Juni 2021

Technological University
of Denmark


# ADMM

*The project work in the course 02953, convex optimization.*

Mads Esben Hansen
s174434

# 1 Intro

This report is the final project in the course 02953, convex optimization at DTU. In the report we will focus on ADMM. We will introduce the theory leading up the ADMM and the algorithm itself. We will investigate how in can be used within distributed learning and/or parallel computing. We will make implementations of the ADMM algorithm to solve suitable problems. These implementations will be tested and the results discussed.

# 2 Theory

## 2.1 Dual ascent

Consider:

$$\min_x \quad f(x) \tag{1}$$
$$s.t. \quad Ax = b \tag{2}$$

We then get the Lagrangian by:

$$\mathbf{L}(x, \lambda) = f(x) + \lambda^T(Ax - b) \tag{3}$$
$$\mathbf{L}^*(\lambda) = \inf_x \mathbf{L}(x, \lambda) = -f^*(-A^T\lambda) - b^T\lambda \tag{4}$$

With $f^*(h) = \sup_x h^T x - f(x)$. Dual ascent is then given by:
Choose some $\lambda_0$.

$$x_{k+1} = \operatorname*{argmin}_x \mathbf{L}(x, \lambda) \tag{5}$$
$$\lambda_{k+1} = \lambda_k + t_k(Ax_{k+1} - b) \tag{6}$$

This method converges when $f$ is strongly convex. If strong duality holds the solution to the primal and dual problem are equal. It has a convergence rate of $O(\frac{1}{k})$, so it is quite slow.

## 2.2 Dual decomposition

A special case of dual ascent is given, when $f$ is decomposeable. Consider:

$$\min_{x_1, x_2, \ldots, x_v} \quad f(x) = \sum_{i=1}^{v} f_i(x_i) \tag{7}$$
$$s.t. \quad Ax = b \tag{8}$$

Let $A$ be a matrix with columns corresponding to to the $x_i$'s, then:

$$f(x) = \sum f_i(x_i) \quad and \quad Ax = \sum A_i x_i = b \tag{9}$$

Now we get the dual ascent algorithm by:

$$x_{k+1} = \operatorname*{argmin}_x f(x) + \lambda_k^T(Ax - b) \tag{10}$$
$$= \operatorname*{argmin}_{x_1, x_2, \ldots} \sum f(x) + \lambda_k^T((\sum A_i x_i) - b) \tag{11}$$
$$\lambda_{k+1} = \lambda_k + t_k(Ax_{k+1} - b) \tag{12}$$

## 2.3 Augmented form

Dual ascent is sensitive when $t_k \leq m$, therefore the augmented form is commonly used which given by:

$$\min_x \quad f(x) + \frac{\rho}{2}||Ax - b||_2^2 \tag{13}$$

$$s.t. \quad Ax = b \tag{14}$$

Notice that $||Ax - b||_2^2 = 0$ when the solution is feasible, hence the problem is equivalent for all feasible solutions. In practise it turn out that adding this "barrier" term introduces strong convexity and improves the convergence. The dual ascent now becomes:

$$x_{k+1} = \underset{x}{\operatorname{argmin}}\ f(x) + \lambda_k^T Ax + \frac{\rho}{2}||Ax - b||_2^2 \tag{15}$$

$$\lambda_{k+1} = \lambda_k + \rho(Ax_{k+1} - b) \tag{16}$$

## 2.4 ADMM

The alternating direction method of multipliers (ADMM) is a way of decomposing the problem on an augmented form. Consider:

$$\min_{x_1, x_2} \quad f_1(x_2) + f_2(x_2) \tag{17}$$

$$s.t. \quad A_1x_1 + A_2x_2 = b \tag{18}$$

Then the augmented Lagrangian becomes:

$$\mathbf{L}_\rho(x_1, x_2, \lambda) = f_1(x_1) + f_2(x_2) + \lambda^T(A_1x_1 + A_2x_2 - b) + \frac{\rho}{2}||A_1x_1 + A_2x_2 - b||_2^2 \tag{19}$$

Notice that $||A_1x_1 + A_2x_2 - b||_2^2$ cannot be decomposed. In stead the ADMM solves the directions iteratively:

$$(x_1)_{k+1} = \underset{x_1}{\operatorname{argmin}}\ \mathbf{L}(x_1, (x_2)_k, \lambda_k) \tag{20}$$

$$(x_2)_{k+1} = \underset{x_2}{\operatorname{argmin}}\ \mathbf{L}((x_1)_{k+1}, x_2, \lambda_k) \tag{21}$$

$$\lambda_{k+1} = \lambda_k + \rho(A_1(x_1)_{k+1} + A_2(x_2)_{k+1} - b) \tag{22}$$

Often we prefer the "scaled ADMM" formulation, where $\lambda$ is replaced by $w = \frac{\lambda}{\rho}$. Then it simplifies to:

$$(x_1)_{k+1} = \underset{x_1}{\operatorname{argmin}}\ f_1(x_1) + \frac{\rho}{2}||A_1x_1 + A_2(x_2)_k - b + w_k||_2^2 \tag{23}$$

$$(x_2)_{k+1} = \underset{x_2}{\operatorname{argmin}}\ f_2(x_2) + \frac{\rho}{2}||A_1(x_1)_{k+1} + A_2x_2 - b + w_k||_2^2 \tag{24}$$

$$w_{k+1} = w_k + A_1(x_1)_{k+1} + A_2(x_2)_{k+1} - b \tag{25}$$

## 2.5 Stopping criterion

The necessary and sufficient conditions of ADMM are given by the primal and dual feasibility:

$$A_1x_1^* + A_2x_2^* = b \tag{26}$$

$$0 \in \partial f_1(x_1^*) + A_1^T\lambda^* \tag{27}$$

$$0 \in \partial f_2(x_2^*) + A_2^T\lambda^* \tag{28}$$

$x_2$ minimizes $\mathbf{L}_\rho((x_1)_{k+1}, x_2, \lambda_k)$, hence the last condition is always satisfied. $x_1$ minimizes $\mathbf{L}_\rho(x_1, (x_2)_k, \lambda_k)$, we therefore have:

$$0 \in \partial f((x_1)_{k+1}) + A_1^T \lambda_k + \rho A_1^T (A_1(x_1)_{k+1} + A_2(x_2)_k - b) \tag{29}$$
$$= \partial f((x_1)_{k+1}) + A_1^T \lambda_{k+1} + \rho A_1^T A_2((x_2)_k - (x_2)_{k+1})) \tag{30}$$
$$\rightarrow \rho A_1^T A_2((x_2)_k - (x_2)_{k+1})) \in \partial f((x_1)_{k+1}) + A_1^T \lambda_{k+1} \tag{31}$$

We can therefore see $s_{k+1} = \rho A_1^T A_2((x_2)_k - (x_2)_{k+1}))$ as a residual for dual feasibility. $r_{k+1} = A_1(x_1)_{k+1} + A_2(x_2)_{k+1} - b$ can be seen as the primal residual.
Both the primal and dual residual will converge to 0, as shown by (Boyd et al. 2011). We therefore stop the algorithm when both the primal and dual residuals are sufficiently close to 0.

## 2.6 Consensus

This idea extends further when a problem can be divided into many part. Consider:

$$\min_x \quad f(x) \quad = \quad \min_{x_1, x_2, \ldots, x_N} \quad f(x) = \sum_{i=1}^{N} f_i(x_i) \tag{32}$$
$$s.t. \quad x_i = z, \quad for\ i = 1, .., N \tag{33}$$

This is known as a global consensus problem, with Lagrangian on augmented form defined by:

$$\mathbf{L}_\rho(x_1, \ldots, x_N, z, w) = \sum \left( f_i(x_i) + w_i^T (x_i - z) + (\rho/2) \|x_i - z\|_2^2 \right) \tag{34}$$

The ADMM algorithm then becomes:

$$(x_i)_{k+1} = \underset{x_i}{\operatorname{argmin}} \ (f_i(x_i) + (w_i)_k^T (x_i - z_k) + (\rho/2)||x_i - z_k||_2^2) \tag{35}$$
$$z_{k+1} = \frac{1}{N} \sum ((x_i)_{k+1} + (1/\rho)(w_i)_k) \tag{36}$$
$$(w_i)_{k+1} = (w_i)_k + \rho((x_i)_{k+1} - z_{k+1}) \tag{37}$$

The primal and dual residuals becomes:

$$r_k = ((x_1)_k - \bar{x}_k, ..., (x_N)_k - \bar{x}_k), \quad s_k = -\rho(\bar{x}_k - \bar{x}_{k-1}, ..., \bar{x}_k - \bar{x}_{k-1}) \tag{38}$$

## 2.7 Lasso

Lasso is given by:

$$\min_\beta \quad \frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_1 \tag{39}$$

This can be re-written as:

$$\min_{\beta, \alpha} \quad \frac{1}{2}||y - X\beta||_2^2 + \lambda||\alpha||_1 \tag{40}$$
$$s.t. \quad \beta = \alpha \tag{41}$$

The problem can now be solved using ADMM by:

$$\beta_{k+1} = \underset{\beta}{\operatorname{argmin}} \ \frac{1}{2}||y - X\beta||_2^2 + \lambda||\alpha_k||_1 + \frac{\rho}{2}||\beta - \alpha_k + w_k||_2^2 \tag{42}$$
$$\alpha_{k+1} = \underset{\alpha}{\operatorname{argmin}} \ \frac{1}{2}||y - X\beta_{k+1}||_2^2 + \lambda||\alpha||_1 + \frac{\rho}{2}||\beta_{k+1} - \alpha + w_k||_2^2 \tag{43}$$
$$w_{k+1} = w_k + \beta_{k+1} - \alpha_{k+1} \tag{44}$$

The solution to the iterates of $\beta$ and $\alpha$ can be found as solution to a ridge regression and by the help of a soft threshold operator, as shown by (Boyd et al. 2011):

$$\beta_{k+1} = (X^T X + \rho I)^{-1}(X^T y + \rho(\alpha_k - w_k)) \tag{45}$$
$$\alpha_{k+1} = \mathbf{S}_{\lambda/\rho}(\beta_{k+1} + w_k) \tag{46}$$

Where

$$\mathbf{S}_\kappa(a) = \begin{cases} a - \kappa & a > \kappa \\ 0 & |a| \leq \kappa \\ a + \kappa & a < -\kappa \end{cases} \tag{47}$$

### 2.7.1 Distributed lasso

Given the lasso problem, and assuming X and y can be partitioned in rows:

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_N \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \tag{48}$$

Then the lasso problem can be formulated as:

$$\min_{\beta,\alpha} \quad \sum \frac{1}{2}||y_i - X_i\beta_i||_2^2 + \lambda||\alpha||_1 \tag{49}$$
$$s.t. \quad \beta_i = \alpha, \quad for\ i = 1, ..., N \tag{50}$$

The distributed ADMM algorithm then becomes (note, the first sub-problem is unchanged, hence the solution is the same as before):

$$(\beta_i)_{k+1} = (X_i^T X_i + \rho I)^{-1}(X_i^T y_i + \rho(\alpha_k - w_k)) \tag{51}$$
$$\alpha_{k+1} = \mathbf{S}_{\lambda/\rho N}(\bar{\beta}_{k+1} + \bar{w}_k) \tag{52}$$
$$(w_i)_{k+1} = (w_i)_k + (\beta_i)_{k+1} - \alpha_{k+1} \tag{53}$$

# 3 Results

To test the properties of ADMM we have implemented 2 functions, one that solves a standard lasso problem, as described in the theory section, and one that solves the lasso problem by partitioning the data points in N sub-parts and solves the problem as a distributed problem.
To test the methods we generate a random problem on the form $b = Ax + v$, where $A_{m\times n} \sim \mathcal{N}(0, I)$, $x$ is a feature vector with 50 randomly selected non-zero values that are normally distributed with unit variance and 450 zero values. $v$ is a noise vector with variance $10^{-3}$. 100.000 data points have been generated, i.e. $m = 100000$.
The goal is now to use lasso to estimate the feature vector, i.e. $x$.

## 3.1 Lasso

The first thing we do, is that we re-estimate $x$. Since we know its true values, we can compute the norm of the difference between the true variable, $x$, and its estimate $\hat{x}$.

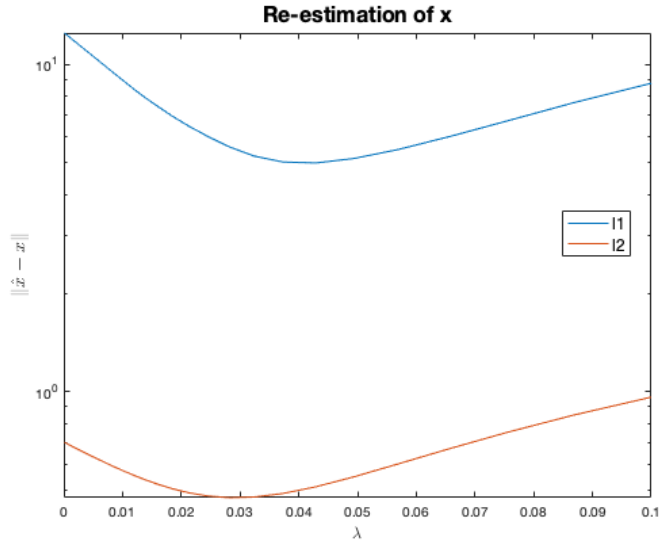


Figure 1: Re-estimation of $x$.

In figure 1 we can see that the regularizing parameter, $\lambda$, has a quite significant effect on the ability to estimate $x$. In this case, it seems that a $\lambda$ in the region of 0.035 produces the best re-estimation, based on both the l1- and l2-norm.
We can now address how the ADMM performs in particular. We can start by addressing the case, where the regularizing parameter is very large, meaning the problem is over-regularized.

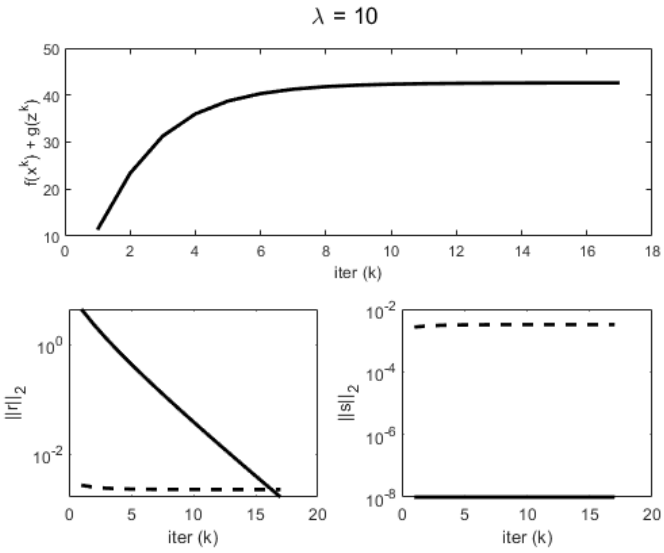


Figure 2: Over-regularized problem.

When $\lambda$ is very large, the regularizing term will be dominant in the objective functions. Hence, all features will very quickly go towards zero. This means that they all become equal very quickly, and we therefore see that we attain dual feasibility very quickly. This is seen in figure 2, where the 2-norm of the dual residual $s$ is below the threshold in 1 iteration.
Next we will address the situation where $\lambda$ is very small.

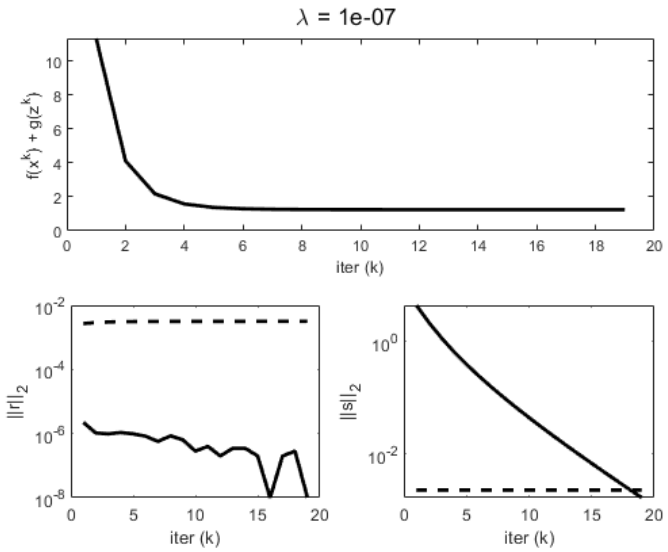


Figure 3: Under-regularized problem.

This time we see that opposite as before, namely that because the problem is under-regularized, the emphasis is on the first term, and hence it quickly becomes primal feasible.
Lastly we will address a reasonable $\lambda$ value.

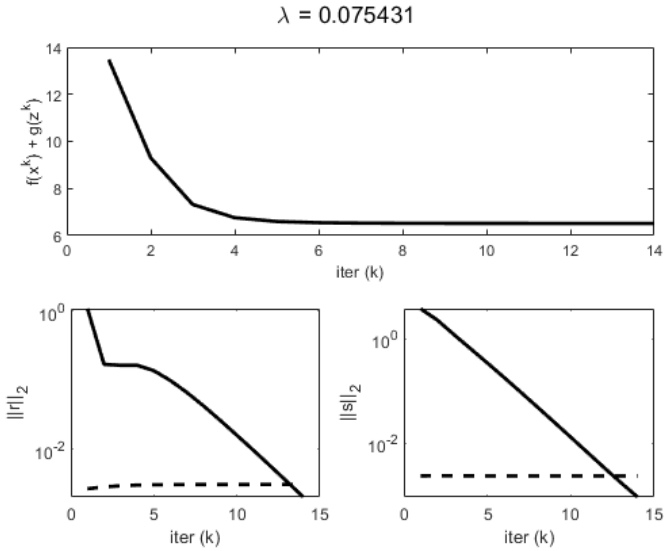


Figure 4: Reasonable-regularized problem.

In this case, we can see that both terms in the objective function is of similar importance. This results in a standard ADMM behaviour, where each term is optimized iteratively. We see this in figure 4 were it takes roughly the same number of iterations for the problem to become both primal and dual feasible. If we address the pareto optimal curve for varying $\lambda$ we get the following result:

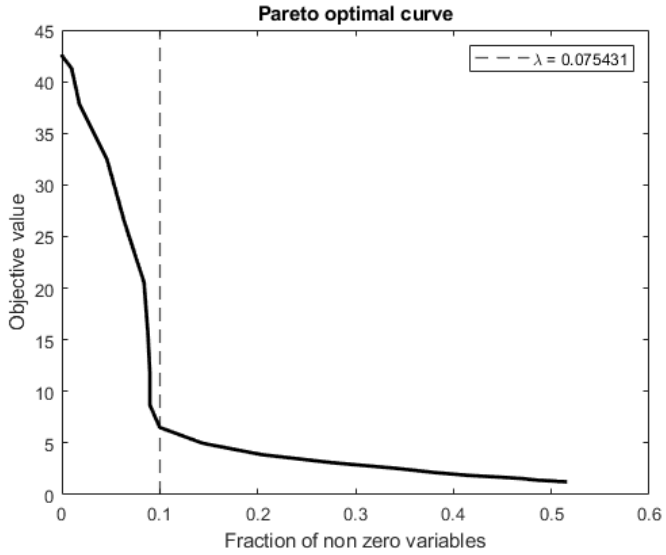


Figure 5: Pareto optimal curve.

In figure 5 we can see for $\lambda = 0.075$ we get the same fraction of non-zero variables in $\hat{x}$ as in $x$. We can further see that from a fraction of 0.5 until 0.1 it is relatively "cheap" on the objective function. This means that the optimal objective value increases very little until the fraction of non-zero variables reaches 0.1. When the regularizing is stronger (seen as a fraction lower than 0.1) it is very expensive. In other words, it seems safe to assume that $x$ can be reasonably estimated with a fraction of non-zero variable of roughly 0.1. In fact, we know that the true faction is 0.1, so this is completely as expected. However, it demonstrates how well this method works.

## 3.2 Distributed lasso

The results for ordinary lasso and distributed lasso are equal. We will therefore not address any of these. The strength of distribution is that large problems can divided into smaller sub-problems to be solved in parallel. This can in some cases come with a performance increase; however, it is worth mentioning that parallelizing always comes with some extra cost. In some cases, it is not possible to solve the problem in serial, either because of limitations in memory, but it can also be because the problem is distributed by nature.
The distributed lasso has been implemented in matlab using the parallel computing toolbox. Solving the problem using partitions of 1-8 we achieved the following compute times:

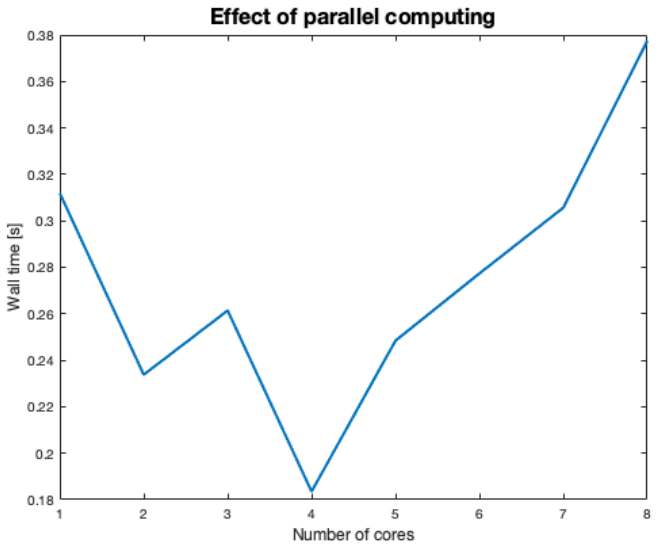


Figure 6: Compute time given a partition number.

In figure 6 we can see that the partition number has a large effect on how quickly the problem can be solved. We can see that increasing the number of partitions slightly produces a speed up, however, at some point the extra overhead eats the performance increase. From the plot, it is evident that using 4 rather than 1 core roughly halves the wall time, which is as expected.
This problem was solved on a laptop with 4 cores, which might be part of the reason that 4 partitions works especially well.

# 4 Conclusion

We have introduced ADMM and demonstrated how it can be used in practise. In general, the ADMM is a very flexible method, that can solve a wide variety of problems. The main concepts of the algorithm are in and by themselves not complicated, and it can in many ways be seen as distributed analogue to the gradient descent algorithm. A large part of the complexity in the algorithm comes from solving the individual optimizing steps, i.e. the alternating directions steps. These need to be convex sub-problem, in our case they largely had analytical solutions, however, this might not generally be the case.
In summary, ADMM is a method than can be used on many different problems, as long as they can be divided into convex sub-problems. It is very likely that many of these problem can be solved faster using other methods or problem specific algorithms, however, it is a good general tool that can be used in any different situations. Especially its distributive properties can become attractive in cases were data is distributed across many platforms.

# References

Boyd, Stephen et al. (Jan. 2011). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3, pp. 1–122. DOI: 10.1561/2200000016.