Solving Large-Scale Optimization Problems with ADMM

Jonatan von Martens, Silja Sormunen
Aalto University School of Science, Department of Mathematics and Systems Analysis
Aalto University School of Science, Department Computer Science
{*jonatan.vonmartens, silja.sormunen*}*@aalto.fi*

1 Background

Alternating Direction Method of Multipliers (ADMM) combines the benefits of the methods it is based on, namely the dual ascent method and the method of multipliers [1]. Dual ascent allows for dual decomposition; if the problem has a decomposable structure, minimisation of the decision variables can be carried out in parallel, after which the solved values are collected to update the dual variables. However, the method converges only under some rather specific assumptions about the properties of the objective function. Method of multipliers is more robust, and it converges even if the objective function is not strictly convex. The main drawback of the method, however, is that the added penalty term in the augmented Lagrangian prevents decomposition of the function and parallel minimisation of the decision variables.

ADMM decomposes the problem at hand into smaller subproblems, which can be solved concurrently. Variables in the objective function as well as their corresponding constraints are decomposed into separate parts. ADMM can solve problems of the form:

$$\begin{aligned} &\min f(x) + g(z) \\ &\text{s.t.} Ax + Bz = c \end{aligned}$$

As in method of multipliers, the augmented Lagrangian is formed, after which the optimal solution is solved iteratively. The algorithm proceeds by finding the x-value, which minimises the augmented Lagrangian, based on previous values of z and y, after which the value of z is updated based on the obtained value of x. Finally, the dual variables are updated with Gradient Descent using the multiplier ρ of the penalty term as a step size. This ensures that the dual variables corresponding to the z-values stay feasible. The aim is to find the optimal dual values v, which maximize the augmented Lagrangian dual function.

Pseudocode of ADMM is shown in Listing 1:

Listing 1: ADMM pseudocode

```
for k in max_iterations:
    for s in scenarios:
        (x_s^{k+1}, y_s^{k+1}) = arg min_{x_s, y_s} L_s^ρ(x_s, y_s, z^k, v_s)
        z^{k+1} = Σ_s p_s x_s^{k+1}
        v_s^{k+1} = v_s^k + ρ(x_s^{k+1} − z^{k+1})
        if Σ_s p_s ρ||x_s^{k+1} − z^k||_2 < ε:
```

```
7                 return x_s
8         return x_s
```

As discussed in [**1**], the iterates approach feasibility, the objective function of the iterates approaches the optimal value, and the dual variables converge to their optimal values as the method proceeds. Obtaining high accuracy can require many iterations, but a few tens of iterations suffices often to obtain intermediate accuracy [**1**]. In general, ADMM converges more slowly than the method of multipliers.

At optimality, the solution is both primal and dual feasible. Assuming that f and g are differentiable - which, however, is not a requirement of ADMM - the optimality conditions can be formulated as follows [**1**]:

$$Ax + Bz - x = 0$$
$$\nabla f(x) + A^T y = 0$$
$$\nabla g(z) + B^T y = 0$$

As the step size is set equal to the penalty parameter, the third condition stays true throughout the iterative process. The algorithm stops, when the primal residual $r^k = Ax^{k+1} + Bz^{k+1} - c$ and the dual residual $s^{k+1} = \rho A^T B(z^{k+1} - z^k)$ are close enough to zero. As shown in [**1**], residuals converging to zero correspond to small objective suboptimality. Different criteria for choosing the tolerance threshold can be used.

In the present implementation, the stopping condition for the algorithm is either when a maximum number of iterations is reached, which is fixed to 200 in the code, or when the non-squared sum of primal and dual residuals multiplied by ρ (described on line 6 in the pseudo code) is below the tolerance (ϵ) of 0.1. We use the penalty term ρ multiplied by the non squared residual as the stopping condition in order to make the convergence smoother.

The project was run on the CS Jupyterlab server, which has 32 Intel Xeon E5-2665 8-core processors clocked at 2.40GHz. The server also has 128891MB of RAM, and is running Ubuntu Bionic Beaver 18.04.2 LTS (Long term support).

2 Applications

In this project, a variant of the ADMM is implemented and used to solve a large-scale stochastic capacity expansion problem, in which the objective is to reserve minimum cost capacity amounts x_i from suppliers $i \in I$ so that realised demands d_{js} of customers $j \in J$ in future scenarios $s \in S$ can be fulfilled. Each scenario is realised with probability p_s. The amount of capacity reserved from supplier i used to fulfil the demand of client j in scenario s is denoted with y_{ijs}, and the amount of demand of client j left unfulfilled in scenario s is denoted with u_{js}. The unfulfilled amounts are penalized at a unit cost q_i, while a unit cost f_{ij} is associated with fulfilling the demand of client j using supplier i. The maximum capacity that can be reserved from supplier i is denoted with b_i, and B denotes the budget available for capacity reservations. The problem is presented below:

$$
\begin{aligned}
\min \quad & \sum_{i \in I} c_i x_i + \sum_{s \in S} p_s \Big(\sum_{i \in I} \sum_{j \in J} f_{ij} y_{ijs} + \sum_{j \in J} q_j u_{js} \Big) \\
\text{subject to} \quad & x_2 \leq (1 - x_1)^3 \\
& \sum_{i \in I} c_i x_i \leq B \\
& \sum_{j \in J} y_{ijs} \leq x_I, \quad \forall i \in I, \forall s \in S \\
& \sum_{i \in I} y_{ijs} = d_{js} - u_{js}, \quad \forall j \in J, \forall s \in S \\
& x_i \leq b_i, \quad \forall i \in I \\
& x_i \geq 0, \quad \forall i \in I \\
& y_{ijs} \geq 0, \quad \forall i \in I, \forall j \in J, \forall s \in S \\
& u_j s \geq 0, \quad \forall j \in J, \forall s \in S
\end{aligned}
$$

As in the course exercise 10, the problem can be reformulated using copy variables x_s for each scenario s, which allows for decomposing the problem into several sub problems, or scenarios, where all $s \in S$ can be solved in parallel:

$$
\begin{aligned}
\min_{x,y,z} \quad & \sum_{s \in S} p_s (c^T x_s + \sum_{i \in I} \sum_{j \in J} f_{ij} y_{ijs} + q^T u_s) \\
\text{subject to} \quad & x_s = z
\end{aligned}
$$

The nonanticipativity constraint $x_s = z$ ensures that the scenario-independent vector x does not vary across final solutions of the subproblems. Next, the augmented augmented Lagrangian dual function is formed as described in section 1:

$$\begin{aligned}\phi(\mu) = \quad & min_{x,y,z} \sum_{s\in S}[p_s(c^T x_s + \sum_{i\in I}\sum_{j\in J} f_{ij}y_{ijs} + q^T u_s) + \mu_s^T(x_s - z) + p_s\frac{\rho}{2}||x_s - z||_2^2] \\ \text{s.t.} \quad & \text{the original constraints}\end{aligned}$$

Denoting $v_s = \mu_s/p_s$ for all s ∈ S enables us to rewrite the problem:

$$\begin{aligned}\phi(v) = \quad & min_{x,y,z} \sum_{s\in S} p_s L_s^\rho(x_s, y_s, z, v_s) \\ \text{s.t.} \quad & \text{the original constraints}\end{aligned}$$

where $L_s^\rho(x_s, y_s, z, v_s) = c^T x_s + \sum_{i\in I}\sum_{j\in J} f_{ij}y_{ijs} + q^T u_s + v_s^T(x_s - z) + \frac{\rho}{2}||x_s - z||_2^2$.

In order to prevent $\phi(v)$ from diminishing to negative infinity, it needs to be the case that $v_s^T z = 0$ for all $s \in S$. Thus, we can rewrite:

$$L_s^\rho(x_s, y_s, z, v_s) = (c + v_s)^T x_s + \sum_{i\in I}\sum_{j\in J} f_{ij}y_{ijs} + q^T u_s + \frac{\rho}{2}||x_s - z||_2^2]$$

As described in the introduction, ADMM proceeds by first updating the values (x_s^k, y_s^k). This update can be done parallel for each scenario s, as the function $\phi(v)$ is expressed separately for each scenario. As described in the pseudocode presented in section 1, the values are updated by finding the values (x_s^{k+1}, y_s^{k+1}) minimizing the augmented Lagrangian.

$$\begin{aligned}(x_s^{k+1}, y_s^{k+1}) &= \min . L_s^\rho(x_s, y_s, z^k, v_s^k) \\ &= \min . (c + v_s)^T x_s + \sum_{i\in I}\sum_{j\in J} f_{ij}y_{ijs} + q^T u_s + \frac{\rho}{2}||x_s - z||_2^2]\end{aligned}$$

After the updated values (x_s^{k+1}, y_s^{k+1}) are computed, z-value is updated based on the obtained values. This step combines the results of the parallel updates of the previous step.

$$\begin{aligned}z_{k+1} &= \min . \sum_{s\in S} p_s L_s^\rho(x_s^{k+1}, y_s^{k+1}, z^k, v_s^k) \\ &= \min . \sum_{s\in S} p_s[(c + v_s^k)^T x_s + \sum_{i\in I}\sum_{j\in J} f_{ij}y_{ijs} + q^T u_s + \frac{\rho}{2}||x_s - z||_2^2]\end{aligned}$$

The formula for the z-update on line 4 of the pseudocode in Listing 1 is obtained by setting the gradient of the above presented equation to zero and solving for z^{k+1}, which results in:

$$z^{k+1} = \sum_{s\in S} p_s x_s^{k+1}$$

Next, the dual variables v_s are computed in parallel for each scenario s using Gradient Descent with step size ρ.

$$v_s^{k+1} = v_s^k + \rho(x_s^{k+1} - z^{k+1})$$

As discussed in section 1, the algorithm stops when primal and dual residuals are below a chosen tolerance threshold. Here, the sum of the two squared residuals has the form:

$$\sum_{s \in S} p_s[||x_s^{k+1} - z^{k+1}||_2^2 + ||z_s^{k+1} - z^k||_2^2],$$

which, as shown in exercise 10, can be transformed into the form:

$$\sum_{s \in S} p_s||x_s^{k+1} - z^k||_2^2$$

However, in order to obtain smoother convergence, we use a stopping criterion of the form:

$$\sum_{s \in S} p_s\rho||x_s^{k+1} - z^k||_2$$

In the present implementation, four parallel threads are used. We solve two problems, a smaller one with 100 different scenarios and a larger one with 500 scenarios. Both problems consider 20 suppliers and 30 clients, and price and demand values are chosen randomly. The maximum number of ADMM iterations is set to 200. The convergence speed of the models are compared to deterministic models.

3 Discussion and conclusions

The small deterministic model converges in 37 iterations, while the large model converges in 76 iterations.

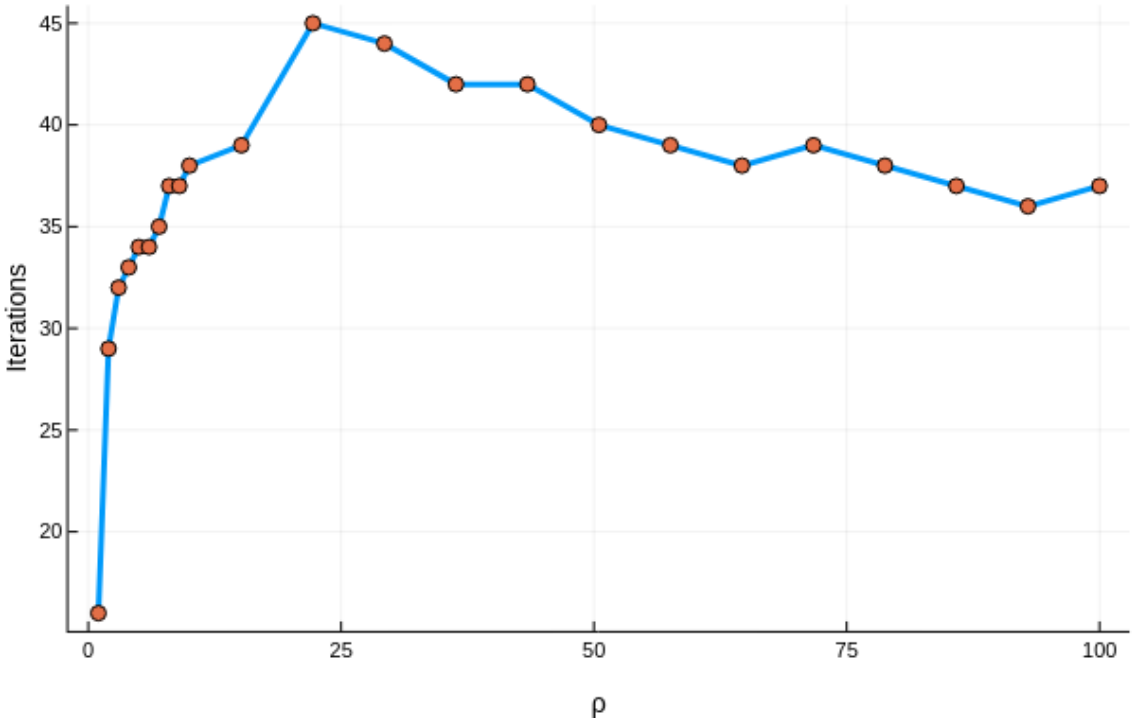

Figure 1: Effect of penalty term on number of iterations for small model

The small ADMM model was tested with 23 different penalty terms ρ ranging from 1 to 100, which are displayed in Figure 1. The figure shows that smaller penalty terms make the model converge much faster than large penalty terms even though the number of iterations required for convergence seem to reduce somewhat for penalty terms higher than 25. All in all, the penalty term should be set to a value smaller than 10 to obtain faster convergence than observed in the deterministic model.

The large ADMM model was also tested with 23 different penalty terms ρ ranging from 1 to 100, which are displayed in Figure 2. Similarly as in the small model, smaller values of ρ correspond clearly to less iterations. Here, however, values of ρ close to 100 require more iterations than values around 25, contrary to the smaller model.

As the algorithm stops when

$$\sum_{s \in S} p_s \rho ||x_s^{k+1} - z^k||_2 < \epsilon,$$

smaller values of ρ lead to smaller left-hand-side values, thus leading to faster convergence.

As dicussed in [1], large values of ρ penalize violations of primal feasibility harshly, while small values of ρ tend to penalize violations of dual feasibility. This is evident from the formulations of

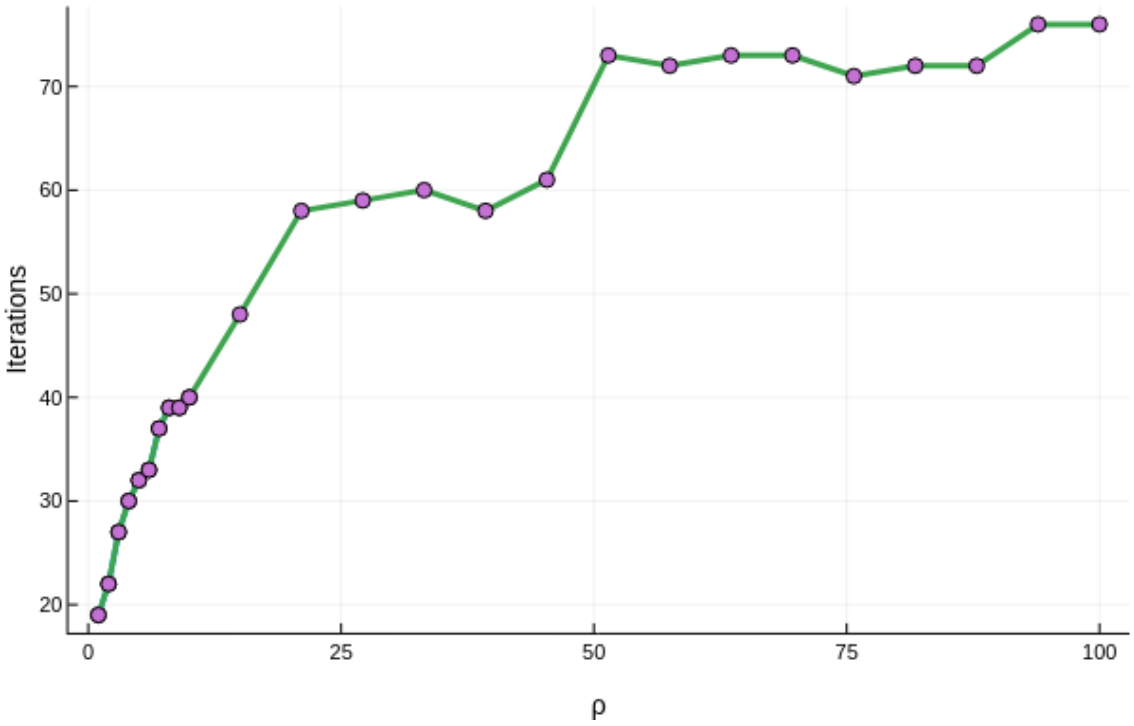

Figure 2: Effect of penalty term on number of iterations for large model

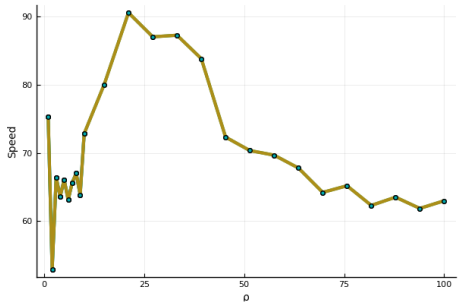

Figure 3: Execution speed as a function of ρ for the small model

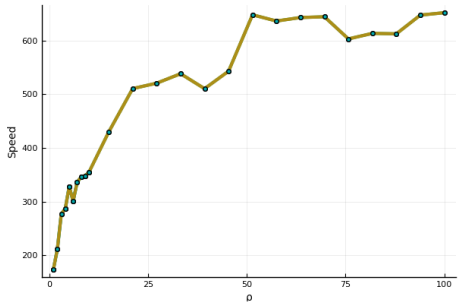

Figure 4: Execution speed as a function of ρ for the large model

the augmented Lagrangian and the dual residual s^{k+1} presented in the introduction.

Finally, the required execution time is shown in Figures 3 and 4 as a function of the penalty parameter ρ. In general, the trend is similar to that observed with respect to the number of iterations, especially for the larger model. In the smaller model the execution times for small values of ρ vary and stay in many cases larger than for values of ρ close to 100. As the model is small and the execution times are shorter than for the large model, the results for the smaller model might suffer from some inaccuracy.
As the models where run on the jupyter server, server load might affect the execution speed results, and can not be considered completely reliable.

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3 (Jan. 2011), pp. 1–122. DOI: 10.1561/2200000016.