SF3847 Convex Optimization with Engineering Applications – Homework Assignment 4

Kaige Tan (960612-8396, kaiget@kth.se)

May 04, 2021

Exercise 4.1 maxsumlog

Part a)

The optimization problem is:

$$\max_{\{x\}} \quad f_0(x) = \sum_i log(1 + x_i c_i) \tag{1a}$$

$$\text{s.t.} \quad h_1(x) = Ax \leq b, \tag{1b}$$

$$h_2(x) = x \geq 0, \tag{1c}$$

The algorithm is implemented with a primal-dual interior method to search for the optimal solution. The source code of the algorithm can be checked in **maxsumlog.m** and performance comparison in **testmaxsumlog.m**. The details realization of the primal-dual interior-point method can be summarized as follows:

Start with a strictly feasible point $x^{(0)}$ and $\lambda^{(0)} > 0$, where x and λ are primal and dual variables. Define $\eta^{(0)} = h(x^{(0)})^T \lambda^{(0)}$. Then for a barrier parameter $\mu > 1$, repeat for $k = 1, 2, 3...$

- Define $t = \mu m / \eta^{(k-1)}$
- Compute primal-dual update direction $\Delta y = \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix}$
- Determine step size s
- Update $y^{(k)} = y^{(k-1)} + s\Delta y = \begin{bmatrix} x^{(k)} \\ \lambda^{(k)} \end{bmatrix}$

- Compute $\eta^{(k)} = h(x^{(k)})^T \lambda^{(k)}$
- Stop if $n^{(k)} \leq \epsilon$, where $\epsilon = 1e^{-}5$ as the required precision.

In the algorithm, Newton's method is applied, derived from a perturbed version of the KKT conditions, to calculate update direction Δy (i.e., the Newton step). The detailed formulation can be found at Eq. (11.54) in [1].

After getting the update direction, a line search is performed to determine a proper step size s. The initial maximal step is set as $s^{max} = 1$. The value update is iterated in a while loop in the code to search the maximal step until $h(x_{new}) \geq 0$ and $\lambda_{new} \geq 0$.

- *"Assume that $c \in \mathbb{R}^m_+$, $A \in \mathbb{R}^{n \times m}_+$ and $b \in \mathbb{R}^n_{++}$. Show that these assumptions make it easy to find a feasible interior starting point."*

These assumptions define the domain of A, b and c. Since A, c are non-negative and b is strictly positive, I can easily choose $x = \mathbf{0}$ as the initial feasible point.

Part b)

The given routine is applied to test the implementation. The seed for the MATLAB random number generator is specified as 1 for reproduction. Problems with different dimensions, including $\{m = 10, n = 2\}$, $\{m = 10, n = 15\}$, $\{m = 100, n = 5\}$, $\{m = 100, n = 20\}$, are evaluated. The performance is compared with the CVX toolbox in Table 1.

It worth noting that the initial points (e.g., $x^{(0)}$, $\lambda^{(0)}$) and hyperparameters selection (e.g., μ, ϵ) of the **maxsumlog** algorithm is quite random. It can be assumed that the algorithm's performance can be varied significantly with the parameter tuning process. For example, within a given problem with fixed objective function and constraints, if the initial feasible point is proximate to the optimal point, it takes fewer steps and computation time for searching. However, I only focus on an abstract problem with arbitrary values in the coefficient. Thus the characteristics of matrices cannot be determined and analyzed. For this reason, the performance variation due to those factors will not be discussed in the report.

problems with different dimensions		computation time (s)	relative difference			number of iterations	number of non-zeros in x
			x	objective	y		
{m=10, n=2}	maxsumlog	0.580	0.161	-0.114	0.072	893	***5***
	CVX	0.554				-	-
{m=10, n=15}	maxsumlog	0.010	0.141	-0.003	0.003	38	7
	CVX	0.854				-	-
{m=100, n=5}	maxsumlog	1.308	0.023	-0.001	0.001	637	***13***
	CVX	3.112				-	
{m=100, n=20}	maxsumlog	0.662	0.152	-0.006	0.003	214	11
	CVX	5.258				-	

Table 1: performance comparison between maxsumlog and CVX toolbox. m: number of variables, n: number of constraints

Part c)

Based on the results from Table 1, it can be noticed that with the increase of problem dimensions, it takes a longer time to finish the computation. In general, the maxsumlog outperforms the CVX toolbox, especially with a greater dimensional problem. However, the computation time is not proportional to the number of iterations, which indicates that, for different problems, the computational complexity to solve a single iteration is different. The computation time is largely influenced by the dimensions of the problem when computing primal-dual update direction Δy. Moreover, some constraint matrices may require more iterations to find a proper step size s, and this can also contribute to a longer computation time. The non-zero values in the optimal solution correspond to the basis variables. Theoretically, the number of non-zero values should equal the number of constraints (less than constraints if degenerates). In the MATLAB implementation, it violates the results in $\{m = 10, n = 2\}$ and $\{m = 100, n = 5\}$ scenarios. One possible reason is that the precision of the calculation is not good enough.

Exercise 4.4 optimal generator dispatch

(a) Price decomposition

For the optimal generator dispatch problem, one of the requirements specifies that the total generated power in each period must equal the demand. This set of constraints couple the different groups of variables (i.e., optimal power for different generators). It can be solved by *decomposition*, where the original optimization problem can be broken up into smaller ones, and each

subproblem can be solved separately. It is mentioned in the problem that, the revenue generated by the sale of power at the prices (Q_t) is introduced as a new variable when the problem is solved independently of each generator. This operation can be viewed as a dual decomposition approach. In dual decomposition, when adding the partial Lagrangian variable to relax the coupling constraint, the introduced variable can be interpreted as a set of prices for the resources. To find the prices Q_t, I took it as a variable in the dual optimization problem. The subproblems can be solved with initial random values of Q_t to obtain optimal solutions. Next, the subgradient method can be applied for the master problem and update the dual variable Q_t. The updated Q_t will be applied again to obtain optimal solutions in subproblems. The update mentioned above works iteratively, and the iterations will terminate when the gap satisfies a given threshold. The detailed realization of the algorithm is not elaborated here but can be found in [2] and [3].

(b) Solve with the given data

The problem is solved in MATLAB, and the source code can be found in **gen_dispatch_data.m**. The threshold of the maximum gap (max_{thres}) in the solution is set to 0.01. Furthermore, the maximal rounds of iteration are 50. In order to speed up the convergence, I set up a dynamic step size (ss) for the dual variable update after several rounds of trial and error, where:

$$ss = \begin{cases} 0.05 & \text{if } max_{thres} > 1 \\ 0.01 & \text{if } 1 \geq max_{thres} > 0.5 \\ 0.005 & \text{if } 0.5 \geq max_{thres} \end{cases} \tag{2}$$

Figure 1 shows the given demand, the optimal solution for each generator powers and the power prices.

After obtaining the power prices, the validation is performed to solve the optimization problem separately. Figure 2 gives an example of the optimal solution of the first generator. It compares the differences between the solution by dual decomposition and solution with Q_t. It can be confirmed that the result is correct without noticeable numerical errors.

- *"Comment on anything you see in your solution that might at first seem odd."*

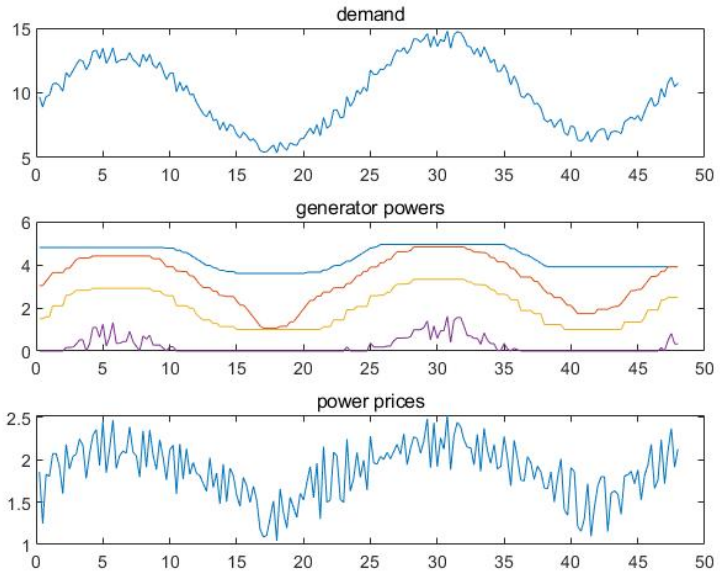

Figure 1: Demand, optimal generator powers, and prices

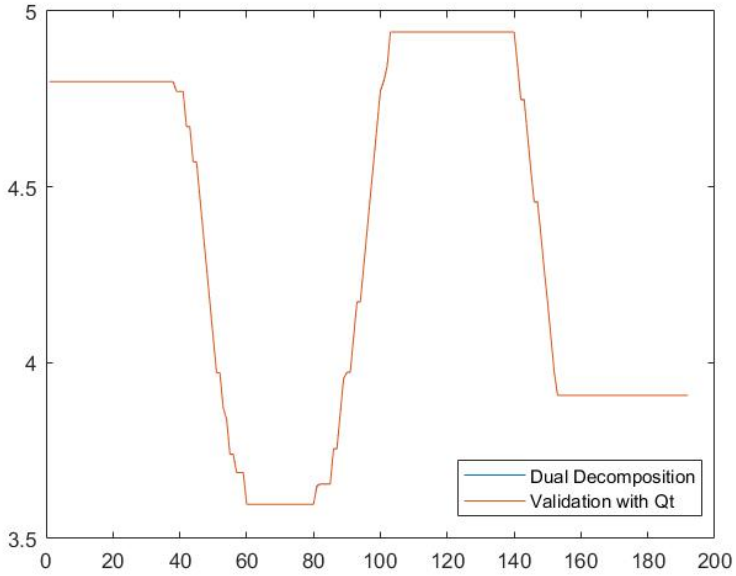

Figure 2: Comparison between optimal power solution solved by dual decomposition and separately optimized with Q_t

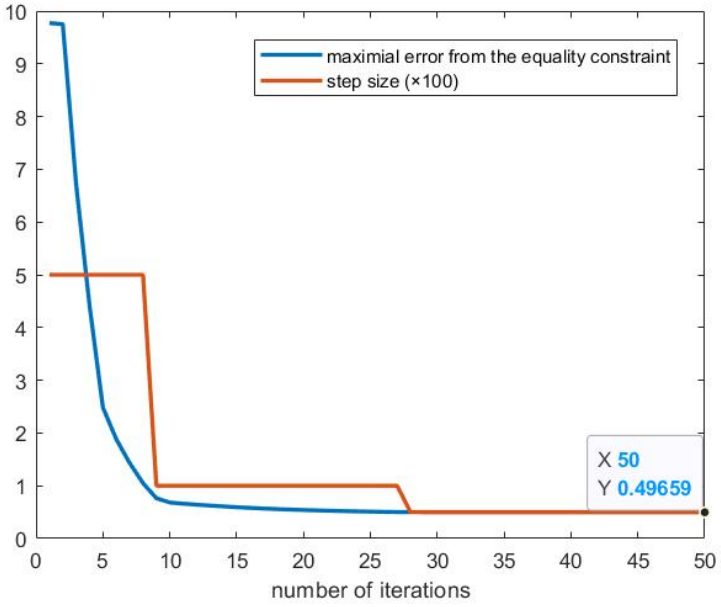

Figure 3: Comparison between maximal error from the resource equality constraint and dual variable update step size (×100)

The only odd thing I found in my solution is, during the optimization iterations, the maximal gap between the solution and the resource equality constraint cannot converge to a significant satisfying value (e.g., $1e^{-5}$), no matter how I was tuning the step size. Figure 3 gives an illustration of the maximal error from the resources equality constraints at each iteration and corresponding step size. The value of step size in the figure is multiplied by 100 for scaling. It can be noticed that the residual error gets stable at around 0.5, and the improvement is tiny during further optimization. Further investigation can be focused on the design of adaptive step size.

References

[1] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] Daniel Pérez Palomar and Mung Chiang. "A tutorial on decomposition methods for network utility maximization". In: *IEEE Journal on Selected Areas in Communications* 24.8 (2006), pp. 1439–1451.

[3] Stanford. *Lecture 9 | Convex Optimization II (Stanford)*. URL: https://www.youtube.com/watch?v=Kwli6FkYQYY. (accessed: 04.05.2021).