

EE364a Homework 8 solutions

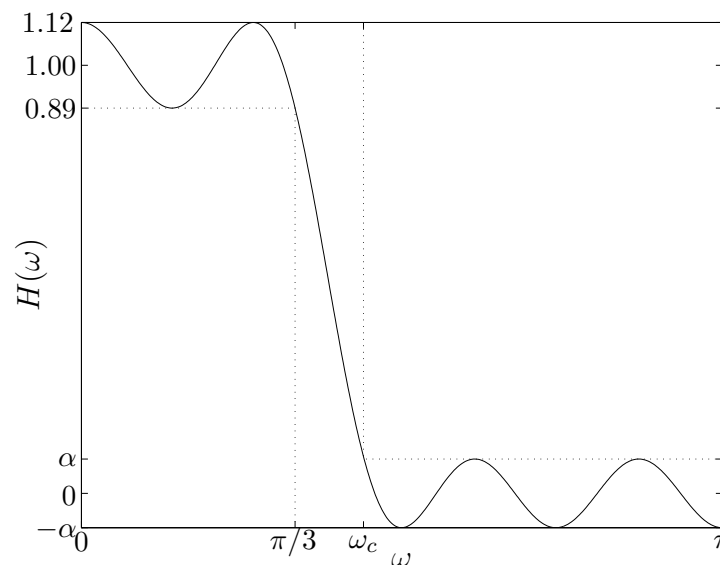
1. *FIR filter design.* Consider the (symmetric, linear phase) FIR filter described by

$$H(\omega) = a_0 + \sum_{k=1}^N a_k \cos k\omega.$$

The design variables are the real coefficients $a = (a_0, \dots, a_N) \in \mathbf{R}^{N+1}$. In this problem we will explore the design of a low-pass filter, with specifications:

- For $0 \leq \omega \leq \pi/3$, $0.89 \leq H(\omega) \leq 1.12$, *i.e.*, the filter has about ± 1 dB ripple in the ‘passband’ $[0, \pi/3]$.
- For $\omega_c \leq \omega \leq \pi$, $|H(\omega)| \leq \alpha$. In other words, the filter achieves an attenuation given by α in the ‘stopband’ $[\omega_c, \pi]$. ω_c is called the ‘cutoff frequency’.

These specifications are depicted graphically in the figure below.



- (a) Suppose we fix ω_c and N , and wish to maximize the stop-band attenuation, *i.e.*, minimize α such that the specifications above can be met. Explain how to pose this as a convex optimization problem.
- (b) Suppose we fix N and α , and want to minimize ω_c , *i.e.*, we set the stopband attenuation and filter length, and wish to minimize the ‘transition’ band (between $\pi/3$ and ω_c). Explain how to pose this problem as a quasiconvex optimization problem.

- (c) Now suppose we fix ω_c and α , and wish to find the smallest N that can meet the specifications, *i.e.*, we seek the shortest length FIR filter that can meet the specifications. Can this problem be posed as a convex or quasiconvex problem? If so, explain how. If you think it cannot be, briefly and informally explain why.
- (d) Plot the optimal tradeoff curve of attenuation (α) versus cutoff frequency (ω_c) for $N = 7$. Is the set of achievable specifications convex? Briefly explain any interesting features, *e.g.*, flat portions, of the optimal tradeoff curve.

For this subproblem, you may sample the constraints in frequency, which means the following. Choose $K \gg N$ (perhaps $K \approx 10N$), and set $\omega_k = k\pi/K$, $k = 0, \dots, K$. Then replace the specifications with

- For k with $0 \leq \omega_k \leq \pi/3$, $0.89 \leq H(\omega_k) \leq 1.12$.
- For k with $\omega_c \leq \omega_k \leq \pi$, $|H(\omega_k)| \leq \alpha$.

With this approximation, the problem in part (a) becomes an LP, which allows you to solve part (d) numerically.

Solution.

- (a) The first problem can be expressed as

$$\begin{aligned}
 & \text{minimize} && \alpha \\
 & \text{subject to} && f_1(a) \leq 1.12 \\
 & && f_2(a) \geq 0.89 \\
 & && f_3(a) \leq \alpha \\
 & && f_4(a) \geq -\alpha
 \end{aligned} \tag{1}$$

where

$$\begin{aligned}
 f_1(a) &= \sup_{0 \leq \omega \leq \pi/3} H(\omega), & f_2(a) &= \inf_{0 \leq \omega \leq \pi/3} H(\omega), \\
 f_3(a) &= \sup_{\omega_c \leq \omega \leq \pi} H(\omega), & f_4(a) &= \inf_{\omega_c \leq \omega \leq \pi} H(\omega).
 \end{aligned}$$

Problem (1) is convex in the variables a, α because f_1 and f_3 are convex functions (pointwise supremum of affine functions), and f_2 and f_4 are concave functions (pointwise infimum of affine functions).

- (b) This problem can be expressed

$$\begin{aligned}
 & \text{minimize} && f_5(a) \\
 & \text{subject to} && f_1(a) \leq 1.12 \\
 & && f_2(a) \geq 0.89
 \end{aligned}$$

where f_1 and f_2 are the same functions as above, and

$$f_5(a) = \inf\{\alpha \mid -\alpha \leq H(\omega) \leq \alpha \text{ for } \omega_c \leq \omega \leq \pi\}.$$

This is a quasiconvex optimization problem in the variables a because f_1 is convex, f_2 is concave, and f_5 is quasiconvex: its sublevel sets are

$$\{a \mid f_5(a) \leq \Omega\} = \{a \mid -\alpha \leq H(\omega) \leq \alpha \text{ for } \Omega \leq \omega \leq \pi\},$$

i.e., the intersection of an infinite number of halfspaces.

(c) This problem can be expressed as

$$\begin{aligned} & \text{minimize} && f_6(a) \\ & \text{subject to} && f_1(a) \leq 1.12 \\ & && f_2(a) \geq 0.89 \\ & && f_3(a) \leq \alpha \\ & && f_4(a) \geq -\alpha \end{aligned}$$

where f_1 , f_2 , f_3 , and f_4 are defined above and

$$f_6(a) = \min\{k \mid a_{k+1} = \dots = a_N = 0\}.$$

The sublevel sets of f_6 are affine sets:

$$\{a \mid f_6(a) \leq k\} = \{a \mid a_{k+1} = \dots = a_N = 0\}.$$

This means f_6 is a quasiconvex function, and again we have a quasiconvex optimization problem.

(d) After discretizing we can express the problem in part (a) as the LP

$$\begin{aligned} & \text{minimize} && \alpha \\ & \text{subject to} && 0.89 \leq H(\omega_i) \leq 1.12 \text{ for } 0 \leq \omega_i \leq \pi/3 \\ & && -\alpha \leq H(\omega_i) \leq \alpha \text{ for } \omega_c \leq \omega_i \leq \pi \end{aligned} \tag{2}$$

with variables α and a . (For fixed ω_i , $H(\omega_i)$ is an affine function of a , hence all constraints in this problem are linear inequalities in a and α .) We obtain the tradeoff curve of α vs. ω_c , by solving this LP for a sequence of values of ω_c in the interval $(\pi/3, \pi]$.

Figure 1 was generated by the following Matlab code.

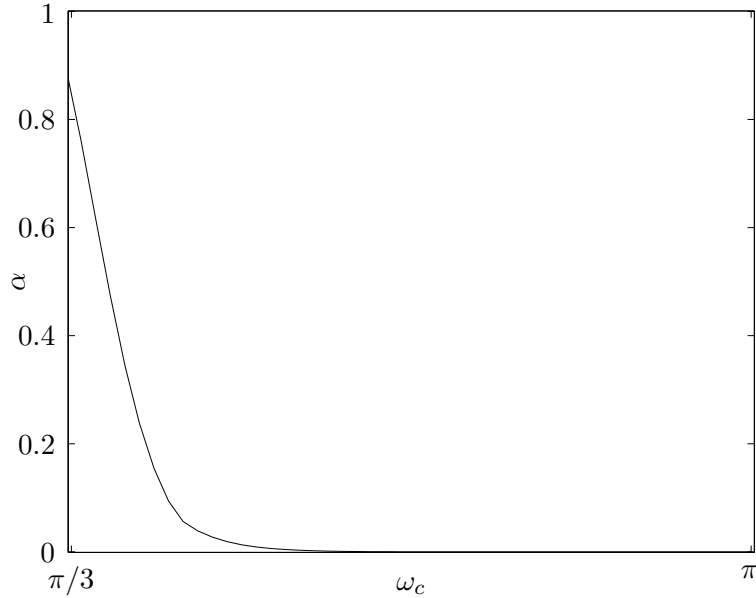


Figure 1 Tradeoff curve for problem 1d.

```

clear all
N = 7;
K = 10*N;
k = [0:N]';
w = [0:K]'/K*pi;
idx= max(find(w<=pi/3));
alphas = [];

for i=idx:length(w)

    cvx_begin
    variables a(N+1,1)
    minimize( norm(cos(w(i:end)*k')*a,inf) )
    subject to
        cos(w(1:idx)*k')*a >= 0.89
        cos(w(1:idx)*k')*a <= 1.12
    cvx_end
    alphas = [alphas; cvx_optval];
end;

plot(w(idx:end),alphas,'-');
xlabel('wc');
ylabel('alpha');

```

2. *Maximum likelihood prediction of team ability.* A set of n teams compete in a tournament. We model each team's ability by a number $a_j \in [0, 1]$, $j = 1, \dots, n$. When teams j and k play each other, the probability that team j wins is equal to $\mathbf{prob}(a_j - a_k + v > 0)$, where $v \sim \mathcal{N}(0, \sigma^2)$.

You are given the outcome of m past games. These are organized as

$$(j^{(i)}, k^{(i)}, y^{(i)}), \quad i = 1, \dots, m,$$

meaning that game i was played between teams $j^{(i)}$ and $k^{(i)}$; $y^{(i)} = 1$ means that team $j^{(i)}$ won, while $y^{(i)} = -1$ means that team $k^{(i)}$ won. (We assume there are no ties.)

- (a) Formulate the problem of finding the maximum likelihood estimate of team abilities, $\hat{a} \in \mathbf{R}^n$, given the outcomes, as a convex optimization problem. You will find the *game incidence matrix* $A \in \mathbf{R}^{m \times n}$, defined as

$$A_{il} = \begin{cases} y^{(i)} & l = j^{(i)} \\ -y^{(i)} & l = k^{(i)} \\ 0 & \text{otherwise,} \end{cases}$$

useful.

The prior constraints $\hat{a}_i \in [0, 1]$ should be included in the problem formulation. Also, we note that if a constant is added to all team abilities, there is no change in the probabilities of game outcomes. This means that \hat{a} is determined only up to a constant, like a potential. But this doesn't affect the ML estimation problem, or any subsequent predictions made using the estimated parameters.

- (b) Find \hat{a} for the team data given in `team_data.m`, in the matrix `train`. (This matrix gives the outcomes for a tournament in which each team plays each other team once.) You may find the `cvx` function `log_normcdf` helpful for this problem. You can form A using the commands

```
A = sparse(1:m,train(:,1),train(:,3),m,n) + ...
      sparse(1:m,train(:,2),-train(:,3),m,n);
```

- (c) Use the maximum likelihood estimate \hat{a} found in part (b) to predict the outcomes of next year's tournament games, given in the matrix `test`, using $\hat{y}^{(i)} = \mathbf{sign}(\hat{a}_{j^{(i)}} - \hat{a}_{k^{(i)}})$. Compare these predictions with the actual outcomes, given in the third column of `test`. Given the fraction of correctly predicted outcomes.

The games played in `train` and `test` are the same, so another, simpler method for predicting the outcomes in `test` it to just assume the team that won last year's match will also win this year's match. Give the percentage of correctly predicted outcomes using this simple method.

Solution.

(a) The likelihood of the outcomes y given a is

$$p(y|a) = \prod_{i=1, \dots, n} \Phi \left(\frac{1}{\sigma} y^{(i)} (a_{j^{(i)}} - a_{k^{(i)}}) \right),$$

where Φ is the cumulative distribution of the standard normal. The log-likelihood function is therefore

$$l(a) = \log p(y|a) = \sum_i \log \Phi \left((1/\sigma)(Aa)_i \right).$$

This is a concave function.

The maximum likelihood estimate \hat{a} is any solution of

$$\begin{aligned} & \text{maximize} && l(a) \\ & \text{subject to} && 0 \preceq a \preceq 1. \end{aligned}$$

This is a convex optimization problem since the objective, which is maximized, is concave, and the constraints are $2n$ linear inequalities.

(b) The following code solves the problem

```
% Form adjacency matrix
A1 = sparse(1:m,train(:,1),train(:,3),m,n);
A2 = sparse(1:m,train(:,2),-train(:,3),m,n);
A = A1+A2;

% Estimate abilities
cvx_begin
    variable a_hat(n)
    minimize(-sum(log_normcdf(A*a_hat/sigma)))
    subject to
        a_hat >= 0
        a_hat <= 1
cvx_end
```

Using this code we get that $\hat{a} = (1.0, 0.0, 0.68, 0.37, 0.79, 0.58, 0.38, 0.09, 0.67, 0.58)$.

(c) The following code is used to predict the outcomes in the test set

```
% Estimate errors in test set
A1 = sparse(1:m_test,test(:,1),1,m_test,n);
A2 = sparse(1:m_test,test(:,2),-1,m_test,n);
A_test = A1+A2;
res = sign(A_test*a_hat);
Pml = 1-length(find(res-test(:,3)))/m_test
Ply = 1-length(find(train(:,3)-test(:,3)))/m_test
```

The maximum likelihood estimate gives a correct prediction of 86.7% of the games in `test`. On the other hand, 75.6% of the games in `test` have the same outcome as the games in `train`.

3. *Planning production with uncertain demand.* You must order (nonnegative) amounts r_1, \dots, r_m of raw materials, which are needed to manufacture (nonnegative) quantities q_1, \dots, q_n of n different products. To manufacture one unit of product j requires at least A_{ij} units of raw material i , so we must have $r \succeq Aq$. (We will assume that A_{ij} are nonnegative.) The per-unit cost of the raw materials is given by $c \in \mathbf{R}_+^m$, so the total raw material cost is $c^T r$.

The (nonnegative) demand for product j is denoted d_j ; the number of units of product j sold is $s_j = \min\{q_j, d_j\}$. (When $q_j > d_j$, $q_j - d_j$ is the amount of product j produced, but not sold; when $d_j > q_j$, $d_j - q_j$ is the amount of unmet demand.) The revenue from selling the products is $p^T s$, where $p \in \mathbf{R}_+^n$ is the vector of product prices. The profit is $p^T s - c^T r$. (Both d and q are real vectors; their entries need not be integers.)

You are given A , c , and p . The product demand, however, is not known. Instead, a set of K possible demand vectors, $d^{(1)}, \dots, d^{(K)}$, with associated probabilities π_1, \dots, π_K , is given. (These satisfy $\mathbf{1}^T \pi = 1$, $\pi \succeq 0$.)

You will explore two different optimization problems that arise in choosing r and q (the variables).

I. Choose r and q ahead of time. You must choose r and q , knowing only the data listed above. (In other words, you must order the raw materials, and commit to producing the chosen quantities of products, before you know the product demand.) The objective is to maximize the expected profit.

II. Choose r ahead of time, and q after d is known. You must choose r , knowing only the data listed above. Some time after you have chosen r , the demand will become known to you. This means that you will find out which of the K demand vectors is the true demand. Once you know this, you must choose the quantities to be manufactured. (In other words, you must order the raw materials before the product demand is known; but you can choose the mix of products to manufacture after you have learned the true product demand.) The objective is to maximize the expected profit.

- (a) Explain how to formulate each of these problems as a convex optimization problem. Clearly state what the variables are in the problem, what the constraints are, and describe the roles of any auxiliary variables or constraints you introduce.
- (b) Carry out the methods from part (a) on the problem instance with numerical data given in `planning_data.m`. This file will define A , D , K , c , m , n , p and π . The K columns of D are the possible demand vectors. For both of the problems described above, give the optimal value of r , and the expected profit.

Solution. We first consider the case when r and q must be decided before the demand is known. The variables are $r \in \mathbf{R}^m$ and $q \in \mathbf{R}^n$. The cost of the resources is deterministic; it's just $c^T r$. The revenue from product sales is a random variable, with values

$$p^T \min\{q, d^{(k)}\}, \quad k = 1, \dots, K,$$

(where $\min\{q, d^{(k)}\}$ is meant elementwise), with associated probabilities π_1, \dots, π_K . The expected profit is therefore

$$-c^T r + \sum_{k=1}^K \pi_k p^T \min\{q, d^{(k)}\}.$$

Our problem is thus

$$\begin{aligned} & \text{maximize} && -c^T r + \sum_{k=1}^K \pi_k p^T \min\{q, d^{(k)}\} \\ & \text{subject to} && q \succeq 0, \quad r \succeq 0, \quad r \succeq Aq, \end{aligned}$$

with variables r and q . The objective is concave (and piecewise-linear), and the constraints are all linear inequalities, so this is a convex optimization problem.

Now we turn to the case when we must commit to the resource order, but can choose the product mix after we know the demand. In this case we need to have a separate product quantity vector for each possible demand vector. Thus we have variables $r \in \mathbf{R}_+^m$ and

$$q^{(1)}, \dots, q^{(K)} \in \mathbf{R}_+^n.$$

Here $q^{(k)}$ is the product mix we produce if $d^{(k)}$ turns out to be the actual product demand. Our problem can be formulated as

$$\begin{aligned} & \text{maximize} && -c^T r + \sum_{k=1}^K \pi_k p^T \min\{q^{(k)}, d^{(k)}\} \\ & \text{subject to} && r \succeq 0 \\ & && q^{(k)} \succeq 0, \quad r \succeq Aq^{(k)}, \quad k = 1, \dots, K. \end{aligned}$$

Note that the variables r and $q^{(1)}, \dots, q^{(K)}$ must be decided at the same time, taking all of them into account; we cannot optimize them separately. Nor can we decide on r first, without taking the different $q^{(k)}$'s into account. (However, provided we have solved the problem above to get the right r , we can then optimize over q , once we know which demand vector is the true one. But there is no need for this, since we have already worked out the optimal q for each possible demand vector.)

We can simplify this a bit. In this case, we will not produce any excess product, because this wastes resources, and we know the demand before we decide how much to manufacture. So we will have $q \preceq d$, and we can write the problem as

$$\begin{aligned} & \text{maximize} && -c^T r + \sum_{k=1}^K \pi_k p^T q^{(k)} \\ & \text{subject to} && r \succeq 0 \\ & && d^{(k)} \succeq q^{(k)} \succeq 0, \quad r \succeq Aq^{(k)}, \quad k = 1, \dots, K. \end{aligned}$$

With the values from `planning_data.m`, the optimal values of r for the two problems, respectively, are

$$\begin{aligned} r_I &= (45.5, 65.8, 44.6, 76.7, 72.0, 77.8, 59.3, 61.9, 83.2, 73.1) \\ r_{II} &= (65.7, 59.6, 55.2, 70.8, 69.7, 68.4, 55.3, 57.4, 63.8, 66.2) \end{aligned}$$

These yield expected profits, respectively, of 114.3 and 133.0.

The `cvx` code that solves this problem is listed below.

```
disp('Problem I.')
```

```
cvx_begin
    variable r(m); % amount of each raw material to buy.
    variable q(n); % amount of each product to manufacture.
    r >= 0;
    q >= 0;
    r >= A*q;
    S = min(q*ones(1,K),D);
    maximize(p'*S*pi - c'*r)
cvx_end
rI = r; disp(r);
disp(['profit: ' num2str(cvx_optval)]);
profitI = p'*S - c'*r; % save for graphing.
```

```
disp('Problem II.')
```

```
cvx_begin
    variable r(m); % amount of each raw material to buy.
    variable q(n, K); % product to manufacture, for each demand.
    q >= 0;
    r >= 0;
    r*ones(1,K) >= A*q;
    S = min(q, D);
    maximize(p'*S*pi - c'*r)
cvx_end
rII = r; disp(rII);
disp(['profit: ' num2str(cvx_optval)]);
profitII = p'*S - c'*r; % save for graphing.
```

We mention two other optimization problems, that we didn't ask you to explore. First, suppose we committed to buy r_I resources under the first plan, but later learned the demand (before production started). The question then is: how suboptimal is the expected profit when using the r_I from the first problem instead of r_{II} ? For the given numerical instance, our expected profit drops from 133.0 to 128.7.

Finally, it's also interesting to look at performance when we know the demand before buying the resources. This is the full prescience, or full information case—we know everything beforehand. In this case we can optimize the choice of r and q for each possible demand vector separately. This obviously gives an upper bound on the profit available when we have less information. In this case, the expected profit with full prescience is 161.4.

The code that solves these two additional problems is listed below.

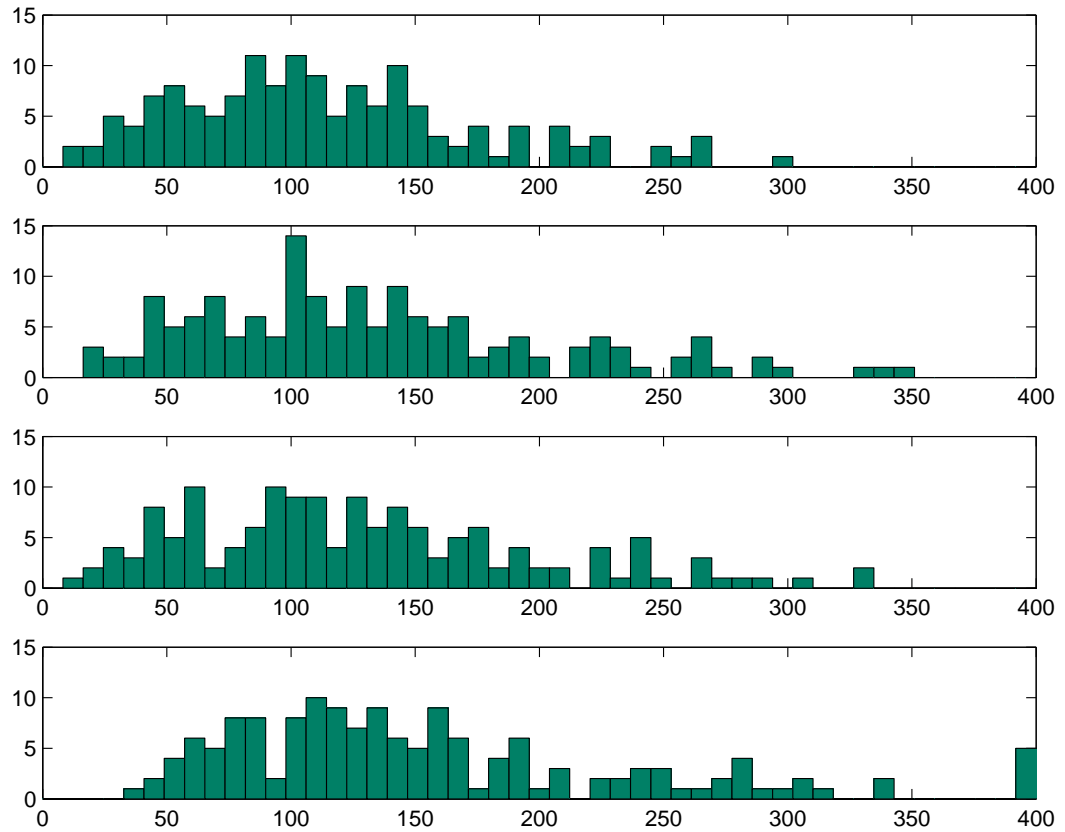
```

disp('Problem II but with rI.')
r = rI;
cvx_begin
    variable q(n, K); % product to manufacture, for each demand.
    q >= 0;
    r*ones(1,K) >= A*q;
    S = min(q, D);
    maximize(p'*S*pi - c'*r)
cvx_end
disp(['profit: ' num2str(p'*S*pi - c'*r)]);
profitIII = p'*S - c'*r; % save for graphing.

disp('Full prescience.')
cvx_begin
    variable r(m, K); % amount of each raw material to buy, for each demand..
    variable q(n, K); % product to manufacture, for each demand.
    q >= 0;
    r >= 0;
    r >= A*q;
    S = min(q, D);
    maximize((p'*S - c'*r)*pi)
cvx_end
disp(['profit: ' num2str((p'*S*pi - c'*r)*pi)]);
profitIV = p'*S - c'*r; % save for graphing.

```

Histograms showing profits with the different possible outcomes are shown below.



Solutions to additional exercises

Standard form LP barrier method

In the following three exercises, you will implement a barrier method for solving the standard form LP

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax = b, \quad x \succeq 0, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, where $A \in \mathbf{R}^{m \times n}$, with $m < n$. Throughout this exercise we will assume that A is full rank, and the sublevel sets $\{x \mid Ax = b, x \succeq 0, c^T x \leq \gamma\}$ are all bounded. (If this is not the case, the centering problem is unbounded below.)

1. *Centering step.* Implement Newton's method for solving the centering problem

$$\begin{aligned} & \text{minimize} && c^T x - \sum_{i=1}^n \log x_i \\ & \text{subject to} && Ax = b, \end{aligned}$$

with variable x , given a strictly feasible starting point x_0 .

Your code should accept A , b , c , and x_0 , and return x^* , the primal optimal point, ν^* , a dual optimal point, and the number of Newton steps executed.

Use the block elimination method to compute the Newton step. (You can also compute the Newton step via the KKT system, and compare the result to the Newton step computed via block elimination. The two steps should be close, but if any x_i is very small, you might get a warning about the condition number of the KKT matrix.)

Plot $\lambda^2/2$ versus iteration k , for various problem data and initial points, to verify that your implementation gives asymptotic quadratic convergence. As stopping criterion, you can use $\lambda^2/2 \leq 10^{-6}$. Experiment with varying the algorithm parameters α and β , observing the effect on the total number of Newton steps required, for a fixed problem instance. Check that your computed x^* and ν^* (nearly) satisfy the KKT conditions.

To generate some random problem data (*i.e.*, A , b , c , x_0), we recommend the following approach. First, generate A randomly. (You might want to check that it has full rank.) Then generate a random positive vector x_0 , and take $b = Ax_0$. (This ensures that x_0 is strictly feasible.) The parameter c can be chosen randomly. To be sure the sublevel sets are bounded, you can add a row to A with all positive elements. If you want to be able to repeat a run with the same problem data, be sure to set the state for the uniform and normal random number generators.

Here are some hints that may be useful.

- We recommend computing λ^2 using the formula $\lambda^2 = -\Delta x_{\text{nt}}^T \nabla f(x)$. You don't really need λ for anything; you can work with λ^2 instead. (This is important for reasons described below.)

- There can be small numerical errors in the Newton step Δx_{nt} that you compute. When x is nearly optimal, the computed value of λ^2 , *i.e.*, $\lambda^2 = -\Delta x_{\text{nt}}^T \nabla f(x)$, can actually be (slightly) negative. If you take the squareroot to get λ , you'll get a complex number, and you'll never recover. Moreover, your line search will never exit. However, this only happens when x is nearly optimal. So if you exit on the condition $\lambda^2/2 \leq 10^{-6}$, everything will be fine, even when the computed value of λ^2 is negative.
- For the line search, you must first multiply the step size t by β until $x + t\Delta x_{\text{nt}}$ is feasible (*i.e.*, strictly positive). If you don't, when you evaluate f you'll be taking the logarithm of negative numbers, and you'll never recover.

2. *LP solver with strictly feasible starting point.* Using the centering code from part (1), implement a barrier method to solve the standard form LP

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax = b, \quad x \succeq 0, \end{aligned}$$

with variable $x \in \mathbf{R}^n$, given a strictly feasible starting point x_0 . Your LP solver should take as argument A , b , c , and x_0 , and return x^* .

You can terminate your barrier method when the duality gap, as measured by n/t , is smaller than 10^{-3} . (If you make the tolerance much smaller, you might run into some numerical trouble.) Check your LP solver against the solution found by `cvx`, for several problem instances.

The comments in part (1) on how to generate random data hold here too.

Experiment with the parameter μ to see the effect on the number of Newton steps per centering step, and the total number of Newton steps required to solve the problem.

Plot the progress of the algorithm, for a problem instance with $n = 500$ and $m = 100$, showing duality gap (on a log scale) on the vertical axis, versus the cumulative total number of Newton steps (on a linear scale) on the horizontal axis.

Your algorithm should return a $2 \times k$ matrix `history`, (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(history(1,:)),history(2,:));
semilogy(xx,yy);
```

3. *LP solver.* Using the code from part (2), implement a general standard form LP solver, that takes arguments A , b , c , determines (strict) feasibility, and returns an optimal point if the problem is (strictly) feasible.

You will need to implement a phase I method, that determines whether the problem is strictly feasible, and if so, finds a strictly feasible point, which can then be fed to the code from part (2). In fact, you can use the code from part (2) to implement the phase I method.

To find a strictly feasible initial point x_0 , we solve the phase I problem

$$\begin{aligned} & \text{minimize} && t \\ & \text{subject to} && Ax = b \\ & && x \succeq (1-t)\mathbf{1}, \quad t \geq 0, \end{aligned}$$

with variables x and t . If we can find a feasible (x, t) , with $t < 1$, then x is strictly feasible for the original problem. The converse is also true, so the original LP is strictly feasible if and only if $t^* < 1$, where t^* is the optimal value of the phase I problem.

We can initialize x and t for the phase I problem with any x^0 satisfying $Ax^0 = b$, and $t^0 = 2 - \min_i x_i^0$. (Here we can assume that $\min_i x_i^0 \leq 0$; otherwise x^0 is already a strictly feasible point, and we are done.) You can use a change of variable $z = x + (t-1)\mathbf{1}$ to transform the phase I problem into the form in part (2).

Check your LP solver against `cvx` on several numerical examples, including both feasible and infeasible instances.

Solution.

1. The Newton step Δx_{nt} is defined by the KKT system:

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{\text{nt}} \\ w \end{bmatrix} = \begin{bmatrix} -g \\ 0 \end{bmatrix},$$

where $H = \mathbf{diag}(1/x_1^2, \dots, 1/x_n^2)$, and $g = c - (1/x_1, \dots, 1/x_n)$. The KKT system can be efficiently solved by block elimination, *i.e.*, by solving

$$AH^{-1}A^T w = -AH^{-1}g,$$

and setting $\Delta x_{\text{nt}} = -H^{-1}(A^T w + g)$. The KKT optimality condition is

$$A^T \nu^* + c - (1/x_1^*, \dots, 1/x_n^*) = 0.$$

When the Newton method converges, *i.e.*, $\Delta x_{\text{nt}} \approx 0$, w is the dual optimal point ν^* .

The following function computes the analytic center using Newton's method.

```
function [x_star, nu_star, lambda_hist] = lp_acent(A,b,c,x_0)
% solves problem
% minimize c'*x - sum(log(x))
% subject to A*x = b
% using Newton's method, given strictly feasible starting point x0
```

```

% input (A, b, c, x_0)
% returns primal and dual optimal points
% lambda_hist is a vector showing  $\lambda^2/2$  for each newton step
% returns [], [] if MAXITERS reached, or x_0 not feasible

% algorithm parameters
ALPHA = 0.01;
BETA = 0.5;
EPSILON = 1e-6;
MAXITERS = 100;

if (min(x_0) <= 0) || (norm(A*x_0 - b) > 1e-3) % x0 not feasible
    fprintf('FAILED');
    nu_star = []; x_star = []; lambda_hist=[];
    return;
end

m = length(b);
n = length(x_0);

x = x_0; lambda_hist = [];
for iter = 1:MAXITERS
    H = diag(x.^(-2));
    g = c - x.^(-1);
    % lines below compute newton step via whole KKT system
    % M = [ H A'; A zeros(m,m)];
    % d = M\[-g; zeros(m,1)];
    % dx = d(1:n);
    % w = d(n+1:end);

    % newton step by elimination method
    w = (A*diag(x.^2)*A')\(-A*diag(x.^2)*g);
    dx = -diag(x.^2)*(A'*w + g);

    lambdasqr = -g'*dx; % dx'*H*dx;
    lambda_hist = [lambda_hist lambdasqr/2];
    if lambdasqr/2 <= EPSILON break; end

    % backtracking line search
    % first bring the point inside the domain
    t = 1; while min(x+t*dx) <= 0 t = BETA*t; end
    % now do backtracking line search

```

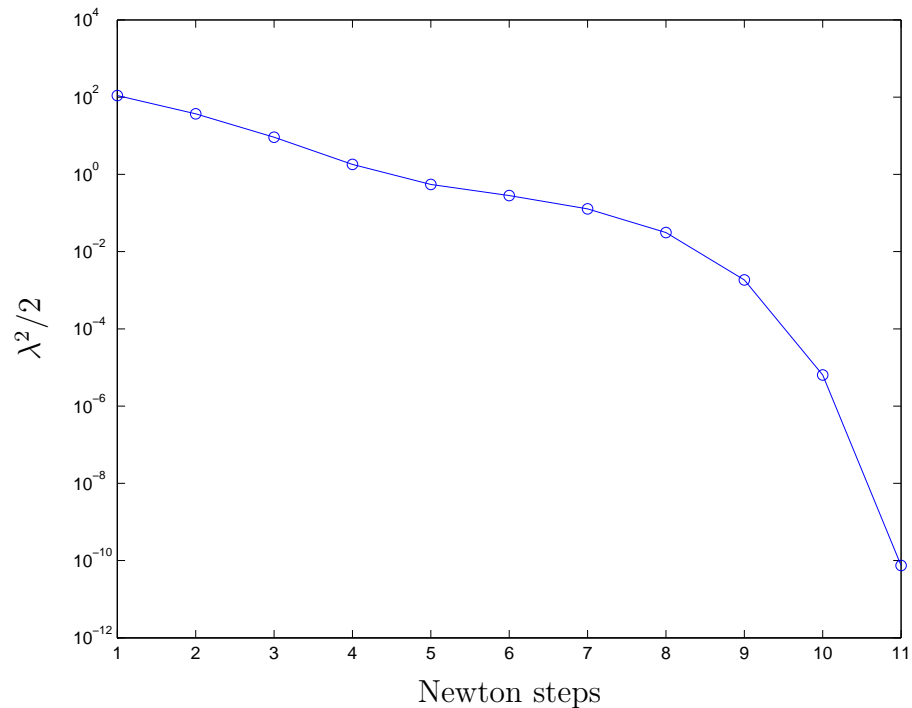
```

while c'*(t*dx)-sum(log(x+t*dx))+sum(log(x))-ALPHA*t*g'*dx > 0
    t = BETA*t;
end
x = x + t*dx;
end

if iter == MAXITERS % MAXITERS reached
    fprintf('ERROR: MAXITERS reached.\n');
    x_star = []; nu_star = [];
else
    x_star = x;
    nu_star = w;
end
end

```

The random data is generated as given in the problem statement, with $A \in \mathbf{R}^{100 \times 500}$. The Newton decrement versus number of Newton steps is plotted below. Quadratic convergence is clear. The Newton direction computed by the two methods are very close. The KKT optimality conditions are verified for the points returned by the function.



2. The following function solves the LP using the barrier method.

```
function [x_star, history, gap] = lp_barrier(A,b,c,x_0)
```

```

% solves standard form LP
% minimize      c^T x
% subject to    Ax = b, x >=0;
% using barrier method, given strictly feasible x0
% uses function std_form_LP_acent() to carry out centering steps
% returns:
% - primal optimal point x_star
% - history, a 2xk matrix that returns number of newton steps
%   in each centering step (top row) and duality gap (bottom row)
%   (k is total number of centering steps)
% - gap, optimal duality gap

% barrier method parameters
T_0 = 1;
MU = 20;
EPSILON = 1e-3; % duality gap stopping criterion

n = length(x_0);
t = T_0;
x = x_0;
history = [];

while(1)
    [x_star, nu_star, lambda_hist] = lp_acent(A,b,t*c,x);
    x = x_star;
    gap = n/t;
    history = [history [length(lambda_hist); gap]];
    if gap < EPSILON break; end
    t = MU*t;
end

```

The following script generates test data and plots the progress of the barrier method. The script also checks the computed solution against cvx.

```

% script that generates data and tests the functions
% std_form_LP_acent
% std_form_LP_barrier

clear all;
m = 100;
n = 500;

```

```

rand('seed',0);
randn('seed',0);
A = [randn(m-1,n); ones(1,n)];
x_0 = rand(n,1) + 0.1;
b = A*x_0;
c = randn(n,1);

% analytic centering
figure
[x_star, nu_star, lambda_hist] = lp_acent(A,b,c,x_0);
semilogy(lambda_hist,'bo-')
xlabel('iters')
ylabel('lambdasqr/2')

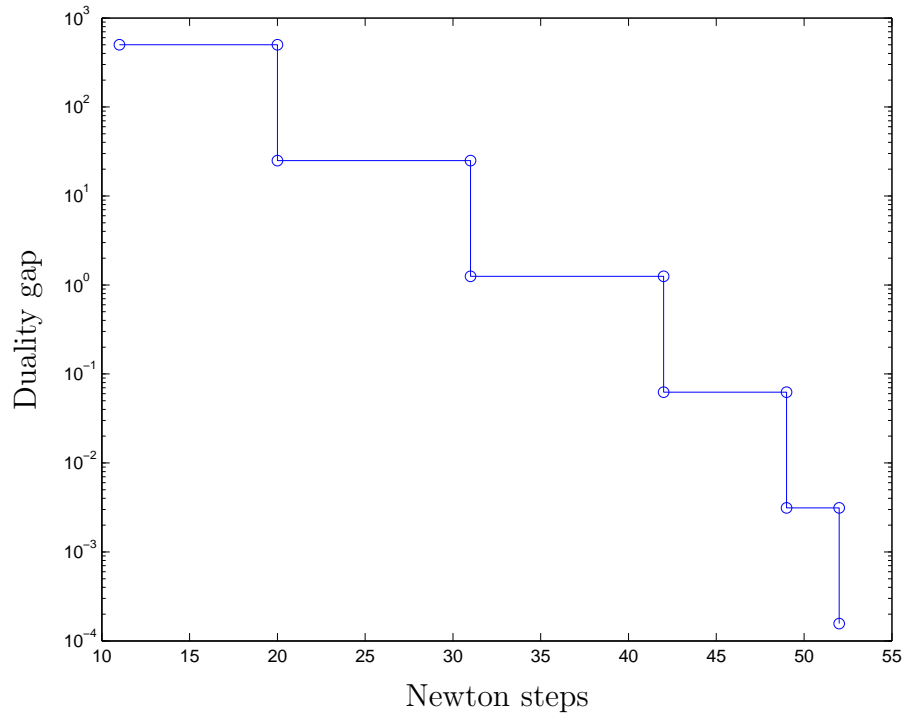
% solve the LP with barrier
figure
[x_star, history, gap] = lp_barrier(A,b,c,x_0);
[xx, yy] = stairs(cumsum(history(1,:)),history(2,:));
semilogy(xx,yy,'bo-');
xlabel('iters')
ylabel('gap')

p_star = c'*x_star;

% solve LP using cvx for comparison
cvx_begin
    variable x(n)
    minimize(c'*x)
    subject to
        A*x == b
        x >= 0
cvx_end

fprintf('\n\nOptimal value found by barrier method:\n');
p_star
fprintf('Optimal value found by CVX:\n');
cvx_optval
fprintf('Duality gap from barrier method:\n');
gap

```



3. The following function implements the full LP solver (phase I and phase II).

```
function [x_star,p_star,gap,status,nsteps] = lp_solve(A,b,c);
% solves the LP
% minimize      c^T x
% subject to    Ax = b, x >= 0;
% using a barrier method
% computes a strictly feasible point by carrying out
% a phase I method
% returns:
% - a primal optimal point x_star
% - the primal optimal value p_star
% - status: either 'Infeasible' or 'Solved'
% - nsteps(1): number of newton steps for phase I
% - nsteps(2): number of newton steps for phase I

[m,n] = size(A);
nsteps = zeros(2,1);

% phase I
x0 = A\b; t0 = 2+max(0,-min(x0));
A1 = [A,-A*ones(n,1)];
b1 = b-A*ones(n,1);
```

```

z0 = x0+t0*ones(n,1)-ones(n,1);
c1 = [zeros(n,1);1];
[z_star, history, gap] = lp_barrier(A1,b1,c1,[z0;t0]);
if (z_star(n+1) >= 1)
    fprintf('\nProblem is infeasible\n');
    x_star = []; p_star = Inf; status = 'Infeasible';
    nsteps(1) = sum(history(1,:)); gap = [];
    return;
end
fprintf('\nFeasible point found\n');
nsteps(1) = sum(history(1,:));
x_0 = z_star(1:n)-z_star(n+1)*ones(n,1)+ones(n,1);

% phase II
[x_star, history, gap] = lp_barrier(A,b,c,x_0);
status = 'Solved'; p_star = c'*x_star;
nsteps(2) = sum(history(1,:));

```

We test our LP solver on two problem instances, one infeasible, and one feasible. We check our results against the output of cvx.

```

% solves standard form LP for two problem instances

clear all;
m = 100;
n = 500;

% infeasible problem instance
rand('state',0);
randn('state',0);
A = [randn(m-1,n); ones(1,n)];
b = randn(m,1);
c = randn(n,1);

[x_star,p_star,gap,status,nsteps] = lp_solve(A,b,c);

% solve LP using cvx for comparison
cvx_begin
    variable x(n)
    minimize(c'*x)
    subject to
        A*x == b

```

```

        x >= 0
cvx_end

% feasible problem instance
A = [randn(m-1,n); ones(1,n)];
v = rand(n,1) + 0.1;
b = A*v;
c = randn(n,1);

[x_star,p_star,gap,status,nsteps] = lp_solve(A,b,c);

% solve LP using cvx for comparison
cvx_begin
    variable x(n)
    minimize(c'*x)
    subject to
        A*x == b
        x >= 0
cvx_end

fprintf('\n\nOptimal value found by barrier method:\n');
p_star
fprintf('Optimal value found by CVX:\n');
cvx_optval
fprintf('Duality gap from barrier method:\n');
gap

```