CME 307 / MS&E 311 Winter 2022
Optimization Jan 28, 2022
Prof. Yinyu Ye

Homework 1 Sample Solutions

1. (15') Show the followings:

   (a) (5') Consider the set

   $$F := \{\mathbf{x} \in R^n : \ A\mathbf{x} = \mathbf{b}, \ \mathbf{x} \geq 0\},$$

   where data matrix $A \in R^{m \times n}$ and vector $\mathbf{b} \in R^m$. Prove that F is a convex set.

   (b) (5') Fix data matrix A and consider the $\mathbf{b}$-data set for F defined in part (a):

   $$B := \{\mathbf{b} \in R^m : \ F \text{ is not empty}\}.$$

   Prove that B is a convex set.

   (c) (5') Fix data matrix A and consider the linearly constrained convex minimization problem

   $$\begin{aligned} z(\mathbf{b}) := \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \ \mathbf{x} \geq \mathbf{0} \end{aligned}$$

   where $f(\mathbf{x})$ is a concave function, and the maximal value function $z(\mathbf{b})$ is an implicit function of $\mathbf{b}$. Prove that $z(\mathbf{b})$ is a concave function of $\mathbf{b} \in B$, where B is defined in part (b).

Sample Solution:

(a) Take any two points $x', x'' \in F$, that is, $Ax' = b, x' \geq 0$ and $Ax'' = b, x'' \geq 0$. Then, for any $\alpha \in [0\ 1]$ (α is so defined through out this homework) we must have

$$\alpha x' + (1-\alpha)x'' \geq 0.$$

Moreover,

$$A(\alpha x' + (1-\alpha)x'') = \alpha Ax' + (1-\alpha)Ax'' = \alpha b + (1-\alpha)b = b.$$

Thus, $\alpha x' + (1-\alpha)x'' \in F$.

(b) Take any two points $b', b'' \in B$. Then we must have $x' \geq 0$ and $x'' \geq 0$ such that $Ax' = b'$ and $Ax'' = b''$. Now we like to prove that the convex combination $\alpha b' + (1-\alpha)b''$ is also in B. Consider the convex combination $x = \alpha x' + (1-\alpha)x''$. Obviously, $x \geq 0$. Furthermore,

$$Ax = A(\alpha x' + (1-\alpha)x'') = \alpha Ax' + (1-\alpha)Ax'' = \alpha b' + (1-\alpha)b'',$$

which give the desired proof.

(c) Take any two points $b', b'' \in B$, and let x' and x'' be two minimizers for $b = b'$ and $b = b''$, respectively. That is, $z(b') = f(x')$ and $z(b'') = f(x'')$. Then, consider $z(\alpha b' + (1-\alpha)b'')$. Since $\alpha x' + (1-\alpha)x'' \geq 0$ and $A(\alpha x' + (1-\alpha)x'') = \alpha b' + (1-\alpha)b''$, $\alpha x' + (1-\alpha)x''$ is a feasible solution

for problem with $b = \alpha b' + (1-\alpha)b''$. Thus, the maximum value must be greater or equal to a feasible solution function value, that is,

$$z(\alpha b' + (1-\alpha)b'') \geq f(\alpha x' + (1-\alpha)x'') \geq \alpha f(x') + (1-\alpha)f(x'') = \alpha z(b') + (1-\alpha)z(b''),$$

where the second inequality is from f is a concave function.

2. (10') Show that the dual cone of the n-dimensional nonnegative orthant cone R^n_+ is itself, that is,

$$(R^n_+)^* = R^n_+.$$

(Hint: show that $R^n_+ \subset (R^n_+)^*$ and $(R^n_+)^* \subset R^n_+$.)

Sample Solution: Prove $R^n_+ \subset (R^n_+)^*$: Let any $y \in R^n_+$. Then $x^T y = \sum_i x_i y_i \geq 0$ for any $x \in R^n_+$ since each of the product in the sum is nonnegative.

Prove $(R^n_+)^* \subset R^n_+$: Suppose this is not true, that is, there is a $y \in (R^n_+)^*$ but $y \notin R^n_+$. Then at least one entry of y is negative, w.l.o.g., say $y_1 < 0$. Now select $e_1 = (1; 0; ...; 0) \in R^n_+$ we have

$$e_1^T y = y_1 < 0$$

which contradicts that $y \in (R^n_+)^*$.

3. (10') Let $g_1, \ldots, g_m$ be a collection of concave functions on R^n such that

$$S = \{\mathbf{x} : g_i(\mathbf{x}) > 0 \text{ for } i = 1, \ldots, m\} \neq \emptyset.$$

Show that for any positive constant μ and any convex function f on R^n, the function (called Barrier function)

$$h(\mathbf{x}) = f(\mathbf{x}) - \mu \sum_{i=1}^{m} \log(g_i(\mathbf{x}))$$

is convex over S. (Hint: directly apply the convex/concave function definition or analyze the Hessian of $h(x)$.)

Sample Solution: It is easy to verify that S is convex. We know that the positively weighted sum of convex functions having a common domain is convex on that domain. The given conditions imply that the function

$$h(x) = f(x) - \mu \sum_{i=1}^{m} \log(g_i(x))$$

is a positively weighted sum of the convex functions. To see this, one can prove that a nondecreasing concave function of a concave function is concave, that is, $\log g_i(x)$ is a concave function. To prove it, take any two points x' and x'' in S, then for each i

$$g_i(\alpha x' + (1-\alpha)x'') \geq \alpha g_i(x') + (1-\alpha)g_i(x'').$$

Since log is nondecreasing,

$$\log(g_i(\alpha x' + (1-\alpha)x'')) \geq \log(\alpha g_i(x') + (1-\alpha)g_i(x'')).$$

Moreover, log is a concave function, so that

$$\log(g_i(\alpha x' + (1-\alpha)x'')) \geq \log(\alpha g_i(x') + (1-\alpha)g_i(x'')) \geq \alpha \log(g_i(x')) + (1-\alpha)\log(g_i(x'')),$$

which complete the proof. Hence its negative $-\log g_i(x)$ is convex. Thus, we see that

$$h(x) = f(x) + \sum_{i=1}^{m} [\mu(-\log g_i(x))]$$

is convex on S.

4. (10') (Lipschitz Functions) Prove the following two implication inequalities:

   (a) (5') Assume f is a first-order β-Lipschitz function, namely there is a positive number β such that for any $\mathbf{x}, \mathbf{y} \in R^n$:

   $$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq \beta \|\mathbf{x} - \mathbf{y}\|,$$

   then for any $\mathbf{x}, \mathbf{y} \in R^n$,

   $$|f(\mathbf{x}) - f(\mathbf{y}) - \nabla f(\mathbf{y})^T(\mathbf{x} - \mathbf{y})| \leq \frac{\beta}{2}\|\mathbf{x} - \mathbf{y}\|^2.$$

   (b) (5') Assume f is a second-order β-Lipschitz function, namely there is a positive number β such that for any $\mathbf{x}, \mathbf{y} \in R^n$:

   $$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}) - \nabla^2 f(\mathbf{y})(\mathbf{x} - \mathbf{y})\| \leq \beta \|\mathbf{x} - \mathbf{y}\|^2,$$

   then for any $\mathbf{x}, \mathbf{y} \in R^n$,

   $$|f(\mathbf{x}) - f(\mathbf{y}) - \nabla f(\mathbf{y})^T(\mathbf{x} - \mathbf{y}) - \frac{1}{2}(\mathbf{x} - \mathbf{y})^T \nabla^2 f(\mathbf{y})(\mathbf{x} - \mathbf{y})| \leq \frac{\beta}{3}\|\mathbf{x} - \mathbf{y}\|^3.$$

Solution: The key tool is Taylor's formula with integral remainder (*cf.*, `https://en.wikipedia.org/wiki/Taylor%27s_theorem#Derivation_for_the_integral_form_of_the_remainder`).

Let $\Delta := y - x$ and $\phi(t) = f(x + t\Delta)$ where t is a scalar variable. Then we have $\phi(0) = f(x)$ and $\phi(1) = f(x + \Delta) = f(y)$. Moreover,

$$f(x + \Delta) - f(x) = \phi(1) - \phi(0) = \int_0^1 \mathrm{d}\phi(t) = \int_0^1 \Delta^T \nabla f(x + t\Delta)\mathrm{d}t.$$

For the first implication inequality, noting $\Delta^T \nabla f(x) = \int_0^1 \Delta^T \nabla f(x)\mathrm{d}t$, we have

$$\begin{aligned}
&|f(x + \Delta) - f(x) - \nabla f(x)^T \Delta| = \left| \int_0^1 \Delta^T \left(\nabla f(x + t\Delta) - \nabla f(x)\right) \mathrm{d}t \right| \\
&\leq \int_0^1 \left| \Delta^T \left(\nabla f(x + t\Delta) - \nabla f(x)\right) \right| \mathrm{d}t \\
&\leq \int_0^1 \|\Delta\| \, \|\nabla f(x + t\Delta) - \nabla f(x)\| \, \mathrm{d}t \quad \text{(Cauchy-Schwartz inequality)} \\
&= \|\Delta\| \int_0^1 \|\nabla f(x + t\Delta) - \nabla f(x)\| \, \mathrm{d}t \\
&\leq \|\Delta\| \int_0^1 \beta \|t\Delta\| \mathrm{d}t \quad \text{(the first-order Lipschitz condition)} \\
&= \|\Delta\| \beta \|\Delta\| \int_0^1 t \mathrm{d}t = \frac{\beta}{2}\|\Delta\|^2.
\end{aligned}$$

For the second implication inequality, further noting $\frac{1}{2}\Delta^T\nabla^2 f(x)\Delta = \Delta^T\nabla^2 f(x)\Delta\int_0^1 t\mathrm{d}t$, we have

$$
\begin{aligned}
&|f(x+\Delta) - f(x) - \nabla f(x)^T\Delta - \frac{1}{2}\Delta^T\nabla^2 f(x)\Delta| \\
&= \left|\int_0^1 \Delta^T\left(\nabla f(x+t\Delta) - \nabla f(x) - \nabla^2 f(x)(t\Delta)\right)\mathrm{d}t\right| \\
&\le \int_0^1 \left|\Delta^T\left(\nabla f(x+t\Delta) - \nabla f(x) - \nabla^2 f(x)(t\Delta)\right)\right|\mathrm{d}t \\
&\le \int_0^1 \|\Delta\|\|\nabla f(x+t\Delta) - \nabla f(x) - \nabla^2 f(x)(t\Delta)\|\mathrm{d}t \quad \text{(Cauchy-Schwartz inequality)} \\
&= \|\Delta\|\int_0^1 \|\nabla f(x+t\Delta) - \nabla f(x) - \nabla^2 f(x)(t\Delta)\|\mathrm{d}t \\
&\le \|\Delta\|\int_0^1 \beta\|t\Delta\|^2\mathrm{d}t \quad \text{(the second-order Lipschitz condition)} \\
&= \beta\|\Delta\|^3\int_0^1 t^2\mathrm{d}t = \frac{\beta}{3}\|\Delta\|^3.
\end{aligned}
$$

This completes our proof. □

5. (10') Consider the following SOCP problem:

$$
\begin{aligned}
\min \quad & 2x_1 + x_2 + x_3 \\
\text{s.t.} \quad & x_1 + x_2 + x_3 = 1, \\
& x_1 - \sqrt{x_2^2 + x_3^2} \ge 0.
\end{aligned}
$$

   (a) (5') Show that the feasible region is a convex set.

   (b) (5') Try to find a minimizer of the problem and "argue" [1] why it is a minimizer.

   Sample Solution:

   - It is clear that the plane set $\{x:\ e^T x = 1\}$ is a convex set. Let $x_{-1} = (x_2; x_3; ...; x_n)$. Then we like to prove that

     $$\{x:\ \|x_{-1}\| \le x_1\}$$

     is a convex set. Consider any two points x' and x'' in the set. For any $\alpha \in [0\ 1]$, we have, by triangle inequality,

     $$\|\alpha x'_{-1} + (1-\alpha)x''_{-1}\| \le \|\alpha x'_{-1}\| + \|(1-\alpha)x''_{-1}\| = \alpha\|x'_{-1}\| + (1-\alpha)\|x''_{-1}\|.$$

     But $\|x'_{-1}\| \le x'_1$ and $\|x''_{-1}\| \le x''_1$, so that

     $$\|\alpha x'_{-1} + (1-\alpha)x''_{-1}\| \le \alpha x'_1 + (1-\alpha)x''_1;$$

     that is, the convex combination point is also in the set. This implies that the set is a convex set. The feasible region is the intersection of two convex sets, so that it is also a convex set.

[1] We recommend to prove this directly, namely without using duality argument which will be introduced in the following lectures.

- The problem can be treated as

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 = 1, \\ & x_1 - \sqrt{x_2^2 + x_3^2} \geq 0; \end{aligned}$$

  which is as the same as

$$\begin{aligned} \max \quad & x_2 + x_3 \\ \text{s.t.} \quad & x_2 + x_3 + \sqrt{x_2^2 + x_3^2} \leq 1. \end{aligned}$$

  For any fixed positive value of $x_2 + x_3$, $\sqrt{x_2^2 + x_3^2}$ would be minimized when $x_2 = x_3$. Thus, we consider the case $x_2 = x_3$: which is as the same as

$$\begin{aligned} \max \quad & 2x_2 \\ \text{s.t.} \quad & 2x_2 + \sqrt{2}x_2 \leq 1. \end{aligned}$$

  That is, $x_2 = \frac{1}{2+\sqrt{2}}$. Thus, the minimal value of the original problem might be $2 - 2x_2 = \sqrt{2}$.

6. (10') Prove that the set $\{A\mathbf{x} : \ \mathbf{x} \geq 0 \in R^n\}$ is a closed and convex cone. (Hint: apply Carathéodory's theorem in Lecture Note to prove the closedness.)

   Sample Solution: Let $C = \{Ax : \ x \geq 0 \in R^n\}$.

   It is easy to see that C is a cone. Take any $b \in C$, then $b = Ax$ for some $x \geq 0$. Now consider βb for any $\beta \geq 0$. But $\beta b = \beta(Ax) = A(\beta x)$ and $\beta x \geq 0$, so that $\beta b \in C$.

   The convexity is easy to prove. Take $b^1 \in C$ and $b^2 \in C$. Then we must have $x^1 \geq 0$ and $x^2 \geq 0$ such that $b^1 = Ax^1$ and $b^2 = Ax^2$. Then for any $\alpha \in [0\ 1]$,

$$\alpha b^1 + (1-\alpha)b^2 = \alpha(Ax^1) + (1-\alpha)(Ax^2) = A(\alpha x^1 + (1-\alpha)x^2).$$

   since $\alpha x^1 + (1-\alpha)x^2 \geq 0$, we have $\alpha b^1 + (1-\alpha)b^2 \in C$.

   Now let $b^k = Ax^k, \ x^k \geq 0, \ k = 1, ...,$ be a bounded and convergent sequence and let its limit points be $\bar{b}$. We would like to prove $\bar{b} \in C$. From Carathéodory's theorem, we can assume that x^k is a basic feasible solution, that is, for some basis $B^k \subseteq [n]$, we have $A_{B^k} x^k_{B^k} = b^k$, and the rest of entries in x^k are all zeros. Then, x^k is bounded for all $k = 1, ...$ Thus, there must be a subsequence of $x^k \geq 0$, $k \in \mathcal{K} := \{k_1, k_2, ...\}$, that is convergent. Let its limit to be $\bar{x}$. We have $\bar{x} \geq 0$. Now consider the subsequence $b^k, \ k \in \mathcal{K}$, that is, $b^k = Ax^k, \ x^k \geq 0, \ k \in \mathcal{K}$. Thus, it is a convergent sequence with limit $A\bar{x} \in C$. But the limit of any convergent subsequence of b^k is $\bar{b}$, so that $\bar{b} = A\bar{x} \in C$.

7. (15') Farkas' lemma can be used to derive many other (named) theorems of the alternative. This problem concerns a few of these pairs of systems. Using Farkas's lemma, prove each of the following results.

   (a) (5') Gordan's Theorem. Exactly one of the following systems has a solution:

$$\begin{aligned} &\text{(i)} \quad A\mathbf{x} > 0 \\ &\text{(ii)} \quad \mathbf{y}^T A = 0, \quad \mathbf{y} \geq 0, \quad \mathbf{y} \neq 0. \end{aligned}$$

   (b) (5') Stiemke's Theorem. Exactly one of the following systems has a solution:

$$\text{(i)} \quad A\mathbf{x} \geq 0, \quad A\mathbf{x} \neq 0$$
$$\text{(ii)} \quad \mathbf{y}^T A = 0, \quad \mathbf{y} > 0$$

(c) (5') Gale's Theorem. Exactly one of the following systems has a solution:

$$\text{(i)} \quad A\mathbf{x} \leq \mathbf{b}$$
$$\text{(ii)} \quad \mathbf{y}^T A = 0, \quad \mathbf{y}^T \mathbf{b} < 0, \quad \mathbf{y} \geq 0$$

Solution: (a) Gordan's Theorem. Let b denote a positive vector. Then, (i) is equivalent to $Ax \geq b$ and it can be written as

$$Ax' - Ax'' - z = b, \ (x'; x''; z) \geq 0$$

By Farkas' lemma, it is alternative system is

$$y^T(A, \ -A, \ -I) \leq 0, \ y^T b = 1$$

which is equivalent to (ii).

(b) Stiemke's Theorem. Let b denote a positive vector. Then, (i) is equivalent to $Ax \geq 0, \ b^T Ax = 1$ and it can be written as:

$$\begin{pmatrix} A & -A & -I \\ b^T A & -b^T A & 0 \end{pmatrix} (x'; x''; z) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \ (x'; x''; z) \geq 0$$

By Farkas' lemma, its alternative system is on $(y'; \tau)$ such that:

$$(y'; \tau)^T \begin{pmatrix} A & -A & -I \\ b^T A & -b^T A & 0 \end{pmatrix} \leq 0, \ (y'; \tau)^T (0; 1) = 1$$

Let $y = y' + \tau \cdot b$. Then, it is a solution to (ii).

(c) Gale's Theorem. Note that (i) can be written as:

$$Ax' - Ax'' + z = b, \ (x'; x''; z) \geq 0$$

By Farkas' lemma, its alternative system is

$$y^T(A, \ -A, \ I) \leq 0, \ y^T b = 1$$

Then, $-y$ is a solution to (ii).

8. (20') Consider the sensor localization problem on plane R^2 with one sensor $\mathbf{x}$ and three anchors $\mathbf{a}_1 = (1;0)$, $\mathbf{a}_2 = (-1;0)$ and $\mathbf{a}_3 = (0;2)$. Suppose the Euclidean distances from the sensor to the three anchors are d_1, d_2 and d_3 respectively and known to us.Then, from the anchor and distance information, we can locate the second by finding $\mathbf{x} \in R^2$ such that

$$\|\mathbf{x} - \mathbf{a}_i\|^2 = d_i^2, \ i = 1, 2, 3.$$

Do the following numerical experiments using CVX (or cvxpy, convex.jl) or MOSEK and answer the questions:

(a) (10') Generate any sensor point **in the convex hull** of the three anchors, compute its distances to three anchors d_i, $i = 1, 2, 3$, respectively. Then solve the SOCP relaxation problem

$$\|\mathbf{x} - \mathbf{a}_i\|^2 \leq d_i^2, \ i = 1, 2, 3.$$

Did you find the correct location? What about if the sensor point was in the **outside** of the convex hull? Try a few different locations of the sensor and identify the pattern.

(b) (10') Now try the SDP relaxation

$$(\mathbf{a}_i; -1)(\mathbf{a}_i; -1)^T \bullet \begin{pmatrix} I & \mathbf{x} \\ \mathbf{x}^T & \mathbf{y} \end{pmatrix} = d_i^2, \ i = 1, 2, 3; \ \begin{pmatrix} I & \mathbf{x} \\ \mathbf{x}^T & \mathbf{y} \end{pmatrix} \succeq 0 \in S^3,$$

which can be written in the standard form

$$\begin{aligned} (1;0;0)(1;0;0)^T \bullet Z &= 1, \\ (0;1;0)(0;1;0)^T \bullet Z &= 1, \\ (1;1;0)(1;1;0)^T \bullet Z &= 2, \\ (\mathbf{a}_i; -1)(\mathbf{a}_i; -1)^T \bullet Z &= d_i^2, \ i = 1, 2, 3, \\ Z &\succeq 0 \in S^3. \end{aligned}$$

Did you find the correct location everywhere on the plane? Try a few different locations of the sensor and identify the pattern.

You can use CVX (or cvxpy, convex.jl) to solve these numerical problems.

Sample Solution: Both the SOCP and SDP relaxations exactly find the sensor location if the sensor is contained in the convex hull of the anchor points.

However, when the sensor is located outside of the convex hull, the SOCP relaxation will fail to find the sensor correctly. This is due to the relaxed $\|x - a_i\| \leq d_i$ constraint, which allows regions of the convex hull to be feasible even if x^* is outside of the convex hull. Thus the SOCP relaxation will tend to return solutions within the convex hull. On the other hand, the SDP relaxation is always exact since it strictly requires that $\|x - a_i\| = d_i$.

Experimental code is given below:

```
%% MS&E 311/CME 307 Homework 1 Problem 9

%% Each column is anchor point
```

```
A = [1 -1 0;
     0  0 2];

%% Generate sensor in convex hull of 3 anchors
%% SOCP relaxation
alpha = rand(3,1);
alpha = alpha/norm(alpha,1);
s_true = A*alpha;
d = norms(A - s_true*ones(1,3));

cvx_begin quiet
    variable s(2)
    minimize( 0 )
    subject to
        norms(A - s*ones(1,3)) <= d;
cvx_end

fprintf('SOCP - Inside of Convex Hull\n');
fprintf('True sensor location     : (%f, %f)\n', s_true(1), s_true(2));
fprintf('Recovered sensor location: (%f, %f)\n', s(1), s(2));
fprintf('Difference : %f\n\n', norm(s_true - s));

%% SDP relaxation
cvx_begin sdp quiet
    variable X(3,3) semidefinite
    minimize( 0 )
    subject to
        X(1:2,1:2) == eye(2)
        for i=1:3
            [A(:,i);-1]'*X*[A(:,i);-1] == d(i)^2
        end
cvx_end

s = X(1:2,3);

fprintf('SDP - Inside of Convex Hull\n');
fprintf('True sensor location     : (%f, %f)\n', s_true(1), s_true(2));
fprintf('Recovered sensor location: (%f, %f)\n', s(1), s(2));
fprintf('Difference : %f\n\n', norm(s_true - s));

%% Generate sensor outside of convex hull of 3 anchors
alpha = 10*rand(3,1);
s_true = A*alpha;
d = norms(A - s_true*ones(1,3));

cvx_begin quiet
    variable s(2)
    minimize( 0 )
    subject to
        norms(A - s*ones(1,3)) <= d;
cvx_end

```

```
fprintf('SOCP - Outside of Convex Hull\n');
fprintf('True sensor location     : (%f, %f)\n', s_true(1), s_true(2));
fprintf('Recovered sensor location: (%f, %f)\n', s(1), s(2));
fprintf('Difference : %f\n\n', norm(s_true - s));

%% SDP relaxation
cvx_begin sdp quiet
    variable X(3,3) semidefinite
    minimize( 0 )
    subject to
        X(1:2,1:2) == eye(2)
        for i=1:3
            [A(:,i);-1]'*X*[A(:,i);-1] == d(i)^2
        end
cvx_end

s = X(1:2,3);

fprintf('SDP - Outside of Convex Hull\n');
fprintf('True sensor location     : (%f, %f)\n', s_true(1), s_true(2));
fprintf('Recovered sensor location: (%f, %f)\n', s(1), s(2));
fprintf('Difference : %f\n\n', norm(s_true - s));
```

9. (20') Consider the sensor localization problem on plane R^2 with **two** sensors $\mathbf{x}_1$ and $\mathbf{x}_2$ and three anchors $\mathbf{a}_1 = (1;0)$, $\mathbf{a}_2 = (-1;0)$ and $\mathbf{a}_3 = (0;2)$. Suppose that we know the (Euclidean) distances from one sensor $\mathbf{x}_1$ to $\mathbf{a}_1$ and $\mathbf{a}_2$, denoted by d_{11} and d_{12}; distances of the other sensor $\mathbf{x}_2$ to $\mathbf{a}_2$ and $\mathbf{a}_3$, denoted by d_{22} and d_{23}; and the distance between the two sensors $\mathbf{x}_1$ and $\mathbf{x}_2$, denoted by $\hat{d}_{12}$. Then, from the anchor and distance information we would like to locate the sensor positions $\mathbf{x}_1, \mathbf{x}_2 \in R^2$.

   Do the following numerical experiments using CVX (or cvxpy, convex.jl) or MOSEK and answer the questions:

   (a) (10') Generate two sensor points anywhere and try the SOCP relaxation model

$$\begin{aligned} \|\mathbf{x}_1 - \mathbf{a}_i\|^2 &\le d_{1i}^2, \ i = 1, 2 \\ \|\mathbf{x}_2 - \mathbf{a}_i\|^2 &\le d_{2i}^2, \ i = 2, 3 \\ \|\mathbf{x}_1 - \mathbf{x}_2\|^2 &\le \hat{d}_{12}^2. \end{aligned}$$

   Did you find the correct locations? What have you observed? Try a few different locations of the sensor pairs and identify the pattern.

   (b) (10') Now try the SDP relaxation: find $X = [\mathbf{x}_1, \ \mathbf{x}_2] \in R^{2\times 2}$ and

$$Z = \begin{pmatrix} I & X \\ X^T & Y \end{pmatrix} \in S^4$$

   to meet the constraints in the standard form:

$$\begin{aligned} (1;0;0;0)(1;0;0;0)^T \bullet Z &= 1, \\ (0;1;0;0)(0;1;0;0)^T \bullet Z &= 1, \\ (1;1;0;0)(1;1;0;0)^T \bullet Z &= 2, \\ (\mathbf{a}_i;-1;0)(\mathbf{a}_i;-1;0)^T \bullet Z &= d_{1i}^2, \ i = 1, 2, \\ (\mathbf{a}_i;0;-1)(\mathbf{a}_i;0;-1)^T \bullet Z &= d_{2i}^2, \ i = 2, 3, \\ (0;0;1;-1)(0;0;1;-1)^T \bullet Z &= \hat{d}_{12}^2, \\ Z &\succeq 0 \in S^4. \end{aligned}$$

   Did you find the correct locations? What have you observed? Can you conclude with something? Try a few different locations of the sensor pairs and identify the pattern.

Solution: (The MATLAB titles of the figures may be a bit misleading. For all the figures, red points are x_1 and blue points are x_2. Titles violating this are all wrong due to previous typos in the title generation codes. The descriptions below are still all consistent. These are minor issues and forget about these if you didn't look into such details.)

(a) For the SOCP formulation, in general we're not able to find the correct locations, even if both the sensors x_1 and x_2 are in the convex hull of the anchors a_1, a_2, a_3. To enable exact recovery, we need to require that x_1 *is inside the convex hull of* x_2, a_1, a_2 *and* x_2 *is inside the convex hull of* x_1, a_2, a_3. We will validate this claim by numerical results.

Specifically, *if at least one sensor is outside the convex hull of the anchors* a_1, a_2, a_3*, we generally cannot ensure exact recovery.* So the exact recovery condition here is *stronger* than just requiring both sensors being inside the convex hull of the anchors.

This comes from the same analysis as in problem 2, where we write down the first-order KKT conditions and keep in mind the non-positiveness of the multipliers. In particular, when the sensors are in the interior of the convex hulls, the corresponding multipliers are positive and hence by complementarity the inequalities become tight, which leads to exact recovery. On the other hand, when the sensors are on the boundaries of the convex hulls, then the inequality constraints already uniquely determine the points, and hence again we obtain exact recovery.

In contrast, if any of the convex hull inclusions of the two sensors is violated, then the corresponding (three) multipliers must all be 0. This will potentially lead to non-tight inequalities, disabling exact recovery.

In the experiments, we provide three options of generating sensors x_1, x_2 for facility of usage. The first is to generate both by `randn.m`. The second is to start by generating x_1 inside the convex hull of a_1, a_2, a_3, and then generate x_2 inside the convex hull of x_1, a_2, a_3. The third is to generate x_1 and x_2 independently inside the convex hull of a_1, a_2, a_3. Notice that the second option does not ensure that x_1 is inside x_2, a_1, a_2.

We showcase the five possible situations in the following figures:

(1) $x_1 \in \text{convhull}(x_2, a_1, a_2), x_2 \in \text{convhull}(x_1, a_2, a_3)$, $(x_1, x_2 \in \text{convhull}(a_1, a_2, a_3))$;

(2) $x_1 \notin \text{convhull}(x_2, a_1, a_2), x_2 \in \text{convhull}(x_1, a_2, a_3)$, $x_1, x_2 \in \text{convhull}(a_1, a_2, a_3)$;

(3) $x_1 \notin \text{convhull}(x_2, a_1, a_2), x_2 \notin \text{convhull}(x_1, a_2, a_3)$, $x_1, x_2 \in \text{convhull}(a_1, a_2, a_3)$;

(4) one of x_1, x_2 outside $\text{convhull}(a_1, a_2, a_3)$, and the other inside;

(5) both x_1, x_2 outside $\text{convhull}(a_1, a_2, a_3)$.

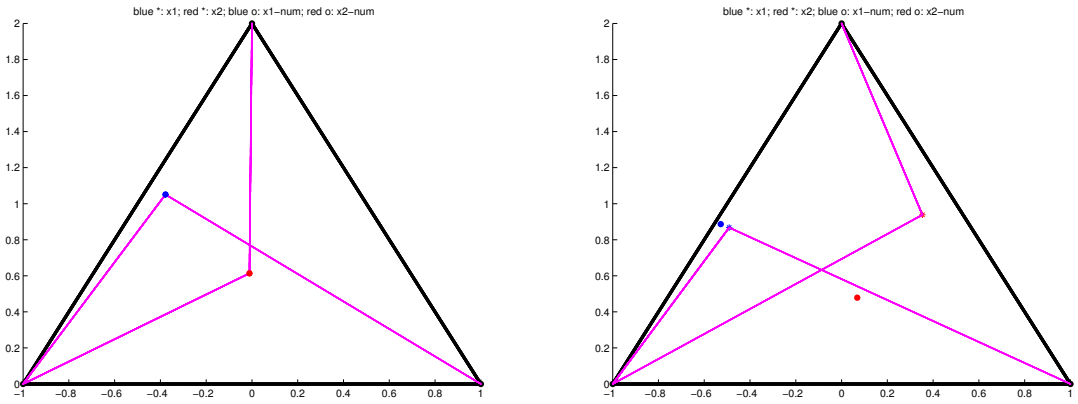

Figure 1: Left: exact recovery, case 1. Right: inexact recovery, case 2.

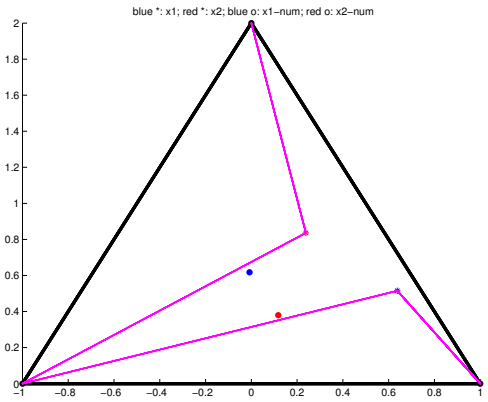

Figure 2: Inexact recovery, case 3.

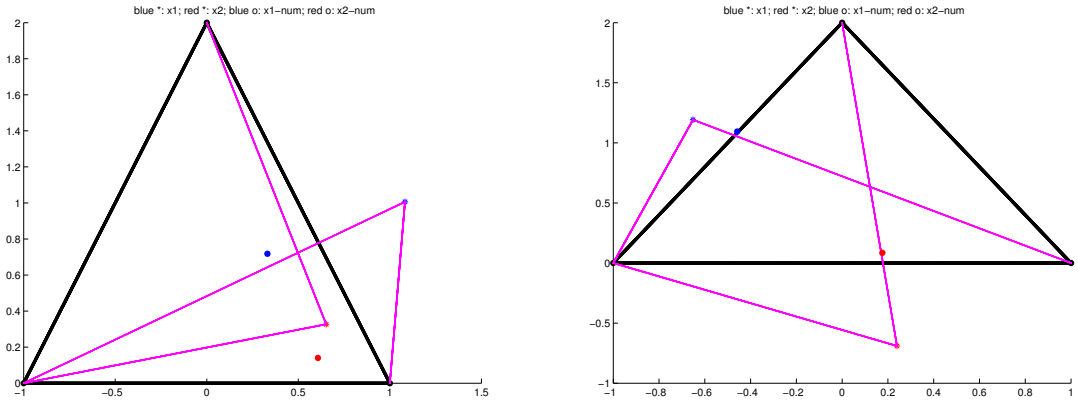

Figure 3: Left: inexact recovery, case 4. Right: inexact recovery, case 4.

We see that apart from the first case, we almost always lose exact recovery.

Also notice that switching the generation of x_1 and x_2 (i.e. first generate x_2 inside the convex hull of a_1, a_2, a_3 and then generate x_1 inside the convex hull of x_2, a_1, a_2) still results in the same observations, i.e. we obtain exact recovery if x_2 is also inside the convex hull of x_1, a_2, a_3, and vice versa.

The code is attached below:

```
clear all; close all;
%% Initialization
A = [1,-1,0;0,0,2];
%%% 1) random initialization choice (remove comment to enable)
% x1 = randn(2, 1);
% x2 = randn(2, 1);
%%% 2) random initialization choice inside the conv-hull (remove comment to
%%% enable)
```

```
% lambda1 = rand(3,1);
% x1 = A * lambda1 / sum(lambda1);
% lambda2 = rand(3,1);
% x2 = A * lambda2 / sum(lambda2);
%%% 2) special initialization choice (add comment to disable)
lambda1 = rand(3,1);
x1 = A * lambda1 / sum(lambda1);
lambda2 = rand(3,1);
x2 = [x1, A(:,2:3)] * lambda2 / sum(lambda2);
%% Plot the figure
scatter(A(1,:),A(2,:), 'k', 'filled');
hold on;
scatter(x1(1), x1(2), 'r*');
scatter(x2(1), x2(2), 'b*');
scatter(linspace(-1,1,1000),zeros(1,1000),5,'k');
scatter(linspace(-1,0,1000),2+2*linspace(-1,0,1000),5,'k');
scatter(linspace(0,1,1000),2-2*linspace(0,1,1000),5,'k');
scatter(linspace(-1,x2(1),1000), ...
    x2(2)/(x2(1)+1)*(linspace(-1,x2(1),1000)+1),2,'m*');
scatter(linspace(x2(1),1,1000), ...
    x2(2)/(x2(1)-1)*(linspace(x2(1),1,1000)-1),2,'m*');
scatter(linspace(-1,x1(1),1000), ...
    x1(2)/(x1(1)+1)*(linspace(-1,x1(1),1000)+1),2,'m*');
scatter(linspace(0,x1(1),1000), ...
    (x1(2)-2)/x1(1)*(linspace(0,x1(1),1000))+2,2,'m*');
%% data generation
d11 = norm(x1-A(:,1));
d12 = norm(x1-A(:,2));
d22 = norm(x2-A(:,2));
d23 = norm(x2-A(:,3));
d12h = norm(x1-x2);
%% SOCP
cvx_begin
variables z1(2) z2(2)
minimize(0)
subject to
norm(z1-A(:,1))<=d11
norm(z1-A(:,2))<=d12
norm(z2-A(:,2))<=d22
norm(z2-A(:,3))<=d23
norm(z1-z2)<=d12h
cvx_end
fprintf('x1 error = %3.4e\n', norm(z1-x1));
fprintf('x2 error = %3.4e\n', norm(z2-x2));
scatter(z1(1), z1(2), 'ro', 'filled');
scatter(z2(1), z2(2), 'bo', 'filled');
title('red *: x1; blue *: x2; red o: x1-num; blue o: x2-num');
hold off
```

(b) For the SDP case, the recovery is still not always exact even if both sensors are inside the convex hull of the anchors a_1, a_2, a_3. But there are *more chances of recovering exactly than the SOCP formulation.*

When both x_1, x_2 are inside the convex hull of a_1, a_2, a_3, as long as one of the following two cases holds: 1) x_1 is inside the convex hull of x_2, a_1, a_2; 2) x_2 is inside the convex hull of x_1, a_2, a_3, then we obtain exact recovery. If these are violated, then we may need some algebraic characterizations (see the remark in the ned). Again, we will validate this claim by numerical experiments.

In the numerical experiments, we again provide three options of generating sensors x_1, x_2 for facility of usage. The first is to generate both by `randn.m`. The second is to *start by generating x_1 randomly using `randn.m`*, and then generate x_2 inside the convex hull of x_1, a_2, a_3. The third is to generate x_1 and x_2 independently inside the convex hull of a_1, a_2, a_3.

As described above, we obtain exact recovery in the second case as long as x_2 is also inside the convex hull of a_1, a_2, a_3. Notice that again, switching the order of generating x_1 and x_2 (i.e. first generating x_2 using `randn.m` and then generate x_1 inside the convex hull of x_2, a_1, a_2) leads to the same observations.

The trickier case is when x_1, x_2 are not both inside the convex hull of a_1, a_2, a_3, but (exactly) one of 1) and 2) holds. In this case, we sometimes get exact recovery while sometimes not. But if we check the rank of a dual optimal slack matrix (see definition in the end remark), we will see that we obtain exact recovery iff its rank is $n = 2$.

In all other cases, we (almost) always lose exact recovery.

We showcase three possible situations below with both x_1, x_2 inside the convex hull of a_1, a_2, a_3 in the following figures:

(1) $x_1 \in \text{convhull}(x_2, a_1, a_2), x_2 \in \text{convhull}(x_1, a_2, a_3)$, $(x_1, x_2 \in \text{convhull}(a_1, a_2, a_3))$;

(2) $x_1 \notin \text{convhull}(x_2, a_1, a_2), x_2 \in \text{convhull}(x_1, a_2, a_3)$, $x_1, x_2 \in \text{convhull}(a_1, a_2, a_3)$;

(3) $x_1 \notin \text{convhull}(x_2, a_1, a_2), x_2 \notin \text{convhull}(x_1, a_2, a_3)$, $x_1, x_2 \in \text{convhull}(a_1, a_2, a_3)$;

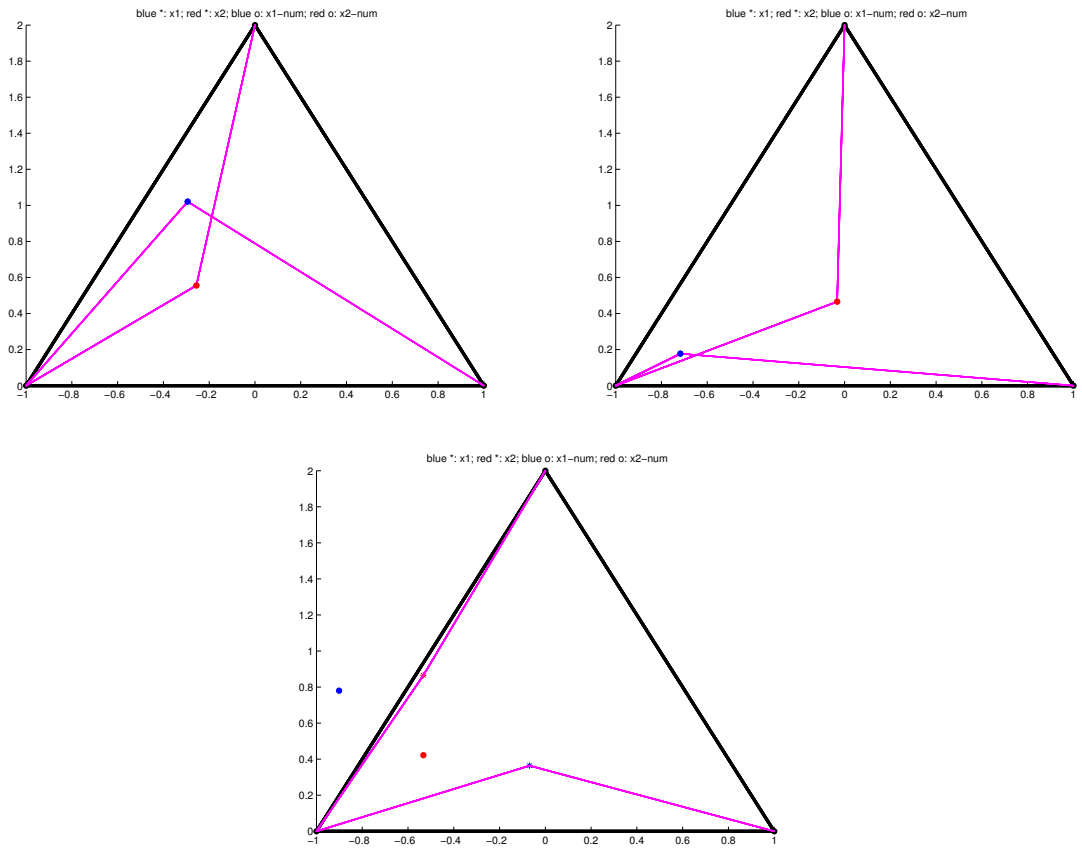

Figure 4: Top left: exact recovery, case 1. Top right: exact recovery, case 2. Bottom: inexact recovery, case 3.

This validates our claim. But when one of x_1, x_2 are outside the convex hull of a_1, a_2, a_3, things become much trickier:

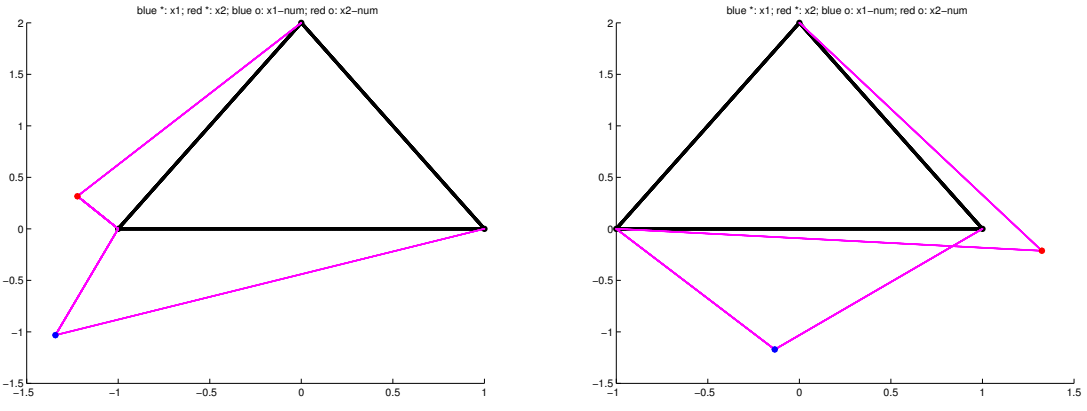

Figure 5: Exact recovery when both 1) and 2) are violated.

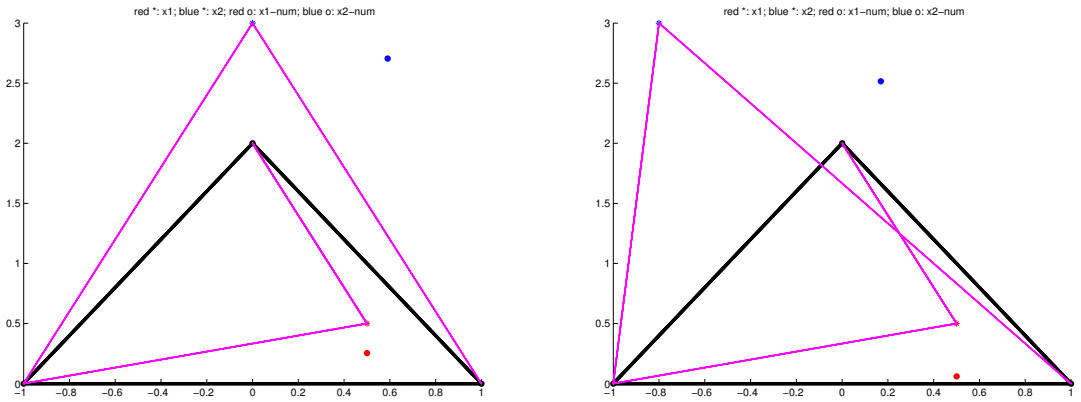

Figure 6: Inexact recovery when one of 1) and 2) holds.

Remark: The special cases which can not be easily characterized by convex hull inclusions can actually be (still partly) explained using Theorem 2 on slide 14 of lecture 8. In fact, the exact recovery is obtained if the optimal dual slack matrix (which can be retrieved e.g. using `dual variable` command in CVX) has rank n. This implies that any primal optimal/feasible solution Z has rank 2, and hence gives exact recovery.

The code is attached below:

```
clear all; close all;
%% Initialization
A = [1,-1,0;0,0,2];
%%% 1) random initialization choice (remove comment to enable)
% x1 = randn(2, 1);
% x2 = randn(2, 1);
%%% 2) random initialization choice inside the conv-hull (remove comment to
%%% enable)
% lambda1 = rand(3,1);
% x1 = A * lambda1 / sum(lambda1);
% lambda2 = rand(3,1);
% x2 = A * lambda2 / sum(lambda2);
%%% 3) special initialization choice (add comment to disable)
lambda1 = rand(3,1);
x1 = A * lambda1 / sum(lambda1);
lambda2 = rand(3,1);
x2 = [x1, A(:,2:3)] * lambda2 / sum(lambda2);
%% Plot the figure
scatter(A(1,:),A(2,:), 'k', 'filled');
hold on;
scatter(x1(1), x1(2), 'r*');
scatter(x2(1), x2(2), 'b*');
scatter(linspace(-1,1,1000),zeros(1,1000),5,'k');
scatter(linspace(-1,0,1000),2+2*linspace(-1,0,1000),5,'k');
scatter(linspace(0,1,1000),2-2*linspace(0,1,1000),5,'k');
scatter(linspace(-1,x2(1),1000), ...
```

```
    x2(2)/(x2(1)+1)*(linspace(-1,x2(1),1000)+1),2,'m*');
scatter(linspace(x2(1),1,1000), ...
    x2(2)/(x2(1)-1)*(linspace(x2(1),1,1000)-1),2,'m*');
scatter(linspace(-1,x1(1),1000), ...
    x1(2)/(x1(1)+1)*(linspace(-1,x1(1),1000)+1),2,'m*');
scatter(linspace(0,x1(1),1000), ...
    (x1(2)-2)/x1(1)*(linspace(0,x1(1),1000))+2,2,'m*');
%% data generation
d11 = norm(x1-A(:,1));
d12 = norm(x1-A(:,2));
d22 = norm(x2-A(:,2));
d23 = norm(x2-A(:,3));
d12h = norm(x1-x2);
%% SDP
a1 = A(:,1);
a2 = A(:,2);
a3 = A(:,3);
cvx_begin
variable Z(4,4) semidefinite
minimize(0)
subject to
Z(1:2,1:2) == eye(2, 2);
%%% constraint formulation 1
% [a1;-1;0]' * Z * [a1;-1;0] == d11^2;
% [a2;-1;0]' * Z * [a2;-1;0] == d12^2;
% [a2;0;-1]' * Z * [a2;0;-1] == d22^2;
% [a3;0;-1]' * Z * [a3;0;-1] == d23^2;
% [0;0;1;-1]' * Z * [0;0;1;-1] == d12h^2;
%%% constraint formulation 2
sum(sum(([a1;-1;0]*[a1;-1;0]').* Z)) == d11^2;
sum(sum(([a2;-1;0]*[a2;-1;0]').* Z)) == d12^2;
sum(sum(([a2;0;-1]*[a2;0;-1]').* Z)) == d22^2;
sum(sum(([a3;0;-1]*[a3;0;-1]').* Z)) == d23^2;
sum(sum(([0;0;1;-1]*[0;0;1;-1]').* Z)) == d12h^2;
cvx_end
z1 = Z(1:2,3);
z2 = Z(1:2,4);
fprintf('x1 error = %3.4e\n', norm(z1-x1));
fprintf('x2 error = %3.4e\n', norm(z2-x2));
scatter(z1(1), z1(2), 'ro', 'filled');
scatter(z2(1), z2(2), 'bo', 'filled');
title('red *: x1; blue *: x2; red o: x1-num; blue o: x2-num');
hold off
```

10. (10') For the Maze Runner example in Lecture Note #1, suppose that the blue-action at State 3 has a probability 0.5 leading to State 4 and 0.5 leading to State 5; and the only action at State 5 leads to State 0. Reformulate the MDP-LP problem with $\gamma = 0.9$ and solve it using any LP solver.

Sample Solution: LP formulation

$$\begin{aligned}
\text{maximize}_{\mathbf{y}} \quad & y_0 + y_1 + y_2 + y_3 + y_4 + y_5, \\
\text{subjectto} \quad & y_0 \le \min\{0 + \gamma y_1, 0 + \gamma(0.5y_2 + 0.25y_3 + 0.125y_4 + 0.125y_5)\}, \\
& y_1 \le \min\{0 + \gamma y_2, 0 + \gamma(0.5y_3 + 0.25y_4 + 0.25y_5)\}, \\
& y_2 \le \min\{0 + \gamma y_3, 0 + \gamma(0.5y_4 + 0.5y_5)\}, \\
& y_3 \le \min\{0 + \gamma y_4, 0 + \gamma(0.5y_4 + 0.5y_5)\}, \\
& y_4 \le 1 + \gamma y_5, \\
& y_5 \le 0 + \gamma y_0.
\end{aligned} \tag{1}$$

Solution

$$\begin{aligned}
y_0^* &= 0.747207793362298, \quad & \pi_0^* &= \text{Red}. \\
y_1^* &= 0.830230881521939, \quad & \pi_1^* &= \text{Red}. \\
y_2^* &= 0.922478757256612, \quad & \pi_2^* &= \text{Red}. \\
y_3^* &= 1.024976396962329, \quad & \pi_3^* &= \text{Blue}. \\
y_4^* &= 1.605238312580964, & & \\
y_5^* &= 0.672487014018313. & &
\end{aligned} \tag{2}$$

Sample code:

```
gamma = 0.9

cvx_begin
    variables y0 y1 y2 y3 y4 y5
    maximize y0 + y1 + y2 + y3 + y4 + y5
    subject to
       y0 <= 0 + gamma * y1
       y0 <= 0 + gamma * (0.5 * y2 + 0.25 * y3 + 0.125 * y4 + 0.125 * y5)
       y1 <= 0 + gamma * y2
       y1 <= 0 + gamma * (0.5 * y3 + 0.25 * y4 + 0.25 * y5)
       y2 <= 0 + gamma * y3
       y2 <= 0 + gamma * (0.5 * y4 + 0.5 * y5)
       y3 <= 0 + gamma * y4
       y3 <= 0 + gamma * (0.5 * y4 + 0.5 * y5)
       y4 <= 1 + gamma * y5
       y5 <= 0 + gamma * y0
cvx_end
```