed anymore.

Why do we choose α as our step-size?

Becuase in the algorithm 1, we have $\mathbf{x}^* = \arg\min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}^*)$,
which is $\mathbf{x}^* = \arg\min_{\mathbf{x}} \quad f(\mathbf{x}) + \langle \boldsymbol{\lambda}^*, \mathbf{A}\mathbf{x}\text{-}\mathbf{b} \rangle$. By using KKT condition, we can have $\nabla f(\mathbf{x}^*) + \mathbf{A}^T \boldsymbol{\lambda}^* = 0$.
If we want this equation still hold true for Method of Multipliers at each iteration, we must satisfy some condition in the new iteration.
In this Method of Multipliers Algorithm, we have $\mathbf{x}^{r+1} = \arg\min_{\mathbf{x}} \quad L_\rho(\mathbf{x}, \boldsymbol{\lambda}^r)$,
which is $\mathbf{x}^* = \arg\min_{\mathbf{x}} \quad f(\mathbf{x}) + \langle \boldsymbol{\lambda}^*, \mathbf{A}\mathbf{x}\text{-}\mathbf{b} \rangle + \frac{\rho}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$.
By using KKT condition, we can have $\nabla f(\mathbf{x}^*) + \mathbf{A}^T[\boldsymbol{\lambda}^* + \rho(\mathbf{A}\mathbf{x}\text{-}\mathbf{b})] = 0$.
Thus, compared the difference of these two algorithms' KKT conditions, we found that if we can update $\boldsymbol{\lambda}$ by $\boldsymbol{\lambda}^{r+1} = \boldsymbol{\lambda}^r + \rho(\mathbf{A}\mathbf{x}^{r+1} - \mathbf{b})$, they would be the same and hold true. That's why in the update of $\boldsymbol{\lambda}$, we use ρ as the step size.

Is this method also extensible to multi-block problem?
Yes, that will be Alternating Direction Method of Multipliers(ADMM).

2.4 Alternating Direction Method of Multipliers(ADMM)

Consider the multi-block problem, if we think of a similar idea used for dual decomposed algorithms derived from dual ascent algorithm, we can also derive this ADMM from Method of Multipliers Algorithm.

Algorithm 5 ADMM with cyclic update rule

procedure UPDATE $\mathbf{x}, \boldsymbol{\lambda}$
loop: $r = 1, 2, 3, ...$
  $\mathbf{x}_i^{r+1} \leftarrow \arg\min_{\mathbf{x}_i} \quad L_\rho(\mathbf{x}_1^{r+1}, ..., \mathbf{x}_{i-1}^{r+1}, \mathbf{x}_i, \mathbf{x}_{i+1}^{r+1}, ..., \mathbf{x}_m^{r+1}; \boldsymbol{\lambda}^r), \quad \forall i.$
  $\boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \rho(\mathbf{A}\mathbf{x}^{r+1} - \mathbf{b}).$
end loop
get $\mathbf{x}^*, \boldsymbol{\lambda}^*$.

It should be noticed that in the update of $\mathbf{x}_i$, we use a cyclic rule of updating order here. But we can also use randomized rule. The understanding of the above multi-block ADMM is still very limited. Therefore, here we focus on the simple two block scenario:

Consider the two block scenario of ADMM, i.e. there are only two block in the multi-block problem.

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & f(\mathbf{x}) + g(\mathbf{z}) \\ s.t. \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{b} \end{aligned} \tag{3}$$

We can write down its augmented Lagrangian Function: $L_\rho(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}) = f(\mathbf{x}) + g(\mathbf{z}) + \langle \boldsymbol{\lambda}, \mathbf{Ax+Bz-b} \rangle + \frac{\rho}{2} \|\mathbf{Ax+Bz} - \mathbf{b}\|^2$. Then we can derive Algorithm 6 from Algorithm 5.

Algorithm 6 ADMM, two block scenario

procedure UPDATE **x,z**, $\boldsymbol{\lambda}$
loop: $r = 1, 2, 3, ...$
  $\mathbf{x}^{r+1} \leftarrow \arg\min_{\mathbf{x}} \quad L_\rho(\mathbf{x}, \mathbf{z}^r, \boldsymbol{\lambda}^r).$
  $\mathbf{z}^{r+1} \leftarrow \arg\min_{\mathbf{z}} \quad L_\rho(\mathbf{x}^{r+1}, \mathbf{z}, \boldsymbol{\lambda}^r).$
  $\boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \rho(\mathbf{A}\mathbf{x}^{r+1} + \mathbf{B}\mathbf{z}^{r+1} - \mathbf{b}).$
end loop
get $\mathbf{x}^*, \mathbf{z}^*, \boldsymbol{\lambda}^*$.

ADMM is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which are then easier to handle. It has recently found wide application in a number of areas. In this lecture, we provide a few examples to interesting applications and implementations of the method.

2.5 Examples

2.5.1 Generic Convex Problem

Consider this general minimize problem with constrants:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ s.t. \quad & \mathbf{x} \in \chi \end{aligned} \tag{4}$$

We can rewrite it into this form and solve it by ADMM.

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) + \mathcal{I}_{\chi}(\mathbf{z}) \\ s.t. \quad & \mathbf{x} = \mathbf{z} \end{aligned} \tag{5}$$

We can write down its augmented Lagrangian Function: $L_\rho(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \mathcal{I}_{\chi}(\mathbf{z}) + \langle \boldsymbol{\lambda}, \mathbf{x}\text{-}\mathbf{z} \rangle + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}\|^2$.

$\mathbf{x}^{r+1} = \arg\min_{\mathbf{x}} f(\mathbf{x}) + \mathcal{I}_{\chi}(\mathbf{z}^r) + \langle \boldsymbol{\lambda}^r, \mathbf{x} - \mathbf{z}^r \rangle + \frac{\rho}{2}\|\mathbf{x}\text{-}\mathbf{z}^r\|^2$.

$\Rightarrow \mathbf{x}^{r+1} = \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{x}\text{-}\mathbf{z}^r + \rho^{-1}\boldsymbol{\lambda}^r\|^2$.

$\mathbf{z}^{r+1} = \arg\min_{\mathbf{z}} f(\mathbf{x}^{r+1}) + \mathcal{I}_{\chi}(\mathbf{z}) + \langle \boldsymbol{\lambda}^r, \mathbf{x}^{r+1} - \mathbf{z} \rangle + \frac{\rho}{2}\|\mathbf{x}^{r+1} - \mathbf{z}\|^2$.

$\Rightarrow \mathbf{z}^{r+1} = \arg\min_{\mathbf{z}\in\chi} \frac{\rho}{2}\|\mathbf{x}^{r+1} - \mathbf{z} + \rho^{-1}\boldsymbol{\lambda}^r\|^2$.

So $\mathbf{z}^{r+1} = \arg\min_{\mathbf{z}\in\chi} \frac{\rho}{2}\|\mathbf{z} - (\mathbf{x}^{r+1} + \rho^{-1}\boldsymbol{\lambda}^r)\|^2$.

$\Rightarrow$ Which is to say, $\mathbf{z}^{r+1}$ is the projection of $[\mathbf{x}^{r+1} + \rho^{-1}\boldsymbol{\lambda}^r]$. Thus, we get Algorithm 7 below.

Algorithm 7 ADMM applied to Generic Convex Problem

1: **procedure** UPDATE **x,z,λ**
2: *loop*: $r = 1, 2, 3, ...$
3: $\quad \mathbf{x}^{r+1} \leftarrow \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{x}\text{-}\mathbf{z}^r + \rho^{-1}\boldsymbol{\lambda}^r\|^2$.
4: $\quad \mathbf{z}^{r+1} \leftarrow [\mathbf{x}^{r+1} + \rho^{-1}\boldsymbol{\lambda}^r]_+$.
5: $\quad \boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \rho(\mathbf{x}^{r+1} - \mathbf{z}^{r+1})$.
6: *end loop*
7: get $\mathbf{x}^*, \mathbf{z}^*, \boldsymbol{\lambda}^*$.

2.5.2 Lasso Problem

Consider this lasso problem:

$$\min_{\mathbf{x}} \quad \frac{1}{2}\|\mathbf{Ax\text{-}b}\|_2^2 + \mu\|\mathbf{x}\|_1 \tag{6}$$

We can rewrite it into this form and solve it by ADMM.

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\|\mathbf{Ax\text{-}b}\|_2^2 + \mu\|\mathbf{z}\|_1 \\ s.t. \quad & \mathbf{x} = \mathbf{z} \end{aligned} \tag{7}$$

$$\begin{aligned} & \mathbf{x}^{r+1} = \arg\min_{\mathbf{x}} \quad L_\rho(\mathbf{x}, \mathbf{z}^r, \boldsymbol{\lambda}^r). \\ \Rightarrow\ & \mathbf{x}^{r+1} = \arg\min_{\mathbf{x}} \quad \tfrac{1}{2}\|\mathbf{Ax\text{-}b}\|_2^2 + \mu\|\mathbf{z}^r\|_1 + \langle \boldsymbol{\lambda}^r, \mathbf{x} - \mathbf{z}^r\rangle + \tfrac{\rho}{2}\|\mathbf{x} - \mathbf{z}^r\|_2^2. \end{aligned}$$

$$\Rightarrow \mathbf{A}^T(\mathbf{A}\mathbf{x}^{r+1} - \mathbf{b}) + \boldsymbol{\lambda}^r + \rho(\mathbf{x}^{r+1} - \mathbf{z}^r) = 0.$$

$$\Rightarrow \mathbf{x}^{r+1} = (\mathbf{A}^T\mathbf{A} + \rho\mathbf{I})^{-1}(\mathbf{A}^T\mathbf{b} + \rho\mathbf{z}^r - \boldsymbol{\lambda}^r).$$

$$\begin{aligned} & \mathbf{z}^{r+1} = \arg\min_{\mathbf{z}} \quad L_\rho(\mathbf{x}^{r+1}, \mathbf{z}, \boldsymbol{\lambda}^r). \\ \Rightarrow\ & \mathbf{z}^{r+1} = \arg\min_{\mathbf{z}} \quad \tfrac{1}{2}\|\mathbf{A}\mathbf{x}^{r+1} - \mathbf{b}\|_2^2 + \mu\|\mathbf{z}\|_1 + \langle \boldsymbol{\lambda}^r, \mathbf{x}^{r+1} - \mathbf{z}\rangle + \tfrac{\rho}{2}\|\mathbf{x}^{r+1} - \mathbf{z}\|_2^2. \end{aligned}$$

$$\Rightarrow \mathbf{z}^{r+1} = \arg\min_{\mathbf{z}} \quad \mu\|\mathbf{z}\|_1 + \tfrac{\rho}{2}\|\mathbf{x}^{r+1} - \mathbf{z} + \rho^{-1}\boldsymbol{\lambda}^r\|_2^2.$$

$$\Rightarrow \mathbf{z}^{r+1} = \arg\min_{\mathbf{z}} \quad \mu\rho^{-1}\|\mathbf{z}\|_1 + \|\mathbf{z} - (\mathbf{x}^{r+1} + \rho^{-1}\boldsymbol{\lambda}^r)\|_2^2.$$

$$\Rightarrow \mathbf{z}^{r+1} = \mathcal{S}_{\mu\rho^{-1}}(\mathbf{x}^{r+1} + \rho^{-1}\boldsymbol{\lambda}^r).$$

$\Rightarrow$ Which is to say, $\mathbf{z}^{r+1}$ is the shrinkage function of $(\mathbf{x}^{r+1} + \rho^{-1}\boldsymbol{\lambda}^r)$ over $\mu\rho^{-1}$. Thus, we get Algorithm 8 below.

2.5.3 Sparse Inverse Covariance Estimation

Recall in Gaussian graphical models, sparsity in inverse convariance matrix (a.k.a precision matrix) represent conditional independence among random variables. Specifically, $\boldsymbol{\Sigma}^{-1}_{ij} = 0$ is equivelent to $\boldsymbol{x}_i \perp \boldsymbol{x}_j | \boldsymbol{x}_{k\neq i,j}$. Consider the following optimization problem

$$\min_{\mathbf{X}} \quad \mathrm{Tr}(\hat{\boldsymbol{\Sigma}}\boldsymbol{X}) - \log\det(\boldsymbol{X}) \tag{8}$$

Algorithm 8 ADMM applied to Lasso Problem

procedure UPDATE $\mathbf{x}$,$\mathbf{z}$,$\boldsymbol{\lambda}$
loop: $r = 1, 2, 3, ...$
  $\mathbf{x}^{r+1} \leftarrow (\mathbf{A}^T\mathbf{A} + \rho\mathbf{I})^{-1}(\mathbf{A}^T\mathbf{b} + \rho\mathbf{z}^r - \boldsymbol{\lambda}^r)$.
  $\mathbf{z}^{r+1} \leftarrow \mathcal{S}_{\mu\rho^{-1}}(\mathbf{x}^{r+1} + \rho^{-1}\boldsymbol{\lambda}^r)$.
  $\boldsymbol{\lambda}^{r+1} \leftarrow \boldsymbol{\lambda}^r + \rho(\mathbf{x}^{r+1} - \mathbf{z}^{r+1})$.
end loop
get $\mathbf{x}^*, \mathbf{z}^*, \boldsymbol{\lambda}^*$.

A clear minimizer is $\boldsymbol{X}^* = \hat{\boldsymbol{\Sigma}}^{-1}$, which is the inverse covariance matrix. To introduce sparsity, we add $L1$ penalty on $\boldsymbol{X}$ which also refers to Graphical Lasso.

$$\begin{aligned} \min_{\mathbf{X}} \quad & \mathrm{Tr}(\hat{\boldsymbol{\Sigma}}\boldsymbol{X}) - \log\det(\boldsymbol{X}) + \mu||\boldsymbol{X}||_1 \\ \text{s.t.} \quad & X_{ij} = 0, \quad \forall (i,j) \in \mathcal{C} \end{aligned} \tag{9}$$

Note that above constraint are certain assumption or prior knowledge we have regarding conditional indenpendence. Rewrite it into the following form and solve it by ADMM.

$$\begin{aligned} \min_{\mathbf{X},\boldsymbol{Z}} \quad & \mathrm{Tr}(\hat{\boldsymbol{\Sigma}}\boldsymbol{X}) - \log\det(\boldsymbol{X}) + \mu||\boldsymbol{Z}||_1 + \mathcal{I}_{\mathcal{X}}(\boldsymbol{Z}) \\ \text{s.t.} \quad & \boldsymbol{X} = \boldsymbol{Z} \end{aligned} \tag{10}$$

Following are update rules with ADMM:

$$\begin{aligned} \boldsymbol{X}^{r+1} &= \arg\min_{\boldsymbol{X}} \mathrm{Tr}(\hat{\boldsymbol{\Sigma}}\boldsymbol{X}) - \log\det(\boldsymbol{X}) + \langle \boldsymbol{\Lambda}^r, \boldsymbol{X} - \boldsymbol{Z}^r \rangle + (\rho/2)||\boldsymbol{X} - \boldsymbol{Z}^r||_F^2 \\ \boldsymbol{Z}^{r+1} &= \arg\min_{\boldsymbol{Z}} \mu||\boldsymbol{Z}||_1 + \mathcal{I}_{\mathcal{X}}(Z) - \langle \boldsymbol{\Lambda}^r, \boldsymbol{Z} \rangle + (\rho/2)||\boldsymbol{X}^{r+1} - \boldsymbol{Z}||_F^2 \\ \boldsymbol{\Lambda}^{r+1} &= \boldsymbol{\Lambda}^r + \rho(\boldsymbol{X}^{r+1} - \boldsymbol{Z}^{r+1}) \end{aligned} \tag{11}$$

The update rule of $\boldsymbol{X}$ can be simplified further by setting the derivative to 0.

$$\begin{aligned} & \hat{\boldsymbol{\Sigma}} - \boldsymbol{X}^{-1} + \rho(\boldsymbol{X} - \boldsymbol{Z}^r + \boldsymbol{\Lambda}^r) = 0 \\ & \Rightarrow \rho\boldsymbol{X} - \boldsymbol{X}^{-1} = \rho(\boldsymbol{Z}^r - \boldsymbol{\Lambda}^r) - \hat{\boldsymbol{\Sigma}} \end{aligned} \tag{12}$$

One can verify that a close form solution can be achieved by taking eigen decomosition of $\rho(\boldsymbol{Z}^r - \boldsymbol{\Lambda}^r) - \hat{\boldsymbol{\Sigma}}$. The update rule of $\boldsymbol{Z}$ also has close form solution by taking element-wise shrinkage on $\boldsymbol{X}^{r+1} + \boldsymbol{\Lambda}^r$

References

[1] D. P. Bertsekas "Nonlinear Programming," *Belmont: Athena scientific*, 1999.