5. *(Coding problem)* In this problem, we consider an inequality form LP

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \ & c^\top x \\ \text{s.t.} \ & Ax = b, x \succeq 0 \end{aligned} \tag{5}$$

where $A \in \mathbb{R}^{m\times n}$, with $m < n$.

Assumption: Throughout this problem, we will assume that A has full row rank, and the sublevel sets $\{x : Ax = b, x \succeq 0, c^\top x \leq \gamma\}$ are bounded when non-empty. (If this is not the case, the centering problems we consider below will be unbounded. We will only be testing your code on instances that satisfy this assumption, so you don't have to worry about it.)

(a) (15 points) Consider the centering problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \ & c^\top x - \sum_{j=1}^{n} \log x_j \\ \text{s.t.} \ & Ax = b \end{aligned} \tag{Centering}$$

Implement feasible-start Newton's method for solving this problem in a file named `LPcentering.m`, with the first line (outside of any comments) `function [xc,valc]=LPcentering(A,b,c,x0)`. Most inputs are self-explanatory (`x0` is a strictly feasible initial point, which in this problem we assume exists and is given), and the outputs are, in order: (i) (approximate) primal optimal solution for the problem (Centering), expressed as a column vector, and (ii) the column vector of objective function values of (Centering) at each iterate, with the last component being the approximate optimal value of (Centering). As a stopping criterion, you should use, $\lambda(x)^2/2 \leq \epsilon$, for $\epsilon \leq 10^{-6}$. This function will be used as a subroutine in another function, so after you've finished debugging, make sure it does not print anything to the screen.

As you can guess, this is a warm-up for implementation of the barrier method for solving problem (LP). In light of this, we recommend that you name the variable for the stepsize at every iteration of Newton's method something other than `t` (e.g., `stepsize`), so it doesn't get confused with the parameter `t` used in the centering problems of the barrier method.

After testing your implementation on a variety of randomly generated instances, generate an instance with $n = 100$ and some reasonable m that has a known strictly feasible point, and test your code, checking that the algorithm does converge quadratically. Keep in mind that the centering problem may not have a solution if *(Assumption)* is not satisfied. In the unlikely event that you randomly generate an instance (A, b, c) that violates *(Assumption)*, and therefore the centering problem doesn't have a solution (and a `cvx` solution confirms it), try another instance.

In addition to submitting your code, include the following in your writeup:

- Details of the system of equations used to calculate the Newton step, including formulae for the gradient and Hessian;
- The efficient way to solve the system, which should be implemented in your code;
- Figure with the above plot, showing the convergence of your implemented algorithm on the generated instance.

Note: you will need a working version of this code to proceed. If you are stuck, you can use a cvx-based implementation to continue with other questions. If that's what you've done, (i) acknowledge it in your writeup, and (ii) you would not be able to generate and plot algorithm's progress, or `history` *in the next question. Partial credit will be assigned accordingly.*

(b) (15 points) Implement a barrier method to solve (LP) given a strictly feasible starting point x^0 using results from part (a), i.e., your code should essentially consist of calls to function `LPcentering` with varying inputs. Your code should be in a file named `LPsolverf.m`, with the first line `function [xstar,history,value]=LPsolverf(A,b,c,x0)`, where (i) `xstar` is the (approximate) solution to (LP), (ii) matrix `history` is explained below, and (iii) `value` is the (approximate) optimal value of (LP).

You can select the initial value of the barrier parameter $t(0) > 0$ without worrying about the centering of the initial iterate, i.e., it's OK if the first centering step of your implementation takes a bit longer.

You can terminate your barrier method when the duality gap is smaller than $\tilde{\epsilon}$, where $\tilde{\epsilon}$ is in the order of 10^{-3} or 10^{-4} (If you make the tolerance much smaller, you might run into some numerical trouble. Professionally-written solvers use a variety of implementation techniques to avoid these numerical issues and solve problems to higher accuracy, but that's not our focus here.)

Try a few different values of μ to ensure the algorithm behaves as expected.

Your algorithm's output `history` should be a 2-by-k matrix (where k is the total number of centering steps), whose first row contains the number of Newton steps required for each centering step, and whose second row shows the duality gap at the end of each centering step. In order to get a plot that looks like the ones in the book (e.g., figure 11.4, page 572), you should use the following code:

```
[xx, yy] = stairs(cumsum(historymod1),historymod2);
semilogy(xx,yy);
```

where `cumsum(historymod1)` and `historymod2` are vectors calculated from history to ensure the staircase plot reflects the full trajectory of the iterates, including the first and the last centering step (experiment to get the vectors right).

Include this plot for the instance from the previous question for three values of μ: a small, a medium, and a larger one, in your writeup. Based on your observations, select the best value of μ, and submit the file with that value of μ defined in the code. (we will run your code on other instances as part of grading.)