The ADMM

An alternating alternative to interior point methods

Operator Splitting Quadratic Programming

We will tackle the new QP using the OSQP solver by Oxford University

OSQP is a modern solver for Quadratic Programs in the form:

$$\arg\min_x \left\{ \frac{1}{2} x^T P x + q^T x \mid l \leq Ax \leq u \right\}$$

The solver:

- is very fast, especially for problems with sparse matrices
- is available under a (very permissive) Apache 2.0 license
- has API for many programming languages

The solver relies on the Alternating Direction Method of Multipliers (ADMM)

- ...Plus a bunch of clever "tricks" to improve speed
- Here we will discuss only the basic ADMM, to provide an intuition

The Alternating Direction Method of Multipliers

The ADMM solves numerical constrained optimization problems in the form:

$$\text{argmin } f(x) + g(z)$$
$$\text{subject to: } Ax + Bz = c$$

- Where f and g are assumed to be convex

The methods relies on a so-called augmented Lagrangian

This is a reformulation where the constraints are turned into penalty terms:

$$\mathcal{L}_\rho(x, z, \lambda) = f(x) + g(z) + \lambda^T(Ax + Bz - c) + \frac{1}{2}\rho\|Ax + Bz - c\|_2^2$$

- The algorithm idea is to optimize the augmented Lagrangian
- ...And to encourage constraint satisfaction via the penalty terms
- In practice, this is done by adjusting the multiplier vector λ

The Alternating Direction Method of Multipliers

The ADMM operates as follows

We start from an initial assignment $\boldsymbol{x}^{(0)}, \boldsymbol{z}^{(0)}, \boldsymbol{\lambda}^{(0)}$, then we iterate:

$$
\begin{aligned}
x^{(k+1)} &= \text{argmin}_x \mathcal{L}_\rho(x, z^{(k)}, \lambda^{(k)}) \\
z^{(k+1)} &= \text{argmin}_z \mathcal{L}_\rho(x^{(k+1)}, z, \lambda^{(k)}) \\
\lambda^{(k+1)} &= \lambda^k + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c)
\end{aligned}
$$

In other words:

- We keep everything fixed and we optimize over $\boldsymbol{x}$ to obtain $\boldsymbol{x}^{k+1}$
- We replace $\boldsymbol{x}^k$ with $\boldsymbol{x}^{k+1}$, keep everything fixed and optimize over $\boldsymbol{z}$
- Finally, we update the multiplier vector

The switch between x and z optimization is the "alternating" part

...While the use of the multipliers $\boldsymbol{\lambda}$ explains the rest of the name

Multiplier Update

Let's try to understand better the multiplier update

...Which consists in the rule:

$$\lambda^{(k+1)} = \lambda^{(k)} + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c)$$

- The term $Ax^{k+1} + Bz^{k+1} - c$ is just the current constraint violation
- ...But it is also $\nabla_\lambda \mathcal{L}_\rho(x^{(k+1)}, z^{(k+1)}, \lambda)$, making this a gradient update

Multiplier Update

Let's try to understand better the multiplier update

...Which consists in the rule:

$$\lambda^{(k+1)} = \lambda^{(k)} + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c)$$

- The term $Ax^{k+1} + Bz^{k+1} - c$ is just the current constraint violation
- ...But it is also $\nabla_\lambda \mathcal{L}_\rho(x^{(k+1)}, z^{(k+1)}, \lambda)$, making this a gradient update

If $(Ax^{(k+1)} + Bz^{(k+1)})_i > c_i$ for some constraint i:

- Then we increase the corresponding multiplier λ_i
- So that the penalty term $\lambda_i(Ax^{(k+1)} + Bz^{(k+1)} - c)_i$ grows
- This will push the next iteration to reduce the degree of violation

Multiplier Update

Let's try to understand better the multiplier update

...Which consists in the rule:

$$\lambda^{(k+1)} = \lambda^{(k)} + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c)$$

- The term $Ax^{k+1} + Bz^{k+1} - c$ is just the current constraint violation
- ...But it is also $\nabla_\lambda \mathcal{L}_\rho(x^{(k+1)}, z^{(k+1)}, \lambda)$, making this a gradient update

If $(Ax^{(k+1)} + Bz^{(k+1)})_i < c_i$ for some constraint i:

- Then we decrease the corresponding multiplier λ_i
- So that the penalty term $\lambda_i(Ax^{(k+1)} + Bz^{(k+1)} - c)_i$ grows (again)
- This will push the next iteration to reduce the degree of violation (again)

Multiplier Update

Let's try to understand better the multiplier update

...Which consists in the rule:

$$\lambda^{(k+1)} = \lambda^{(k)} + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c)$$

- The term $Ax^{k+1} + Bz^{k+1} - c$ is just the current constraint violation
- ...But it is also $\nabla_\lambda \mathcal{L}_\rho(x^{(k+1)}, z^{(k+1)}, \lambda)$, making this a gradient update

If $(Ax^{(k+1)} + Bx^{(k+1)})_i = c_i$ for some constraint i:

- Then we keep the corresponding multiplier λ_i as it is
- The constraint is not violated, so there is nothing to do

Main Advantages of the Method

The method has two major advantages:

1) The x and z variables can be handled in isolation

- This results into simpler problems
- ...And in some cases enables massive parallelization

2) The ADMM converges under relatively mild conditions

In the classical formulation, f and g need to be closed, proper, convex functions

- They do not need to be differentiable
- They can take the value $+\infty$ (and hence represent constraints)

The second condition is that $\mathcal{L}_0(x, z, \lambda)$ should have a saddle point

- This one is way trickier to check...

The full convergence proof can be found e.g. here

How are we going to use this to address Quadratic Programs?

QP Reformulation

Let's see these advantages at work on Quadratic Programs

We need to solve:

$$\text{argmin}_x \left\{ \frac{1}{2} x^T P x + q^T x \mid l \leq Ax \leq u \right\}$$

...Which we reformulate to:

$$\begin{aligned} \text{argmin } & x^T P x + q^T x \\ \text{subject to: } & z = Ax \\ & l \leq z \leq u \end{aligned}$$

- We have introduced a new variables z
- ...And posted the inequality constraints over that

QP Reformulation

Then, we turn the inequality constraints into a function

$$\operatorname{argmin}\; x^T P x + q^T x + I_{l \le z \le u}(z)$$
$$\text{subject to: } z = Ax$$

Where $I_{l \le z \le u}$ is the constraint indicator function of $l \le z \le u$

- It's value is $+\infty$ when the constraint is violated and 0 elsewhere
- In this case, it is non-differentiable, but closed, proper, and convex!

...And $x^T P x + q^T x$ is our usual cost term

- It is differentiable
- ...And closed, proper, and convex if P is semi-definite positive

We can now proceed to apply the ADMM!

The ADMM Steps for QP

We need to start from a feasible $x^{(0)}, z^{(0)}, \lambda^{(0)}$:

...Which we can get by setting $\lambda^{(0)} = 0$, $z^{(0)} = l$, then solving $Ax^{(0)} = l$

The ADMM Steps for QP

We need to start from a feasible $x^{(0)}, z^{(0)}, \lambda^{(0)}$:

...Which we can get by setting $\lambda^{(0)} = 0$, $z^{(0)} = l$, then solving $Ax^{(0)} = l$

The x minimization step for $\tilde{z} = z^{(k)}$ becomes:

$$\text{argmin}_x \; x^T P x + q^T x + I_{l \leq z \leq u}(\tilde{z}) + \lambda^T(\tilde{z} - Ax) + \frac{1}{2}\rho\|\tilde{z} - Ax\|_2^2$$

And then, since $\tilde{z}$ is fixed and feasible:

$$\text{argmin} \; x^T P x + q^T x + \lambda^T(\tilde{z} - Ax) + \frac{1}{2}\rho\|\tilde{z} - Ax\|_2^2$$

- This is a convex, differentiable, quadratic minimization problem
- ...And it can be tackled by solving a linear system of equations

The ADMM Steps for QP

The z minimization step for $\tilde{x} = x^{(k+1)}$ becomes

$$\operatorname{argmin}_z \; \tilde{x}^T P \tilde{x} + q^T \tilde{x} + \chi_{l \le z \le u}(z) + \lambda^T (z - A\tilde{x}) + \frac{1}{2}\rho \|z - A\tilde{x}\|_2^2$$

Since $\tilde{x}$ is fixed, this can be reformulated as:

$$\operatorname{argmin} \; \lambda^T z + \frac{1}{2}\rho \|z - A\tilde{x}\|_2^2$$
$$\text{subject to: } l \le z \le u$$

...And finally separated in to n very simple problems (one per variable):

$$\operatorname{argmin}_{z_j} \left\{ \lambda_j z_j + \frac{1}{2}\rho (z_j - A_j \tilde{x})^2 \mid l \le z_j \le u \right\}$$

Some Considerations

We used the ADMM to break QP into a sequence of simpler problems

The method can be used in other clever ways:

- Optimization with non-differentiable reguralizers
- Parallel training, by splitting examples into multiple problems

The ADMM is best used for convex problems

- Classical results are for convex problems only
- There are some (local) results non non-convex problems (e.g. this one)

About the convergence pace

- It's very fast in the first iterations, but much slower later
- You can high-quality solutions early, but reaching the optimum takes long
- All in all, it's best used as an approximate method

Some References

If you are interested, there's a great book on the topic:

- "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers", by Stephen Boyd