Solving Large-Scale Optimization Problems with ADMM

Christian Segercrantz, Tuukka Mattlar
Aalto University School of Science, Department of Mathematics and Systems Analysis
{*christian.segercrantz, tuukka.mattlar*}*@aalto.fi*

1 Introduction

Large optimization problems are computationally heavy tasks. Moreover, having to consider multiple varying scenarios of a problem can easily increase the computational time required exponentially. Having efficient methods for solving large problems is therefore essential, as especially stochastic problems tend to quickly grow large in size. These kinds of problems do, however, usually display separability which may allow for algorithm modifications that allow the use of decompositions and parallelization.

This project aims to create an optimization algorithm making use of the alternating direction method of multipliers (ADMM). We will begin by discussing the motivation and theory behind the method and algorithm. Having discussed this, we continue with an in-depth walk-through of the algorithm and its application of it. We will then present our results, including results on time taken and iterations for different penalty term parameters used in the algorithm. Additionally, we compare the results to a classic deterministic method as well as convergence statistics. Finally, we discuss the results and conclusions that can be drawn from them and discuss potential future improvements.

The computations were performed on two separate platforms, these will be referred to as the cloud server and the local setup. The local setup runs Windows 10 build 19043.1319. The central processing unit (CPU) is an Intel Core i7-6700K, which has 4 cores with 2 threads each, overclocked to a clock speed of 4.00GHz. The PC has 16GB of rapid access memory (RAM) clocked at 2667 MHz. The cloud server used is the Aalto CS's JupyterLab server. The detailed information of this server is not available but the authors speculate that it runs a Unix-based operating system. The server is part of the Triton computing cluster. The Interior Point OPTimizer (Ipopt) was used as a solver in all cases.

2 Background

2.1 Augmented Lagrangian method of multiplies

We start by considering the primal problem:

$$\text{min.} f(x) \tag{1}$$

$$\text{st.} Ax = b \tag{2}$$

The general augmented Lagrangian for the problem can be formed as

$$L_\mu(x, v) = f(x) + \sum_{i=1}^{l} v_i h_i(x) + \mu \sum_{i=1}^{l} h_i(x)^2 \tag{3}$$

where (x, v) is the primal-dual pair.

The augmented Lagrangian method of multipliers (ALMM) combines the Lagrangian function and penalty terms with the perks of duality. Here, the penalty term allows for exact convergence with a finite penalty which is caused by the penalty term losing conditions of the dual function. The method of multipliers is basically a dual ascent method, but with the penalty parameter used to determine the step sizes. The alternating direction method of multipliers (ADMM), which is our main algorithm of interest, is a combinatorial variant of the ALMM.

2.2 ADMM

ADMM is the alternating direction version of the ALMM as its primal and dual variables are updated in a sequential way. This separation of the primal and dual variables allows for the decomposition of the whole problem, in case the problem is separable. In fact, we can consider the following problem for the separated version:

$$\min_{x,y} . f(x) + g(y) \tag{4}$$

$$\text{st.} Ax + By = c \tag{5}$$

Compared to a standard linear problem, the problem is split into functions f and g and variables x and y. This means that the problem is separable and allows for decomposition. Applying the ADMM method to optimization problems can provide an efficient solution since it makes use of the separation of the problem which further can allow for parallelization of the problem computation. Further assumptions of the problem will be discussed later.

The problem can be further converted to a augmented Lagrangian form:

$$\phi(x, y, v) = f(x) + g(z) + v^\top(c - Ax - By) + \mu(c - Ax - By)^2.$$

This form of the problem can be solved using ADMM according to the pseudo-code 1 where steps 3, 4, and 5 correspond to updating the variables.

Algorithm 1 ADMM

```
1: Initialize:
       Tolerance $\epsilon > 0$, initial dual and primal solutions $v^0$ and $y^0$, $k = 0$
2: while $|c - A\bar{x}^k - B\bar{y}^k|$ and $||y^{k+1} - y^k|| > \epsilon$ do
3:     $\bar{x}^{k+1} = \text{argmin } \phi_\mu(x, \bar{y}^k, \bar{v}^k)$
4:     $\bar{y}^{k+1} = \text{argmin } \phi_\mu(\bar{x}^k, y, \bar{v}^k)$
5:     $\bar{v}^{k+1} = \bar{v}^k + \mu(c - A\bar{x}^{k+1} - B\bar{y}^{k+1})$
6:     $k = k + 1$
   return $(x^k, y^k)$
```

In the algorithm 1, we first initialize the variables on row 1. Here, we introduce and set the tolerance parameter ϵ with a value greater than 0, declare the initial variables for the primal decision variables x, y, and finally set a counter for following the number of iterations.

In row 2 we start a while loop that waits for the convergence of the primal decision variables with the set tolerance. If the change between recurring iterations becomes small enough, the optima are achieved with the pursued precision.

On rows 3 and 4 we optimize both primal variables separately resulting in alternating direction, a key characteristic of the model. The value for x^{k+1} purely just updates the state based on the previous iteration round so that the values of y^{k+1} and v^{k+1} can be achieved.

Finally, in row 5 we take into account the dual step by computing the dual optima based on the primal optima. Finally, in row 6, the iteration count is increased to keep track of the number of iterations.

2.3 Convergence of ADMM

In order for us to study the convergance of ADMM we need to discuss and state the following assumptions.

1. Functions f and g are convex, proper and closed. [1]
2. For the augmented Lagrangian there exists a saddle point. [1]

Based on the first assumption, we know that the augmented Lagrangian is solvable, i.e. there exists such x and y that minimize it. These values are not necessarily unique. The augmented Lagrangian will be finite for any saddle points (x', y', v') which therefore implies that the saddle point (x', y') is a solution [1]. Based on both assumptions, we know that a solution found from the augmented Lagrangian will fulfill the constraints of the original problem as the functions f and g will be finite. Additionally, strong duality holds for the problem, i.e., the primal and dual optimal solutions will be equal. Hence, the solution found for y will be dual optimal. Due to these constraints, we will know that the (primal) residual, $r = Ax + By - c$, will converge towards 0 as k increases. In other words, the algorithm moves towards feasibility. Secondly, we can state that the original objective function value will also converge towards optimality. Lastly, the dual variable v will converge towards its optimal value.

2.4 Optimality and stopping conditions of ADMM

The necessary and sufficient optimality conditions for alternating direction method of multiplier problems are primal, $h(x) = 0 \implies Ax + By - c = 0$, and dual feasibility, $\nabla f(x) + A^\top v = 0$, $\nabla g(y) + B^\top v = 0$ (in case the functions are differentiable). [1]. Since y^k and v^k always satisfy $\nabla g(y^k) + B^\top v^k = 0$, we only need to look at the two other equations, i.e.

$$Ax + By - c = 0 \tag{6}$$

$$\nabla f(x) + A^\top v = 0. \tag{7}$$

From equation 6, we can derive the residual $r^k = Ax^k + By^k - c$. From Equation 7, we can, by knowing that x^{k+1} minimizes the augmented Lagrangian, transform it into

$$\nabla f(x^{k+1}) + A^\top v^{k+1} + \rho A^\top B(y^k - y^{k+1}) = 0 \tag{8}$$

where we will call $s^{k+1} = \rho A^\top B(y^k - y^{k+1})$ the dual residual. By making these two terms converge to zero, optimality can be archived. The proof that the ADMM algorithm will indeed make both the primal and dual residuals converge to zero is found in the references appendix [1].

It has been seen that ADMM converges quickly close to the optimal solution, but slowly to a very high accuracy solution. Because of this, it's good to have reasonable stopping conditions. As stated earlier, the residuals are strongly tied to the convergence of the algorithm. Thus, basing the stopping criterion on the (scaled) residuals is reasonable.

3 Applications

3.1 The stochastic capacity expansion model

We examine the following stochastic capacity expansion problem.

$$\begin{aligned}
\text{min.} \quad & \sum_{i\in I} C_i x_i + \sum_{s\in S} P_s\Big(\sum_{i\in I}\sum_{j\in J} F_{i,j} y_{i,j,s} + \sum_{j\in J} Q_j u_{j,s}\Big) && && (9)\\
\text{st.} \quad & \sum_{i\in I} C_i x_i \le B && && (10)\\
& \sum_{j\in J} y_{i,j,s} \le x_i, && \forall i \in I, \forall s \in S && (11)\\
& \sum_{i\in I} y_{i,j,s} = D_{j,s} - u_{j,s}, && \forall j \in J, \forall s \in S && (12)\\
& x_i \le B_i, && \forall i \in I && (13)\\
& x_i \ge o, && \forall i \in I && (14)\\
& y_{i,j,s} \ge 0, && \forall i \in I, \forall j \in J, \forall s \in S && (15)\\
& u_{j,s} \ge 0, && \forall j \in J, \forall s \in S && (16)\\
& && && (17)
\end{aligned}$$

Here, I is the set of suppliers, J is the set of clients with unknown demand and S represents the set of scenarios. Moreover, D_{js} is the demand realization, P_s associated scenario probabilities. x_i's are reserving capacity amounts from supplier j and $y_{ij,s}$'s the amount of capacity reserved from supplier i. Further, $u_{j,s}$ are not fulfilled demands in case capacity was not adequate and those come with a unit cost Q_j and $f_{i,j}$'s are the unit costs to fulfill demand. Finally, a budget of B_i capacity is available for all i with a maximum overall budget of B.

3.2 Applying ADMM to the stochastic capacity expansion model

To apply the problem in question to the ADMM algorithm we first identify the separability of the objective function. This allows the use of ADMM instead of ALMM.

In order for us to use the ADMM algorithm, we need to transform the problem into a augmented Lagragian form, as discussed earlier. In the below equation, we can see the different steps of the pseudo-code from earlier. Equation 19 is the objective function that is to be minimized. Instead of the tolerance in the while part, line 2, in the pseudo-code, will we use a modified version seen in Equation 24. This tolerance consists of a scenario probability-weighted sum of the penalty term times the norm of x_s and z.

$$\text{Based on} \qquad \phi(v) = \min \sum_{s \in S} P_s L_s^{\rho}(x_s, y_s, z, v_s) \tag{18}$$

$$\text{min.} \qquad L_s^{\rho}(x_s, y_s, z, v_s) = \sum_{i \in I}((C_i + v_{i,s})x_i + \sum_{j \in J}(F_{i,j}y_{i,j} + Q_j u_j + \frac{\rho}{2}(x_i - z_i)^2)) \tag{19}$$

$$= \sum_{i \in I}(C_i + v_{i,s})x_i + \sum_{i \in I}\sum_{j \in J} F_{i,j}y_{i,j} \sum_{j \in J} Q_j u_j + \sum_{i \in I} \frac{\rho}{2}(x_i - z_i)^2 \tag{20}$$

$$\text{x-step} \qquad x^{k+1} = \text{argmin } L_{\rho}(x, y^k) \tag{21}$$

$$\text{z-step} \qquad z^{k+1} = \sum_{s \in S} p_s x_s^{k+1} \tag{22}$$

$$\text{v-step} \qquad v_s^{k+1} = \sum_{s \in S} v_S^k + \rho(x_S^{+1} - z^{k+1}) \tag{23}$$

$$\text{Tolerance} \qquad \epsilon > \sum_{s \in S} p_s \rho ||x_s^{k+1} - z^k||_2 \tag{24}$$

In more detail, in Equation 18 we introduce the augmented Lagrangian based on the objective function, seen in Equation 9, with a penalty term, as was seen in Algorithm 1.

The x-step in Equation 21 is purely updated based on the last iteration of new z, v values. This step, therefore, updates the x in order to compute the other steps.

Since the z-variable is responsible for the nonanticipativity of the variable x, we simply update it by taking the weighted sum of the newly obtained x-values. This can be seen in Equation 22.

Finally, the v-step is represented in Equation 23 representing the dual solution by utilizing the gradient descent method with a penalty step size ρ.

The tolerance condition for ending, in the Equation 24 is obtained from the sum of the squares of the residuals $||x_s^{k+1} - z^{k+1}||_2^2 + ||z^{k+1} - z^k||_2^2 = ||x_s^{k+1} - z^k||_2^2$, which is then squared. Moreover, the value is multiplied with the penalty step size ρ and the scenario probability p_s is taken into account.

4 Discussion and conclusions

The ADMM algorithm was run using a max iteration limit of $N = 200$, a tolerance of $\epsilon = 10^{-1}$, and a varying ρ. For each iteration, the time taken in seconds and the iteration amount needed was recorded. The algorithm was run on small, medium, and large instances. The instance specifications can be seen in Table 1. The algorithm was run for each instance using $rho \in [0.5, 99.5]$ with a step size of 1. The number of iterations as a function of ρ can be seen in Figure 1. The results of the time, with the deterministic model as a reference, can respectively be seen in Figure 2. During all of the runs, the algorithm converged in less than N iterations, in fact, all runs were completed in less than 7 iterations. All runs for Figures 1 and 2 were done on the cloud setup discussed in Section 1.

Table 1: The specifications used to generate the instances.

	Suppliers	**Clients**	**Scenarios**
Small	15	20	50
Medium	15	20	75
Large	15	20	100

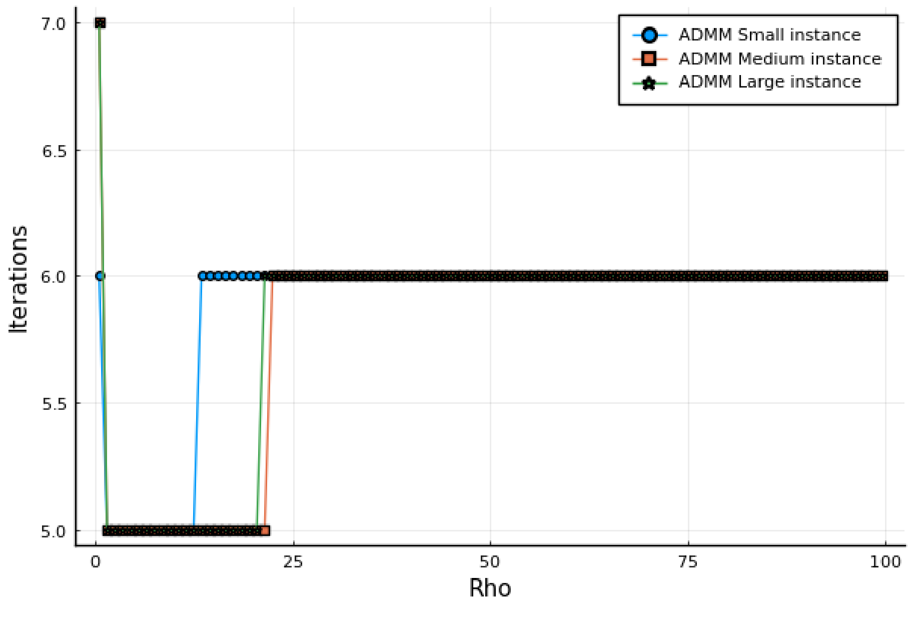

Figure 1: The number of iterations k as a function of ρ. The ADMM ran for values between 0.5 and 95.5 with a step size of 1. The deterministic models as a reference. The simulations were run on the cloud setup.

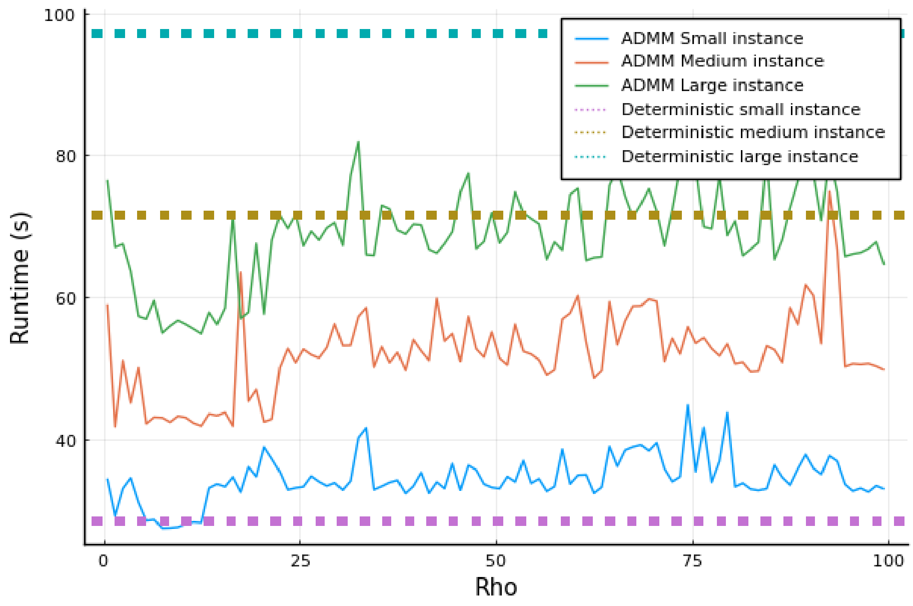

Figure 2: The run time until convergence as a function of ρ. The ADMM ran for values between 0.5 and 95.5 with a step size of 1. The deterministic models as a reference. The simulations were run on the cloud setup.

We begin by examining the number of iterations, visualized in Figure 1. Overall the runs, the algorithm only resulted in three distinct iteration amount values, 5, 6, or 7. For the very lowest values of ρ, the solving took 7 iterations. For values in the range [1.5, 20.5], the medium and large instances needed 5 iterations. The small instance converged with 5 iterations in the interval [1.5,12.5]. For all other runs, the algorithm needed 6 iterations to converge. We can, therefore, say that there is no clear effect of the penalty term ρ on the iteration amount needed for convergence.

Figure 2 displays a similar story. We can see that the time needed for the algorithm to converge does not seem to have much dependence on the size of rho. There seem to be a few numbers that perform more poorly for the algorithm, but this can be contributed to software and hardware-related technical specifications especially since each step was only computed once. The figure additionally displays the deterministic full model. We can see that the ADMM runs mostly fall between the time needed for the deterministic model to solve the small and medium instances. This is a somewhat expected result since the ADMM excels at solving larger problems. We can see that on average the large instance is solved 1.3 times faster when using the ADMM algorithm than when using the deterministic model. However, the small instance is solved, in all but for a few ρ values, faster using the deterministic model.

One of the strengths of ADMM lies in the decomposition of the problem, into separate parts that can be computed separately. In other words, the problem is parallelizable. However, the technical implementation of the algorithm in this project does not make us of that at all. In our code, we iterate through all of the scenarios three separate times for each (except for the last run, which only uses two runs) scenario s. In particular, one of the loops is constantly solving the optimization sub-problem, which is the heaviest part of the code. There is naturally some overhead that is not affected by the parallelization, but for the most part, the code would be sped up by as many threads that could be used for the computation.

An aspect that was not investigated in this work is the effect of the size of the tolerance ϵ. Since the ADMM algorithm quickly converges close to the optimal solution but slowly to a high accuracy solution [1], defining ϵ well could save a lot of time. In real-life scenarios, a close to accurate number is often enough, especially if it speeds up the process by a large portion.

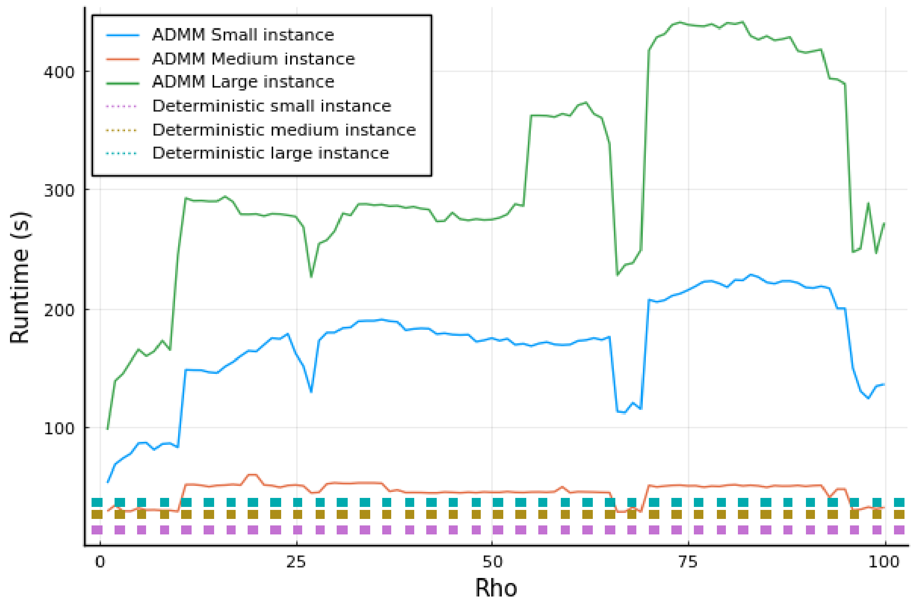

Figure 3: Same conditions as 2, but the simulations were run on the local setup.

Figure 3 shows the same algorithm with the same specifications ran on the local setup discussed in Section 1. We can see that the results vary widely from that of the cloud setup: the run time on ADMM clearly seems to slow down as the function of ρ and all the ADMM solutions are around a magnitude slower than their deterministic ones. Additionally, the local setup solves the deterministic models way faster than the cloud setup. This indicates that knowledge of hardware, software, and solver specifications are important to know in order to determine if the choice of algorithm is good. For the local setup case, it does not seem to be worth building an extensive ADMM model that is slower than the full deterministic model in the studied case.

References

[1] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers.* Now Publishers Inc, 2011.