EE364b Homework 5

1. *Distributed method for bi-commodity network flow problem.* We consider a network (directed graph) with n arcs and p nodes, described by the incidence matrix $A \in \mathbf{R}^{p\times n}$, where

$$A_{ij} = \begin{cases} 1, & \text{if arc } j \text{ enters node } i \\ -1, & \text{if arc } j \text{ leaves node } i \\ 0, & \text{otherwise.} \end{cases}$$

   Two commodities flow in the network. Commodity 1 has source vector $s \in \mathbf{R}^p$, and commodity 2 has source vector $t \in \mathbf{R}^p$, which satisfy $\mathbf{1}^T s = \mathbf{1}^T t = 0$. The flow of commodity 1 on arc i is denoted x_i, and the flow of commodity 2 on arc i is denoted y_i. Each of the flows must satisfy flow conservation, which can be expressed as $Ax + s = 0$ (for commodity 1), and $Ay + t = 0$ (for commodity 2).

   Arc i has associated flow cost $\phi_i(x_i, y_i)$, where $\phi_i : \mathbf{R}^2 \to \mathbf{R}$ is convex. (We can impose constraints such as nonnegativity of the flows by restricting the domain of ϕ_i to $\mathbf{R}^2_+$.) One natural form for ϕ_i is a function only the total traffic on the arc, *i.e.*, $\phi(x_i, y_i) = f_i(x_i + y_i)$, where $f_i : \mathbf{R} \to \mathbf{R}$ is convex. In this form, however, ϕ is not strictly convex, which will complicate things. To avoid these complications, we will assume that ϕ_i is strictly convex.

   The problem of choosing the minimum cost flows that satisfy flow conservation can be expressed as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^n \phi_i(x_i, y_i) \\ \text{subject to} & Ax + s = 0, \quad Ay + t = 0, \end{array}$$

   with variables $x,\ y \in \mathbf{R}^n$. This is the *bi-commodity network flow problem.*

   (a) Propose a distributed solution to the bi-commodity flow problem using dual decomposition. Your solution can refer to the conjugate functions ϕ_i^*.

   (b) Use your algorithm to solve the particular problem instance with

$$\phi_i(x_i, y_i) = (x_i + y_i)^2 + \epsilon(x_i^2 + y_i^2), \qquad \mathbf{dom}\,\phi_i = \mathbf{R}^2_+,$$

   with $\epsilon = 0.1$. The other data for this problem can be found in `bicommodity_data.m`. To check that your method works, compute the optimal value $p^\star$, using `cvx`.

   For the subgradient updates use a constant stepsize of 0.1. Run the algorithm for 200 iterations and plot the dual lower bound versus iteration. With a logarithmic vertical axis, plot the norms of the residuals for each of the two flow conservation equations, versus iteration number, on the same plot.

Hint: We have posted a function `[x,y] = quad2_min(eps,alpha,beta)`, which computes

$$(x^*, y^*) = \operatorname*{argmin}_{x \geq 0, y \geq 0} \left((x+y)^2 + \epsilon(x^2+y^2) + \alpha x + \beta y \right)$$

analytically. You might find this function useful.

Solution:

(a) The Lagrangian of the flow control problem is

$$\begin{aligned} L(x,y,\mu,\nu) &= \sum_{i=1}^n \phi_i(x_i,y_i) + \mu^T(Ax+s) + \nu^T(Ay+t) \\ &= \mu^T s + \nu^T t + \sum_{i=1}^n \left(\phi_i(x_i,y_i) + (\Delta\mu_i)x_i + (\Delta\nu_i)y_i \right), \end{aligned}$$

where $\Delta\mu_i = a_i^T\mu$ and $\Delta\nu_i = a_i^T\nu$. The dual function is

$$\begin{aligned} g(\mu,\nu) &= \inf_{x \succeq 0, y \succeq 0} L(x,y,\mu,\nu) \\ &= \mu^T s + \nu^T t - \sum_{i=1}^n \phi_i^*(-\Delta\mu_i, -\Delta\nu_i), \end{aligned}$$

where $\phi_i^*(u,v) = \sup_{x \geq 0, y \geq 0} (xu + yv - \phi_i(x,y))$. The dual is the unconstrained problem

$$\text{maximize} \quad g(\mu,\nu).$$

In complete analogy to the analysis of the lecture notes, by using a subgradient method to solve the dual problem we obtain the following algorithm.

given initial potential vectors μ and ν.
repeat
Determine link flows from potential differences.
$(x_i,y_i) := \operatorname{argmin} \left(\phi_i(x_i,y_i) + (\Delta\mu_i)x_i + (\Delta\nu_i)y_i \right), \quad j = 1,\dots,n.$
Compute flow surplus at each node.
$S_i := a_i^T x + s_i, \quad i = 1,\dots,p.$
$T_i := a_i^T y + t_i, \quad i = 1,\dots,p.$
Update node potentials.
$\mu_i := \mu_i + \alpha_k S_i, \quad i = 1,\dots,p.$
$\nu_i := \nu_i + \alpha_k T_i, \quad i = 1,\dots,p.$

(b) The following code solves the problem

```
bicommodity_data;

% Get solution
cvx_begin
```

```
    variables x_star(n) y_star(n)
    dual variables mu_star nu_star
    minimize(sum((x_star+y_star).^2)+eps*(sum(x_star.^2+y_star.^2)))
    subject to
        mu_star: A*x_star+s==0;
        nu_star: A*y_star+t==0;
        x_star >= 0;
        y_star >= 0;
cvx_end
f_min = cvx_optval;

% Dual decomposition
mu = zeros(p,1); nu = zeros(p,1);
x = zeros(n,1); y = zeros(n,1);
MAX_ITER = 200;
L = []; infeas1 = []; infeas2 = [];
for i = 1:MAX_ITER
    % Potential differences
    delta_mu = A'*mu; delta_nu = A'*nu;

    % Update flows
    for j = 1:n
        [x(j),y(j)] = quad2_min(eps,delta_mu(j),delta_nu(j));
    end
    infeas1 = [infeas1 norm(A*x+s)];
    infeas2 = [infeas2 norm(A*y+t)];

    % Update lower bound
    l = sum((x+y).^2)+eps*(sum(x.^2+y.^2))+mu'*(A*x+s)+nu'*(A*y+t);
    L = [L l];

    % Update potentials
    alpha = .1;
    mu = mu+alpha*(A*x+s);
    nu = nu+alpha*(A*y+t);
end

figure
plot(1:MAX_ITER,L,'b-',[1 MAX_ITER],[f_min f_min],'r--')
legend('lb','opt')
xlabel('iter')
```

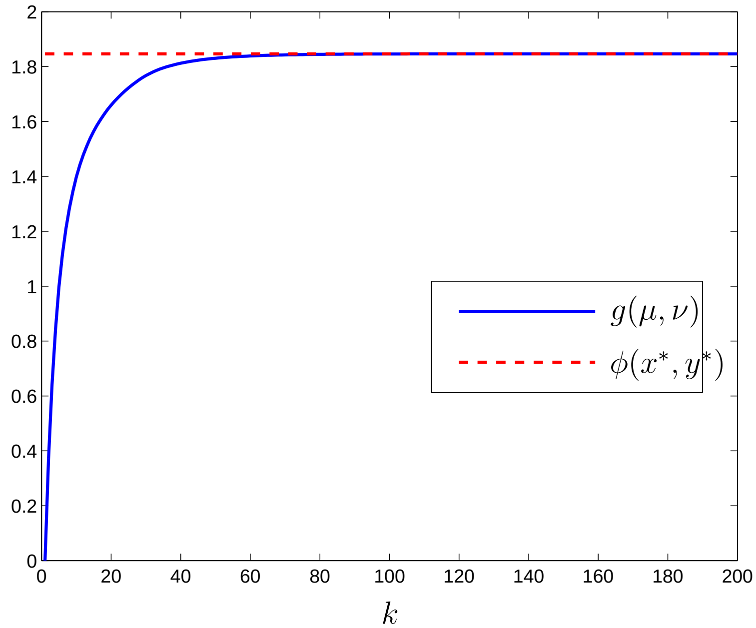


Figure 1: Lower bound $g(\mu, \nu)$ on flow cost versus iteration k.

```
figure
semilogy(infeas1)
xlabel('iter')
ylabel('ninfeas1')

figure
semilogy(infeas2)
ylabel('ninfeas2')
xlabel('iter')
```

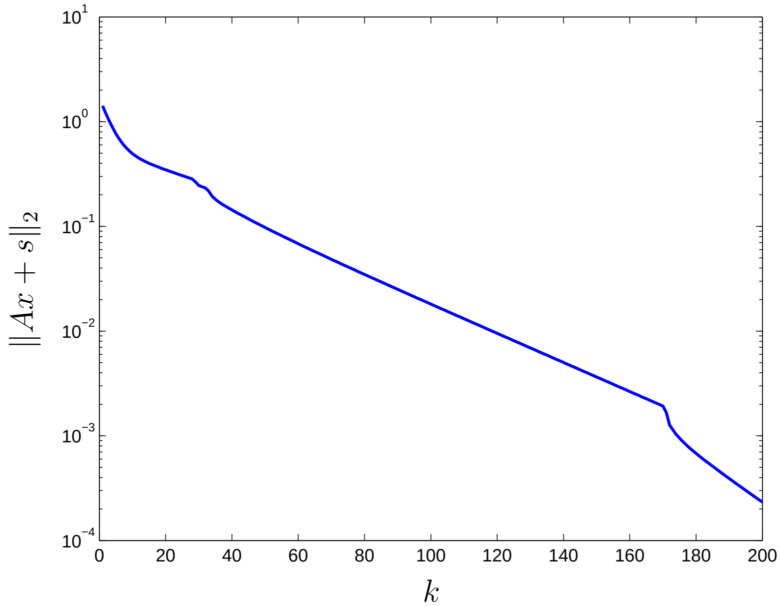


Figure 2: Evolution of infeasibility for commodity 1 versus iteration k.

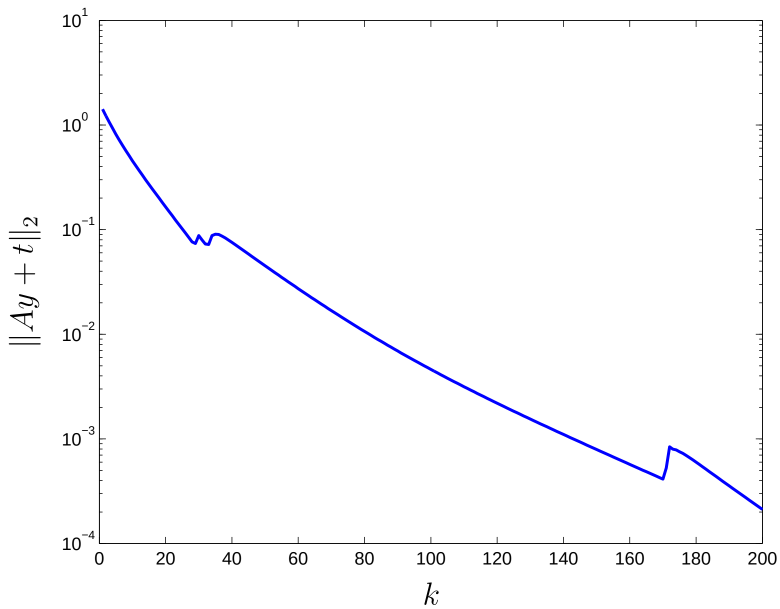


Figure 3: Evolution of infeasibility for commodity 2 versus iteration k.

2. *Minimum eigenvalue via convex-concave procedure.* The (nonconvex) problem

$$\begin{array}{ll} \text{minimize} & x^T P x \\ \text{subject to} & \|x\|_2^2 \geq 1, \end{array}$$

with $P \in \mathbf{S}_+^{n \times n}$, has optimal value $\lambda_{\min}(P)$; x is optimal if and only if it is an eigenvector of P associated with $\lambda_{\min}(P)$. Explain how to use the convex-concave procedure to (try to) solve this problem.

Generate and (possibly) solve a few instances of this problem using the convex-concave procedure, starting from a few (nonzero) initial points. Compare the values found by the convex-concave procedure with the optimal value.

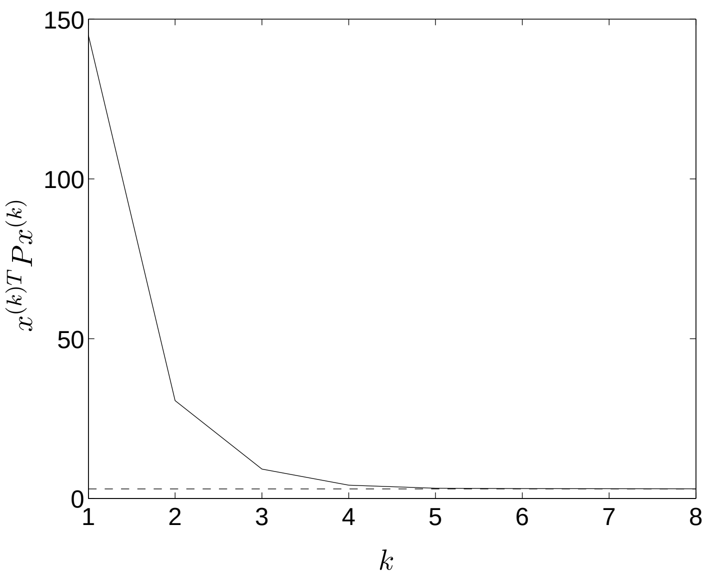
Solution. We solve the problem

$$\begin{array}{ll} \text{minimize} & x^T P x \\ \text{subject to} & 1 - \|x\|_2^2 \leq 0 \end{array}$$

using the convex-concave procedure. Linearizing $1 - \|x\|_2^2$ around the point $x^{(k)}$, we set $x^{(k+1)}$ to the optimal point x in the problem

$$\begin{array}{ll} \text{minimize} & x^T P x \\ \text{subject to} & 1 - (\|x^{(k)}\|^2 + 2x^{(k)T}(x - x^{(k)})) \leq 0. \end{array}$$

An example graph, and its generating Matlab code, appear below.




```
randn('state', 1091); rand('state', 1091); n = 40;

P = randn(n); P = 3*eye(n) + P'*P;
```

```
vals = [];
xold = 1/n*ones(n, 1);
Nmax = 8;
for i = 1:Nmax
    cvx_begin
        cvx_quiet(true);
        variable x(n)
        minimize(x'*P*x)
        1 - 2*xold'*(x - xold) - xold'*xold <= 0;
    cvx_end
    disp(cvx_optval)
    vals = [vals; cvx_optval];
    xold = x;
end

figure(1); cla;
plot(vals); hold on;
plot([1 Nmax], min(eig(P))*[1 1], 'k--');
xlabel('x'); ylabel('y');
print -deps2 mineig.eps
```

3. *Ellipsoid method for an SDP.* We consider the SDP

$$\begin{array}{ll} \text{maximize} & \mathbf{1}^T x \\ \text{subject to} & x_i \succeq 0, \quad \Sigma - \mathbf{diag}(x) \succeq 0, \end{array}$$

with variable $x \in \mathbf{R}^n$ and data $\Sigma \in \mathbf{S}^n_{++}$. The first inequality is a vector (componentwise) inequality, and the second inequality is a matrix inequality. (This specific SDP arises in several applications.)

Explain how to use the ellipsoid method to solve this problem. Describe your choice of initial ellipsoid and how you determine a subgradient for the objective (expressed as $-\mathbf{1}^T x$, which is to be minimized) or constraint functions (expressed as $\max_i(-x_i) \leq 0$ and $\lambda_{\max}(\mathbf{diag}(x) - \Sigma) \leq 0$). You can describe a basic ellipsoid method; you do not need to use a deep-cut method, or work in the epigraph.

Try out your ellipsoid method on some randomly generated data, with $n \leq 20$. Use a stopping criterion that guarantees 1% accuracy. Compare the result to the solution found using `cvx`. Plot the upper and lower bounds from the ellipsoid method, versus iteration number.

Solution. There are many possible choices for an initial ellipsoid that contains the optimal point $x^\star$. For example, we can use the fact that all feasible x lie in the box $0 \preceq x \preceq \mathbf{diag}(\Sigma)$. One simple starting ellipsoid is the Euclidean ball centered at $x = (1/2)\,\mathbf{diag}(\Sigma)$ (the center of the box), with radius $(1/2)(\sum_{i=1}^n \Sigma_{ii}^2)^{1/2}$. Suppose that at the kth iteration we have $D^{(k)} = \mathbf{diag}(x^{(k)})$.

We'll transform the problem to one of minimizing $f(x) = -\mathbf{1}^T x$. When $x^{(k)}$ is feasible, the subgradient of f is simply $-\mathbf{1}$. If a component of $x^{(k)}$ (say, the jth component) is negative, we use the constraint subgradient $g = -e_j$.

The interesting case comes when $x^{(k)} \succeq 0$, but $\lambda_{\max}(\mathbf{diag}(x^{(k)}) - \Sigma) > 0$. In this case we can find a constraint subgradient as follows. Find an eigenvector v associated with the largest eigenvalue of $\Sigma - \mathbf{diag}(x^{(k)})$. By the weak subgradient calculus, we can find any subgradient of the function

$$v^T(\mathbf{diag}(x^{(k)} - \Sigma)v = -v^T \Sigma v + \sum_{i=1}^n v_i^2 x_i^{(k)}.$$

But this is trivial, since this function is affine: We can take $g_i = v_i^2$, for $i = 1, \ldots, n$.

The following Matlab code implements the ellipsoid method.

```
randn('state',0); n = 20;
Sigma = randn(n,n);
Sigma = Sigma'*Sigma+eye(n);

x = 0.5*diag(Sigma);
A = diag(ones(n,1)*0.25*sum(diag(Sigma).^2));
maxiters = 5000; U = inf; L = -inf; hist = []; feas = 0;
tol = 0.01;

for k = 1:maxiters
    % find a subgradient
    [val,ind] = min(x);
    [V,d] = eig(diag(x)-Sigma);
    if (val <= 0)
        feas = 0;
        g = zeros(n,1); g(ind) = -1;
    elseif (max(diag(d)) >= 0)
        feas = 0;
        g = V(:,n).^2;
    else
        g = -ones(n,1);
        feas = 1; D = diag(x);
        U = min(U,-sum(x));
        L = max(L,-sum(x)-sqrt(g'*A*g));
    end
    hist = [hist [k;feas;U;L]];
    if (((U-L) < tol*sum(x)) && (feas == 1)) break; end;
    % update the ellipsoid
    g = g/sqrt(g'*A*g);
```

```
    x = x-A*g/(n+1);
    A = (n^2/(n^2-1))*(A-A*g*g'*A*(2/(n+1)));
end

cvx_begin
    variable xopt(n)
    Sigma-diag(xopt) == semidefinite(n);
    xopt >= 0;
    minimize(-sum(xopt))
cvx_end

figure;
set(gca,'Fontsize',16);
plot(hist(3,:),'k-'); hold on;
plot(hist(4,:),'k--');
plot(1:length(hist(3,:)),cvx_optval*ones(length(hist(3,:)),1),'k:');
xlabel('k'); ylabel('ul');
axis([0,2000,-120,-20]);
print('-depsc', 'ellipsoid_sdp.eps');
```

The following figure shows the progress of the ellipsoid method with iteration number k. The solid line shows the upper bound u_k, the dashed line shows the lower bound l_k, and the dotted line is the optimal value $p^\star$, obtained by using `cvx`.