ECE 1505 Convex Optimization
Winter 2020

Homework Set #4

1. **SDP Relaxation and Heuristics for the Two-Way Partitioning Problem**

   (a) Problem 5.39 of Boyd and Vandenberghe.

   (b) Problem 11.23(b-d) of Boyd and Vandenberghe.

   Test the effectiveness of various solutions using three W's defined in a file hw4data.mat available on the course website. The matrices for the small- ($n = 5$), medium- ($n = 10$) and large-size ($n = 50$) problems are named `W5`, `W10` and `W50`.

   Please fill in the following table (except the $\times$ slot) with objective values found for problems of different sizes and with different methods. **20 ite**

	SDP Bound	Optimum	11.23(b)	11.23(c)	11.23(d-a)	11.23(d-b)	11.23(d-c)
small	**5.334**	**5.334**	**5.334**	**5.334**	**5.334**	**5.334**	**5.334**
medium	**-42.23**	**-38.37**	**-33.42**	**-38.37**	**-27.459**	**-38.37**	**-38.37**
large	**66.086**	$\times$	**679.61**	**462.18**	**604.637**	**344.734**	**314.095**

   Please use exhaustive search to find the "optimum". For 11.23(c), use $K = 100$ and find the best among the 100. For 11.23(d-a), please use 100 random starting points chosen uniformly among 2^n sequences in $\{+1, -1\}^n$, and choose the best among them. For 11.23(d-c), instead of following the instruction in the book, please use 100 random starting points chosen according to the distribution defined in 11.23(c), do greedy search on all, and find the best among them.

   Please submit your `MATLAB` code. Please comment on the effectiveness of various SDP relaxation methods. Also comment on the tightness of the SDP bound.

   To numerically solve an SDP, you may use a software package `CVX`, which runs under `MATLAB` and is available at `http://www.stanford.edu/~boyd/cvx/`. `CVX` uses an SDP solver called `SeDuMi`. As an example, after installing `CVX`, the following `MATLAB` code segment solves the SDP problem (11.66) in Boyd and Vandenberghe.

```
load('hw4data.mat');
W = W5;
[n,n] = size(W);
cvx_begin
      variable v(n);
      maximize(-sum(v));
```

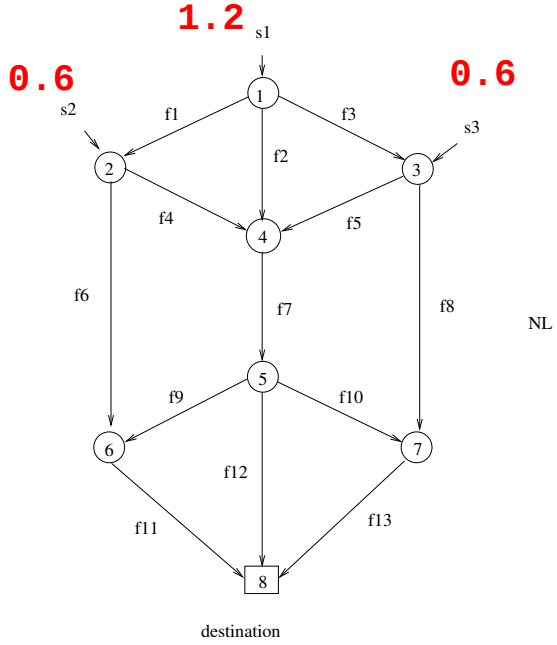
```
        subject to
              W + diag(v) == semidefinite(n);
    cvx_end
    opt_value = -sum(v)
```

You may find the CVX package useful for your project. For more information, please check the user guide available at the above website.

2. **Interior-Point Method**

In this problem, you are asked to implement an interior-point method for a network optimization problem known as multicommodity flow. Consider a network modeled as a directed acyclic graph shown below. The network has $N = 8$ nodes and $L = 13$ directed links. The capacity of each link is $c_l = 1$. The objective is to support some fixed source rates: $s_1 = 1.2$, $s_2 = 0.6$ and $s_3 = 0.6$ (originating from nodes 1, 2 and 3, respectively) to a common destination (node 8) with minimal delay by intelligently assigning flow rates $(x_1, \cdots, x_{13})$.

We model each link as an $M/M/1$ queue. The queuing delay on each link for an $M/M/1$ queue can be expressed as $x_l/(c_l - x_l)$, where c_l is the link capacity and x_l is the flow rate on the link. (As expected, the delay is always positive; delay increases as x_l approaches c_l.) The total delay of the network can be expressed as $\sum_l x_l/(c_l - x_l)$.



NL

The flow rate assignment must satisfy several constraints. First, the flow rate on each link cannot exceed the link capacity. Thus, $0 \leq x_l \leq c_l$.

Second, the network must satisfy a flow conservation constraint on each node. For example, for node 1, we must have $s_1 = x_1 + x_2 + x_3$. Likewise, for node 2 we must have $s_2 + x_1 =$

$x_4 + x_6$, and so on. The flow conservation constraint can be represented compactly as a matrix equality constraint. Define a so-called node-link incidence matrix A, with N rows and L columns as follows:

$$A = \begin{cases} 1 & \text{if } n \text{ is the start node for link } l \\ -1 & \text{if } n \text{ is the end node for link } l \\ 0 & \text{else} \end{cases}$$

The flow conservation constraint can be represented as $A^+\mathbf{x} = \mathbf{s}$ where A^+ is a $(N-1) \times L$ matrix which equals to A with the last row removed, the link capacity $\mathbf{c} = [c_1, c_2, ..., c_L]^T$ is a constant $L \times 1$ column vector, and the source rates $\mathbf{s} = [s_1, s_2, ..., s_{N-1}]^T$ is a constant $(N-1) \times 1$ column vector. For this problem, $s_n = 0$ except for $n = 1, 2, 3$.

Third, in this problem we further constrain the "switching capacity" of nodes 2, 3 and 5 so that the sum of the flow rates on the outgoing links emanating from each of these nodes must be less than or equal to 1. Mathematically, this means that $x_4 + x_6 \leq 1$, $x_5 + x_8 \leq 1$ and $x_9 + x_{12} + x_{10} \leq 1$. You may represent this as a matrix inequality $B\mathbf{x} \leq \mathbf{b}$, where B is an appropriately defined matrix and $\mathbf{b} = [111]^T$.

Thus, the minimum-delay problem is that of finding $[x_1, \cdots, x_{13}]$ to solve the following optimization problem:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_l \frac{x_l}{c_l - x_l} \\ \text{subject to} & A^+\mathbf{x} = \mathbf{s} \\ & B\mathbf{x} \leq \mathbf{b} \\ & c_l \geq x_l \geq 0, \quad \forall l \end{array}$$

Here A^+, B, c_l, $\mathbf{s}$, $\mathbf{b}$ are all constants. The variables are $x_1, \cdots, x_{13}$.

(a) Verify that the minimum-delay problem is a convex optimization problem.

(b) Use an interior-point method to find the optimal $x_1, \cdots, x_{13}$. Please use a log barrier function for inequality constraints and use the infeasible-start Newton's method for the equality constraint. Please write down the relevant gradient, Hessian, and update equations explicitly. What is the minimal delay?

(c) Produce a plot of delay vs. the outer-loop iterations (i.e., every time you update t in the barrier function).

(d) Produce a plot of residue (defined as the sum of the norm of residue of the infeasible-start Newton's method and the residue associated with each outer iteration in the interior-point method, namely m/t, i.e., residue $= ||r||_2 + m/t$) in log scale vs. the number of inner-loop iterations (i.e., every time you update the flow rate variables). Submit your `MATLAB` code.

(Textbook reference: §10.3 and §11 of Boyd and Vandenberghe.)

Algorithm 10.2 *Infeasible start Newton method.*

given starting point $x \in \mathbf{dom}\, f$, ν, tolerance $\epsilon > 0$, $\alpha \in (0, 1/2)$, $\beta \in (0, 1)$.
repeat
1. Compute primal and dual Newton steps Δx_{nt}, $\Delta\nu_{\text{nt}}$.
2. *Backtracking line search on* $\|r\|_2$.
   $t := 1$.
   while $\|r(x + t\Delta x_{\text{nt}}, \nu + t\Delta\nu_{\text{nt}})\|_2 > (1 - \alpha t)\|r(x, \nu)\|_2$, $t := \beta t$.
3. *Update.* $x := x + t\Delta x_{\text{nt}}$, $\nu := \nu + t\Delta\nu_{\text{nt}}$.

until $Ax = b$ and $\|r(x, \nu)\|_2 \leq \epsilon$.

minimize $f_0(x) + \sum_{i=1}^m I_-(f_i(x))$
subject to $Ax = b$, (11.3)

where $I_- : \mathbf{R} \to \mathbf{R}$ is the indicator function for the nonpositive reals,

$$I_-(u) = \begin{cases} 0 & u \leq 0 \\ \infty & u > 0. \end{cases}$$

$\therefore \sum_{i=1}^{m} I_-(f_i(x)) = 0$ for $f_i(x) \leq 0$ (constraint)

Barrier method

given strictly feasible x, $t := t^{(0)} > 0$, $\mu > 1$, tolerance $\epsilon > 0$.
repeat
1. *Centering step.* Compute $x^*(t)$ by minimizing $tf_0 + \phi$, subject to $Ax = b$.
2. *Update.* $x := x^*(t)$.
3. *Stopping criterion.* **quit** if $m/t < \epsilon$.
4. *Increase* t. $t := \mu t$.

Infeasible start Newton method

In one variation on the barrier method, an infeasible start Newton method (described in §10.3) is used for the centering steps. Thus, the barrier method is initialized with a point $x^{(0)}$ that satisfies $x^{(0)} \in \mathbf{dom}\, f_0$ and $f_i(x^{(0)}) < 0$, $i = 1, \ldots, m$, but not necessarily $Ax^{(0)} = b$. Assuming the problem is strictly feasible, a full Newton step is taken at some point during the first centering step, and thereafter, the iterates are all primal feasible, and the algorithm coincides with the (standard) barrier method.

$r(x, \nu) = (r_{\text{dual}}(x, \nu), r_{\text{pri}}(x, \nu))$.

Here

$$r_{\text{dual}}(x, \nu) = \nabla f(x) + A^T\nu, \qquad r_{\text{pri}}(x, \nu) = Ax - b$$

Algorithm 10.1 *Newton's method for equality constrained minimization.*

given starting point $x \in \mathbf{dom}\, f$ with $Ax = b$, tolerance $\epsilon > 0$.
repeat
1. Compute the Newton step and decrement Δx_{nt}, $\lambda(x)$.
2. *Stopping criterion.* **quit** if $\lambda^2/2 \leq \epsilon$.
3. *Line search.* Choose step size t by backtracking line search.
4. *Update.* $x := x + t\Delta x_{\text{nt}}$.