Conjugate Gradient and Constrained optimization

1 Conjugate gradient

Suppose we want to solve the system of linear equations

$$\mathbf{Ax} = \mathbf{b}$$

for the vector $\mathbf{x}$, where the known $n \times n$ matrix $\mathbf{A}$ is symmetric (i.e., $\mathbf{A}^T = \mathbf{A}$), positive-definite (i.e. $\mathbf{x}^T\mathbf{Ax} > 0$ for all non-zero vectors $\mathbf{x} \in \mathbb{R}^n$), and real, and $\mathbf{b}$ is known as well. We denote the unique solution of this system by $\mathbf{x}^*$.

Solution of the above system of equations is the same as $\text{argmin}_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}$.

Definition 1 ($\mathbf{A}$-conjugate directions)**.** *Let $\mathbf{A}$ ($n \times n$) be a symmetric matrix. The vectors $\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$ (all $\in \mathbb{R}^n$) are called conjugate (or orthogonal) with respect to $\mathbf{A}$ if $\mathbf{d}_i^T\mathbf{Ad}_j = 0$ for all $i \neq j$.*

Lemma 1 (Linear independence)**.** *Let $\mathbf{A}$ ($n \times n$) be positive definite. If the vectors $\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$ (all $\in \mathbb{R}^n$) are conjugate (orthogonal) with respect to $\mathbf{A}$, then they are linearly independent.*

Proof. Let $\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$ are linearly dependent and let $\mathbf{d}_k = \sum_{j=1}^{k-1} \alpha_j\mathbf{d}_j$.

By definition of conjugacy, we have $\mathbf{d}_k^T\mathbf{Ad}_j = 0$ for all $j \neq k$. Since, $\mathbf{A}$ is pd, we have $\mathbf{d}_k^T\mathbf{Ad}_k > 0$. On the other hand, $\mathbf{d}_k^T\mathbf{Ad}_k = \sum_{j=1}^{k-1} \alpha_j\mathbf{d}_k^T\mathbf{Ad}_j = 0$ (Contradiction!) □

If $\mathbf{x}^*$ is a solution of the above system of equation, we can represent it as a linear combination of $\{\mathbf{d}_1, \ldots, \mathbf{d}_n\}$, which are conjugate vectors with respect to $\mathbf{A}$.

Let $\mathbf{x}^* = \sum_{j=1}^{n} \alpha_j\mathbf{d}_j$. It is easy to check that $\mathbf{d}_j^T\mathbf{Ax}^* = \alpha_j\mathbf{d}_j^T\mathbf{Ad}_j$. Thus $\alpha_j = \frac{\mathbf{d}_j^T\mathbf{Ax}^*}{\mathbf{d}_j^T\mathbf{Ad}_j}$. But by assumption, $\mathbf{Ax}^* = \mathbf{b}$. Thus $\alpha_j = \frac{\mathbf{d}_j^T\mathbf{b}}{\mathbf{d}_j^T\mathbf{Ad}_j}$. Hence, to know α_j, we do not need to know $\mathbf{x}^*$.

Hence $\mathbf{x}^* = \sum_{j=1}^{n} \frac{\mathbf{d}_j^T\mathbf{b}}{\mathbf{d}_j^T\mathbf{Ad}_j}\mathbf{d}_j$, no matrix inversion or nothing.

In practice, we do not know the conjugate vectors. We thus may follow Gram–Schmidt type algorithm.

Algorithm 1: Conjugate gradient

```
(i) Start setting $\mathbf{r}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_1$ with some starting value $\mathbf{x}_1$.
(ii) If $\mathbf{r}_0$ is small enough, return $\mathbf{x} = \mathbf{x}_1$ as solution.
(iii) Otherwise, set $\mathbf{d}_1 = \mathbf{r}_1$ and $k = 1$.
(iv) Start loop
    • $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}$
    • $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$
    • $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{d}_k$
    • If $\mathbf{r}_{k+1}$ is small enough, return $\mathbf{x} = \mathbf{x}_{k+1}$ as solution.
    • Otherwise, $\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$, $\mathbf{d}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{d}_k$, move $k = k + 1$ and Repeat
```

In the above algorithm, how the term 'gradient' fits in? Say $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$ and we are trying to solve $\min_{\mathbf{x}} f(\mathbf{x})$. Following the steps in gradient descent, the updates should look like $\mathbf{x}_{k+1} = \mathbf{x}_k - t_k \bigtriangledown f(\mathbf{x}_k)$. For our function, we have $\bigtriangledown f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$. Thus, we need to move towards $\mathbf{b} - \mathbf{A}\mathbf{x}$. The other vectors in the basis will be made conjugate to this gradient vector. Thus, the term 'conjugate gradient' comes in.

To see above, note that $\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k} = -\frac{\mathbf{r}_{k+1}\mathbf{A}\mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}$. Then, $\mathbf{d}_{k+1}^T \mathbf{A}\mathbf{d}_k = 0$

If the condition number of $\mathbf{A}$ is large, the convergence is slow as it leads to slower improvement. Hence, to ensure faster convergence in this case, **Preconditioning conjugate gradient** is used. [Condition number of a matrix $\mathbf{A}$ is $\kappa(\mathbf{A}) = \dfrac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})}$, where $\sigma_{\max}(\mathbf{A})$ and $\sigma_{\min}(\mathbf{A})$ are maximal and minimal singular values of $\mathbf{A}$ respectively. In our case, the singular values and eigenvalues are the same as $\mathbf{A}$ is symmetric]

Algorithm 2: Pre-conditioned conjugate gradient

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$$
$$\mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0$$
$$\mathbf{p}_0 = \mathbf{z}_0$$
$$k = 0$$

repeat

$$\alpha_k = \frac{\mathbf{r}_k^{\mathsf{T}}\mathbf{z}_k}{\mathbf{p}_k^{\mathsf{T}}\mathbf{A}\mathbf{p}_k}$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$$
$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$$

if r_{k+1} is sufficiently small, then exit loop end if

$$\mathbf{z}_{k+1} = \mathbf{M}^{-1}\mathbf{r}_{k+1}$$
$$\beta_k = \frac{\mathbf{r}_{k+1}^T\mathbf{z}_{k+1}}{\mathbf{r}_k^T\mathbf{z}_k}$$
$$\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$$
$$k = k + 1,$$

end repeat;
The result is x_{k+1}

In the above algorithm $\mathbf{M}$ is fixed. We often set $\mathbf{M}$ as $\mathbf{L}\mathbf{L}^T$ where $\mathbf{L}$ is an incomplete Cholesky decomposition of $\mathbf{A}$ or sometimes the diagonal entries in $\mathbf{A}$ (Jacobi preconditioner). Or, we often set $\mathbf{M}^{-1}$= some approximate inverse of $\mathbf{A}$. The approximate inverse computation would depend on the shape and structure of $\mathbf{A}$.

2 Constraint optimization

A typical optimization problem under constraints, look like:

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ such that } g_1(\mathbf{x}) \le 0, g_2(\mathbf{x}) \le 0, g_3(\mathbf{x}) \ge 0, \text{ etc.},$$

Lagrange multiplier methods are one of the most popular methods to solve such minimization problem. The objective function $J = f(\mathbf{x})$ is Lagrangian as by the constraint equations through a set of non-negative multiplicative Lagrange multipliers, $\lambda_j \geq 0$. The Lagrangian as objective function, $J_A(\mathbf{x}, \lambda_1, \lambda_2, \lambda_3)$, is,

$$f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \lambda_2 g_2(\mathbf{x}) - \lambda_3 g_3(\mathbf{x}).$$

Notice, that the last term is negative is due to the $\geq$ inequality. The reason is similar to the motivation around the penalty-induced regression methods. Specifically, since the aim is to minimize $f(\mathbf{x})$, the idea is to keep the Lagrangian as objective function a minimization problem. Since $g_1(\mathbf{x}) \leq 0, g_2(\mathbf{x}) \leq 0, g_3(\mathbf{x}) \geq 0$ for all $\mathbf{x}$, we have $J_A(\mathbf{x}, \lambda_1, \lambda_2, \lambda_3) \leq f(\mathbf{x})$.

Thus, $\max_{\lambda_1,\lambda_2,\lambda_3} J_A(\mathbf{x}, \lambda_1, \lambda_2, \lambda_3) \leq f(\mathbf{x})$ as $J_A(\mathbf{x}, \lambda_1, \lambda_2, \lambda_3) \leq f(\mathbf{x})$ holds for all positive λ_i's. And our optimization problem becomes,

$$\max_{\lambda_1,\lambda_2,\lambda_3} \min_{bx} J_A(\mathbf{x}, \lambda_1, \lambda_2, \lambda_3) \text{ such that } \lambda_1, \lambda_2, \lambda_3 \geq 0.$$

1. For minimization problem, we need to make sure $J_A(\mathbf{x}, \boldsymbol{\lambda}) \leq f(\mathbf{x})$ for $\lambda \geq 0$.
2. For maximization problem, we need to make sure $J_A(\mathbf{x}, \boldsymbol{\lambda}) \geq f(\mathbf{x})$ for $\lambda \geq 0$.

A popular strategy to solve Lagrange multiplier based constraint optimization is,

1. First, compute the derivate with respect to $\mathbf{x}$ as $\frac{\partial J_A(\mathbf{x},\lambda_1,\lambda_2,\lambda_3)}{\partial \mathbf{x}}$
2. Also the derivatives with respect to λ_j's which are $g_1(\mathbf{x})$, $g_2(\mathbf{x})$ and $g_3(\mathbf{x})$.

These equations define the relations to be used to derive expressions for, or compute values of, the value $\mathbf{x}^*$ (the optimal value). However, the optimal value of somes λ_i's may be zero (thus that constraint will not have any impact on the optimization) if the solution leads to something inconsistent.

We first (blindly) solve the system of equations that we get after taking first derivates with respect of $\mathbf{x}$ and λ_j's. In case of multiple constraints, we get multiple sets of possible solutions. If an inequality $g_j(\mathbf{x}) \geq 0$ constrains the optimum point, the corresponding Lagrange multiplier, λ_j^* , is positive. If an inequality $g_j(\mathbf{x}) \geq 0$ does not constrain the optimum point, the corresponding Lagrange multiplier, λ_j^* , is set to zero.

Consider the following constrained minimization problem:

$$\min_x \frac{1}{2}kx^2, \text{ for } k > 0 \text{ such that } x \geq b$$

The inequality is like $g_3()$ in our earlier definition. Thus, The Lagrangian as objective function is,

$$\max_\lambda \min_x \frac{1}{2}kx^2 - \lambda(x-b)$$

Now the taking derivatives would give us two equations $kx - \lambda = 0$ and $x - b = 0$. Let (x^*, λ^*) are the optimal values.

Case 1: $b > 0$ Then $x = b$ and $\lambda = kb > 0$ which satisfies the required condition. This is thus an optimal solution for x and λ.

Case 2: $b < 0$ Then, $x = b$ and $\lambda = kb < 0$, does not satisfy the required positivity condition on λ. Hence, we set $\lambda^* = 0$. Having set $\lambda^* = 0$, we have to minimize $\frac{1}{2}kx^2$ which leads to $x^* = 0$.

Hence, when $b > 0$, we have $(x^*, \lambda^*) = (b, kb)$, but when $b < 0$, we have $(x^*, \lambda^*) = (0, 0)$.

In this example, we can verify following two important properties:

1. $J_A(x^*, \lambda) \leq J_A(x^*, \lambda^*) \leq J_A(x, \lambda^*)$. Thus the Lagrangian as objective function is minimized at $x = x^*$ when we set $\lambda = \lambda^*$ and is maximized at $\lambda = \lambda^*$ when we set $x = x^*$.
2. Formally, $\min_x J_A(x, \lambda^*) = \max_\lambda J_A(x^*, \lambda) = J_A(x^*, \lambda^*)$.

Short digression

Minimax inequality It can be shown easily that $\max_\lambda \min_x J_A(x, \lambda) \leq \min_x \max_\lambda J_A(x, \lambda)$.

Proof. $J_A(x, \lambda) \leq \max_\lambda J_A(x, \lambda)$ for any x and λ. Also, $J_A(x, \lambda) \geq \min_x J_A(x, \lambda)$ for any x and λ. Combining the two, we have $\min_x J_A(x, \lambda) \leq \max_\lambda J_A(x, \lambda)$. Now, take the minimum over x on the right-hand side, then the maximum over λ on the left-hand side. □

Let $u^* = \min_x \max_\lambda J_A(x, \lambda)$ and $l^* = \max_\lambda \min_x J_A(x, \lambda)$. The difference $u^* - l^*$ is called the duality gap.

Minmax equality It can be shown that the gap $u^* - l^*$ is zero if:

- The domains of x and λ are both convex, and one of them is compact.
- The function J_A is convex-concave: $J_A(, \lambda)$ is convex for any given λ, and $J_A(x,)$ is concave for any given x.
- The function J_A is continuous

Above strategy to solve Lagrangian as objective function may become hard when we are in a multiparameter setting with several constraints. We then take another way which is the following,

1. First, compute $\frac{\partial J_A(\mathbf{x}, \lambda_1, \lambda_2, \lambda_3)}{\partial \mathbf{x}} = 0$ and solve $\mathbf{x}$ in terms of λ. Let the solution be, $h(\lambda_1, \lambda_2, \lambda_3)$
2. Use this solution of x and solve $\max_{\lambda_1, \lambda_2, \lambda_3} J_A(h(\lambda_1, \lambda_2, \lambda_3), \lambda_1, \lambda_2, \lambda_3)$

This strategy leads to a new objective function $J'_A(\lambda_1, \lambda_2, \lambda_3) = J_A(h(\lambda_1, \lambda_2, \lambda_3), \lambda_1, \lambda_2, \lambda_3)$ for our original problem. This is also called the 'dual' problem. Whenever we write the optimization problem in terms of the Lagrange multiplier's only, it is called the *Dual*. Solving a dual problem is easier due to fewer number of parameters in the objective function. If I am too strict, this is called Lagrange dual. There are other types of dual representations as well.

Example:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{M}\mathbf{x} + \mathbf{c}^T\mathbf{x} + d \text{ such that } \mathbf{A}\mathbf{x} \leq \mathbf{b}.$$

First, we write the Lagrangian as objective function, which is (*the primal form*),

$$\max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} J(\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{x}^T\mathbf{M}\mathbf{x} + \mathbf{c}^T\mathbf{x} + d + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}),$$

such that $\boldsymbol{\lambda} \geq 0$.

We first solve $\mathbf{x}$ in terms of $\boldsymbol{\lambda}$ from $\frac{\partial J(\mathbf{x}, \boldsymbol{\lambda})}{\partial \mathbf{x}} = 0 \implies \mathbf{M}\mathbf{x} + \mathbf{c} + \mathbf{A}^T\boldsymbol{\lambda} = 0 \implies \mathbf{x} = -\mathbf{M}^{-1}(\mathbf{c} + \mathbf{A}^T\boldsymbol{\lambda})$.

Hence, the Lagrangian dual problem (*the dual form*) is,

$$\max_{\boldsymbol{\lambda}} \frac{1}{2}(\mathbf{c}+\mathbf{A}^T\boldsymbol{\lambda})^T\mathbf{M}^{-1}(\mathbf{c}+\mathbf{A}^T\boldsymbol{\lambda}) - \mathbf{c}^T\mathbf{M}^{-1}(\mathbf{c}+\mathbf{A}^T\boldsymbol{\lambda}) + d + \boldsymbol{\lambda}^T(\mathbf{A}\{-\mathbf{M}^{-1}(\mathbf{c}+\mathbf{A}^T\boldsymbol{\lambda})\} - \mathbf{b}).$$

Upon simplification,

$$\max_{\boldsymbol{\lambda}:\boldsymbol{\lambda}\geq 0} -\frac{1}{2}\boldsymbol{\lambda}^T\mathbf{A}\mathbf{M}^{-1}\mathbf{A}^T\boldsymbol{\lambda} - (\mathbf{c}^T\mathbf{M}^{-1}\mathbf{A}^T + \mathbf{b}^T)\lambda - \frac{1}{2}\mathbf{c}^T\mathbf{M}^{-1}\mathbf{c} + d.$$

Note that if the quadratic term in the primal form (quadratic in the primal variable $\mathbf{x}$) is positive, then the quadratic term in the dual form (quadratic in the dual variable $\boldsymbol{\lambda}$) is negative. So, again, it makes sense to minimize over the primal variable $\mathbf{x}$ and to maximize over the dual variable $\boldsymbol{\lambda}$.

In general, the dual function is $g(\boldsymbol{\lambda}) = \min_{\mathbf{x}} J_A(\mathbf{x}, \boldsymbol{\lambda})$.

However, this may not also be feasible to solve $\mathbf{x}$ explicitly from $\frac{\partial J(\mathbf{x},\boldsymbol{\lambda})}{\partial \mathbf{x}}$.

Lagrange multipliers do have some similarities with penalty based methods like LASSO and Ridge.

For example, the well-known LASSO objective function $\min\limits_{\boldsymbol{\beta}} \frac{1}{n}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1$ is a Lagrangian dual of

$$\min_{\boldsymbol{\beta}} \left\{\frac{1}{n}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2\right\} \text{ subject to } \|\boldsymbol{\beta}\|_1 \leq t.$$

Similar representations also hold for Ridge or Elastic net.

2.1 In case of equality constraint:

Consider the problem

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ such that } \mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

The entire subsection is to solve this problem.
Dual gradient ascent:
The Lagrangian is,

$$J(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b})$$

Our objective is to $\max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} J(\mathbf{x}, \boldsymbol{\lambda})$.

The dual gradient ascent repeats for $k = 1, 2, 3, \ldots$

$$\begin{aligned}
\mathbf{x}^{(k+1)} =& \arg\min_{\mathbf{x}} f(\mathbf{x}) + (\boldsymbol{\lambda}^{(k)})^T \mathbf{A}\mathbf{x} \text{ (only the part that involves } \mathbf{x}) \\
\boldsymbol{\lambda}^{(k+1)} =& \boldsymbol{\lambda}^{(k)} + t_{k+1}(\mathbf{A}\mathbf{x}^{(k+1)} - \mathbf{b}), \text{ (Gradient-based update for } \boldsymbol{\lambda})
\end{aligned}$$

The step size is chosen just like Gradient descent.

Why 'ascent'? We need to maximize $J(\mathbf{x}, \boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$ and thus, we are applying the gradient-based move to maximize the loss with respect to $\boldsymbol{\lambda}$. Thus, we are ascending along the loss function with respect to $\boldsymbol{\lambda}$.

In order for dual ascent to work, it requires some additional conditions on the function. To avoid such conditions and bring robustness, the following representation was developed.

Augmented Lagrangians and the Method of Multipliers (MM):
The augmented Lagrangian is,

$$J(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2,$$

where $\rho > 0$ is called the penalty parameter. The augmented Lagrangian can be viewed as the (unaugmented) Lagrangian associated with the problem

$$\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \text{ such that } \mathbf{A}\mathbf{x} = \mathbf{b}$$

Applying dual ascent to the modified problem yields the algorithm:

$$\begin{aligned}
\mathbf{x}^{(k+1)} =& \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + (\boldsymbol{\lambda}^{(k)})^T \mathbf{A}\mathbf{x} \text{ (only the part that involves } \mathbf{x}) \\
\boldsymbol{\lambda}^{(k+1)} =& \boldsymbol{\lambda}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} - \mathbf{b}), \text{ (Gradient-based update for } \boldsymbol{\lambda}),
\end{aligned}$$

which is known as the method of multipliers (MM) for solving our original problem in (1). The penalty parameter ρ now comes as the step-length parameter. Cross-validation can be applied to tune ρ.

2.2 Very brief on Alternating direction method of multipliers (ADMM)

ADMM is to solve the following optimization problem,

$$\min_{\mathbf{x},\mathbf{z}} f(\mathbf{x}) + f(\mathbf{z}) \text{ such that } \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c}$$

The augmented Lagrangian is,

$$J(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + f(\mathbf{z}) + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}\|_2^2,$$

Normal MM algorithm is

$$\begin{aligned}(\mathbf{x}^{(k+1)}, \mathbf{z}^{(k+1)}) &= \arg\min_{\mathbf{x},\mathbf{z}} f(\mathbf{x}) + f(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}\|_2^2 + (\boldsymbol{\lambda}^{(k)})^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z}) \text{ (Part that involves } \mathbf{x}, \mathbf{z}) \\ \boldsymbol{\lambda}^{(k+1)} &= \boldsymbol{\lambda}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)}\mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}), \text{ (Gradient-based update for } \boldsymbol{\lambda}),\end{aligned}$$

ADMM algorithm is

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z}^{(k)} - \mathbf{c}\|_2^2 + (\boldsymbol{\lambda}^{(k)})^T(\mathbf{A}\mathbf{x}) \text{ (only the part that involves } \mathbf{x}) \\ \mathbf{z}^{(k+1)} &= \arg\min_{\mathbf{z}} f(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z} - \mathbf{c}\|_2^2 + (\boldsymbol{\lambda}^{(k)})^T(\mathbf{B}\mathbf{z}) \text{ (only the part that involves } \mathbf{z}) \\ \boldsymbol{\lambda}^{(k+1)} &= \boldsymbol{\lambda}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)}\mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}), \text{ (Gradient-based update for } \boldsymbol{\lambda}),\end{aligned}$$

The first two steps will find exact minimizer having others fixed and then 3rd step will update the $\boldsymbol{\lambda}$. As you may have guessed, we can generalize the loss function to have even more than 2 terms in the objective function.

Application in distributed computing: ADMM is a popular method for online and distributed optimization on a large scale. Here is an example for LASSO penalty based linear regression.

Say, we have collected, 10000 datapoints. Thus, we have $(y_1, \mathbf{x}_1), \dots, (y_{10000}, \mathbf{x}_{10000})$. Here $\mathbf{x}_i$'s are p-dimensional and p is also large, say 1 million. Then the standard LASSO regression would solve $\sum_{i=1}^{10000}(y_i - \mathbf{x}_i^T\boldsymbol{\beta})^2 + \theta\|\boldsymbol{\beta}\|_1$. This may be a computational nightmare in terms of both memory and efficiency. Also, this could be a part of streaming data source. In that case, more data will be collected overtime, and we need to refit this giant model again.

Fortunately, ADMM can help us solve this problem using less memory. However, we might loose some accuracy. We break this data into several subsets, say each with size 1000. Then there will be 10000/1000 = 10 batches of datasets. In this case, the constraint optimization routine will be,

$$\min_{\boldsymbol{\beta}_1,\dots,\boldsymbol{\beta}_{10},\boldsymbol{\beta}} \sum_{k=1}^{10} \|\mathbf{y}_k - \mathbf{X}_k^T\boldsymbol{\beta}_k\|_2^2 + \theta\|\boldsymbol{\beta}\|_1 \text{ such that } \boldsymbol{\beta}_k = \boldsymbol{\beta} \text{ for all } k = 1, \dots, 10,$$

where $\mathbf{y}_k$ is a block of data points with index $1000(k-1)+1, \ldots, 1000k$ and $\mathbf{X}_k$ is data matrix of dimension $1000 \times p$ with observations $1000(k-1)+1, \ldots, 1000k$. **Here we have batch-specific separate regression coefficients and then a constraint to bring them the same.**

The augmented Lagrangian is,

$$J(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{10}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{10}) = \sum_{k=1}^{10} \|\mathbf{y}_k - \mathbf{X}_k^T \boldsymbol{\beta}_k\|_2^2 + \theta \|\boldsymbol{\beta}\|_1 + \sum_{k}^{10} \boldsymbol{\lambda}_k^T (\boldsymbol{\beta}_k - \boldsymbol{\beta}) + \frac{\rho}{2} \sum_{k}^{10} \|\boldsymbol{\beta}_k - \boldsymbol{\beta}\|_2^2.$$

ADMM algorithm will then run as,

1. Minimize $\boldsymbol{\beta}_k$'s separately and parallelly. There will be closed form solution for this as we can reformulate the loss for $\boldsymbol{\beta}_k$ like a least square loss.
2. Minimize $\boldsymbol{\beta}$. This will also have close form solution based on Qn4 of HW3.
3. Then gradient ascent update for $\boldsymbol{\lambda}_k$'s.

We may attempt to solve this problem in another HW, will see. But similar ideas are applicable for other large scale distributed optimizations and online computing as well. ADMM is also not the only candidate for distributed computing. But ADMM often helps to transform a problem into a convex problem.

In case of online computing, the objective funtion is,

$$\min_{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2} f(\boldsymbol{\beta}_1) + f(\boldsymbol{\beta}_2) \text{ such that } \boldsymbol{\beta}_1 = \boldsymbol{\beta}_2,$$

where $f(\boldsymbol{\beta}_1)$ is the loss based on old data and $f(\boldsymbol{\beta}_2)$ is the loss of newly acquired data.

ADMM is generally applied with linear and equality constraint. It is possible to extend ADMM for inequality and linear constaint. For non-linear constraint, I am not sure if there is any well established method yet.

If interested to know more about ADMM, do read [1] (`https://stanford.edu/~boyd/papers/pdf/admm_distr_stats.pdf`) and the slides at `https://web.stanford.edu/class/ee364b/lectures/admm_slides.pdf`.

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.