A Theorist's Toolkit **(CMU 18-859T, Fall 2013)**

Lecture 15: Semidefinite Programming

October 28, 2013

Lecturer: Ryan O'Donnell *Scribe: Michael Nugent*

1 The Ellipsoid Algorithm

Recall from the previous lectures that the problem **Linear Programming** (**LP**) is solvable in polynomial time. The Ellipsoid Algorithm (hereafter, referred to as Ellipsoid) is a polynomial time algorithm that solves LP. Recall that solving LP reduces to the following (simpler question): Given

$$K = \left\{ x : \begin{array}{c} Ax \le b \\ -R \le x_i \le R \quad \forall i \in [n] \end{array} \right\}$$

is $K = \emptyset$? [Here $A \in \mathbb{Q}^{n \times n}$, $b \in \mathbb{Q}^m$, and $R \in \mathbb{R}$ and R is exponential in the input size, so R can be written in $\mathrm{poly}(\langle A \rangle, \langle b \rangle)$ bits]. Recall that Ellipsoid only needs to output $x \in K$ if $K \neq \emptyset$, and do nothing if $K = \emptyset$.

1.1 Reduction to Robust LP

Before we proceed, we actually need one additional reduction to make the $K \neq \emptyset$ case more robust: we want to modify the problem such that Ellipsoid only needs to output $x \in K$ if K contains some tiny cube in $\mathbb{R}^n$ (i.e., K is full-dimensional).

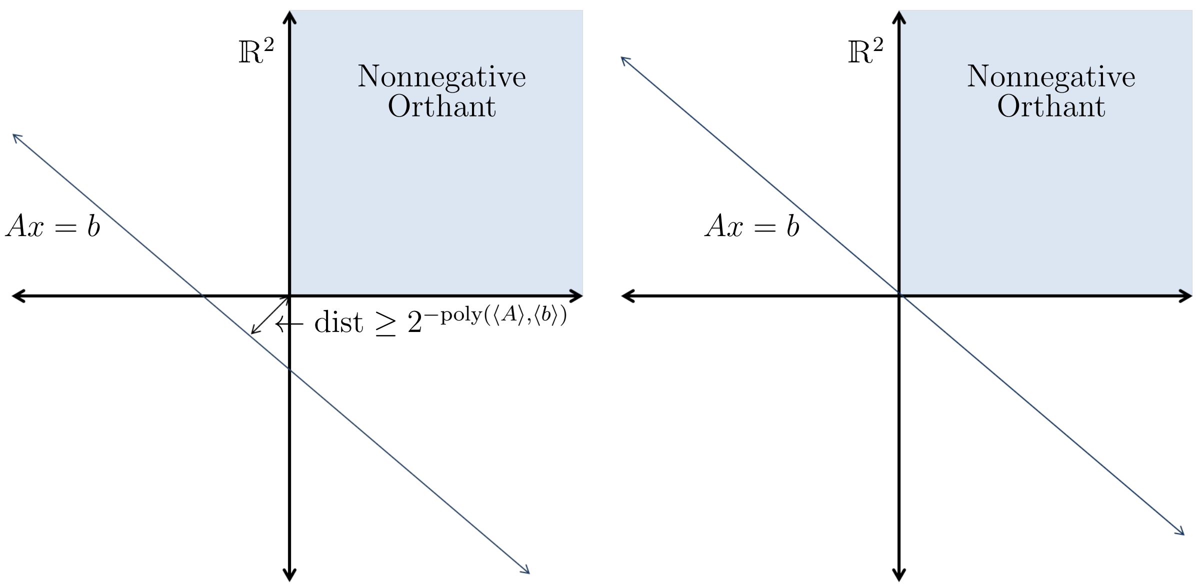
Claim 1.1. *LP further reduces to* ***Robust LP****: Given an input to LP, and additionally $r \in \mathbb{R}$ such that r can be written in poly($\langle A \rangle, \langle b \rangle$) bits*

- *If $K \neq \emptyset$ and K contains a cube in $\mathbb{R}^n$ of side length r, output some $x \in K$.*
- *Otherwise, output nothing.*

Clearly, if we can solve LP, we can solve Robust LP, so we just need to show the other direction.

"Proof" of other direction. Suppose we're given K as an input to LP. First we convert K to the equivalent form K', which has some nice properties:

$$K' = \left\{ x : \begin{array}{c} Ax = b \\ x_i \ge 0 \quad \forall i \in [n] \end{array} \right\}$$



(a) The case when $K' = \emptyset$.   (b) The case when $K' \neq \emptyset$.

Figure 1: The plot of the solution space of K' (assuming $n = 2$)

In particular, the solution space is now the intersection of the hyperplane $H = \{x : Ax = b\}$ with the nonnegative orthant. See Figure 1.

First, suppose $K' = \emptyset$. Then the distance between H and and the positive orthant cannot be arbitrarily small, since if there is a solution it can be written in $\mathrm{poly}(\langle A\rangle, \langle b\rangle)$ bits. Thus, we can relax all constraints by some r that is exponentially small in $\langle A\rangle$ and $\langle b\rangle$. In other words, we can replace the constraint $Ax = b$ with $b - \vec{r} \leq Ax \leq b + \vec{r}$, where $\vec{r}$ is the n-dimensional vector where every entry is r. This "thickens" the hyperplane in both cases, and preserves emptiness or nonemptiness. But now K' is full-dimensional, so it contains a tiny cube of side length r. See Figure 2.

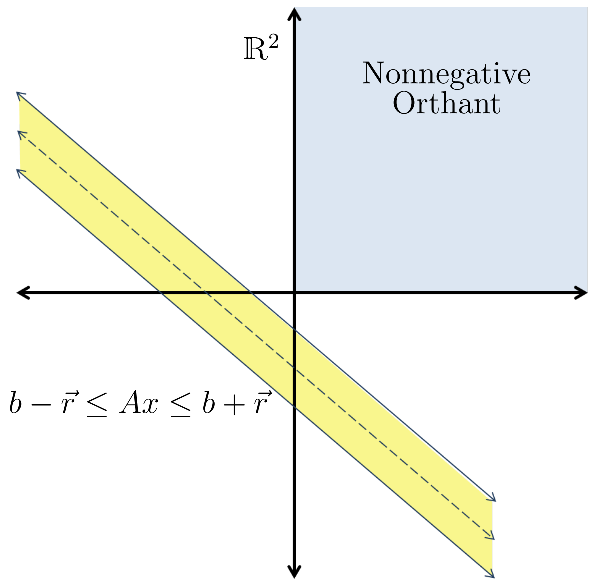
□

1.2 The Ellipsoid Algorithm

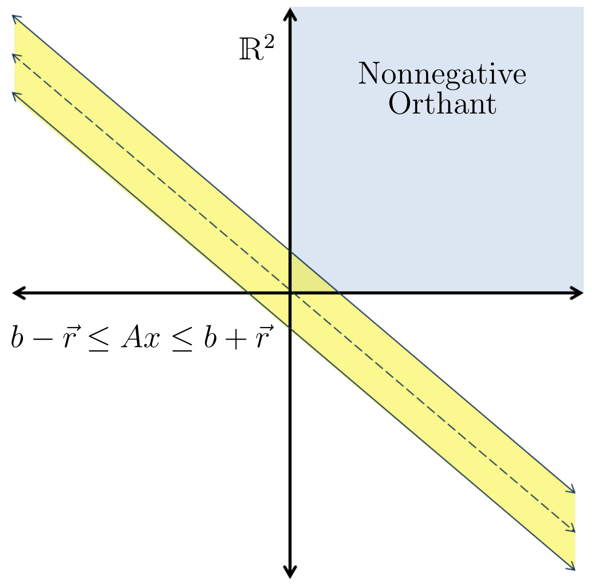
So, we just need to solve Robust LP, which Ellipsoid can do. Given R, r, and the linear inequalities, Ellipsoid finds a point inside K if one exists. The runtime of Ellipsoid is $\mathrm{poly}\big(n, \langle A\rangle, \langle b\rangle, \log R, \log \frac{1}{r}\big)$. Ellipsoid's name comes from the fact that throughout it's execution it maintains an ellipsoid containing K (assuming $K \neq \emptyset$). An ellipsoid, the generalization of an ellipse to higher dimensions, is a sort of "stretched ball" (i.e., in n dimensions it is the result of a linear transformation applied to an n-dimensional ball).

So to recap, Ellipsoid is an iterative algorithm that maintains the ellipsoid Q and the invariant that, at each iteration, $K \subseteq Q$.

Note that once we get the separating hyperplane $a \cdot c \geq b$, we know that K lies in an "ellipsoidal chunk", i.e., the set $Q \cap \{x : a \cdot x \geq b\}$. This, however, would be messy to maintain



(a) The case when $K' = \emptyset$. Note that the solution space is still empty.

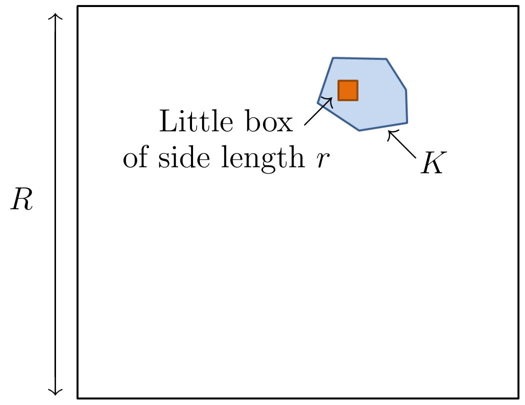


(b) The case when $K' \neq \emptyset$. Note that the solution space is now full-dimensional.

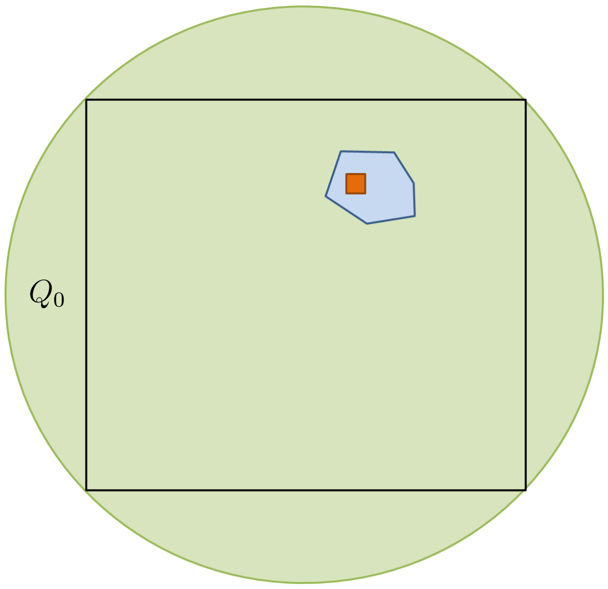
Figure 2: The plot of the solution space of K' after relaxing the constraints by $\vec{r}$. (assuming $n = 2$)

```
Initialize $Q$ to be the sphere with every vertex of the box of side length $R$ on it
repeat
    if The center of $Q$ is in $K$ // (this can be easily checked using the
        linear inequalities).
    then
        We're done! Return the point.
    else
        Get a "separating hyperplane" $a \cdot x \geq b$ (i.e., a halfspace) to separate the
        center of $Q$ from $K$.
        Compute $Q'$ to be the smallest ellipsoid containing $Q \cap \{x : a \cdot x \geq b\}$.
        Set $Q = Q'$.
    end
until Until $Q$ is too small
```

Algorithm 1: The Ellipsoid Algorithm

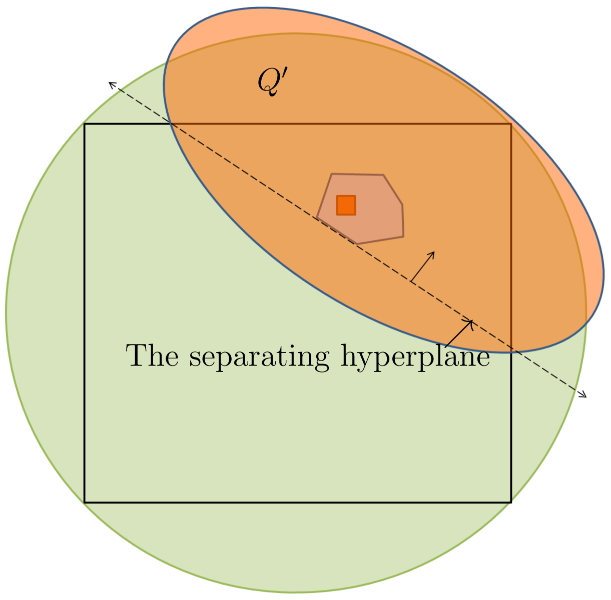


(a) The input to the Ellipsoid Algorithm.

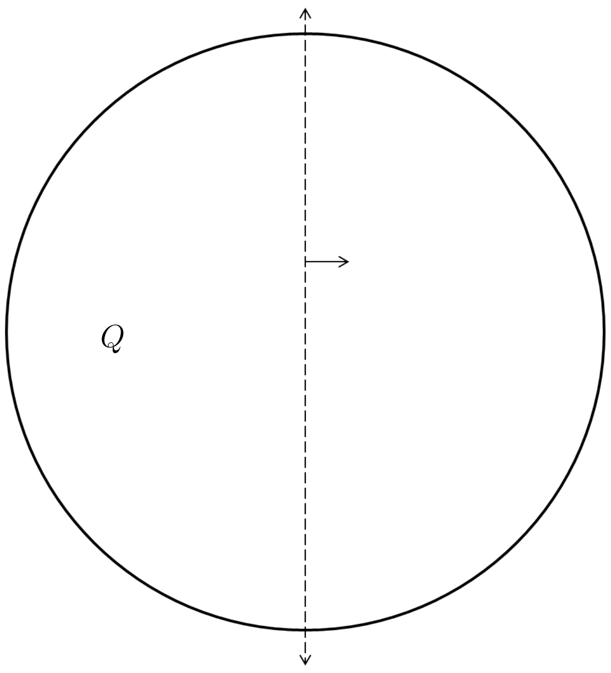


(b) The initial ellipsoid calculated by the Ellipsoid Algorithm.

Figure 3: The beginning of the Ellipsoid Algorithm.



(a) The first separating hyperplane and Q' of Ellipsoid.



(b) The worst case for the Ellipsoid Algorithm: Q is a ball and the separating hyperplane goes right through the center.

Figure 4

and use, so instead we calculate the smallest ellipsoid containing $Q \cap \{x : a \cdot x \geq b\}$, which is far simpler.

1.3 Analysis of the Ellipsoid Algorithm

One could imagine that Ellipsoid would work correctly if run forever, but we need to show that it terminates in a polynomial number of steps.

Lemma 1.2. *(An easy Lemma)* $\mathrm{vol}(Q') \leq \left(1 - \frac{1}{3n}\right) \mathrm{vol}(Q')$

Proof sketch. The worst case is when Q is a ball, and the halfspace goes directly through it's center. □

This lemma says that the ellipsoid gets a little bit smaller at each step. Thus, after $O(n)$ iterations, the volume of the maintained ellipsoid halves. If Q_0 is the initial ellipsoid, we have that $\mathrm{vol}(Q_0) \leq R^n n^{O(n)}$ and $\mathrm{vol}(K) \geq r^n$. Thus, we have that the number of iterations t until Ellipsoid completes is

$$t = O(n) \log \left(\frac{R^n n^{O(n)}}{r^n} \right) = \mathrm{poly}(n) \log \left(\frac{R}{r} \right)$$

which is polynomial in the input size.

Note that we assumed K was given as input, but we really only checked if a point was in K, and if not we found a separating hyperplane.

Key Observation: The algorithm really just needs as inputs R, r, and a *separation oracle.*

So we don't need the LP explicitly, we can just "imagine" it as long as we can query the separation oracle in polynomial time.

2 Semidefinite Programming

Ellipsoid also has a generalization called **Semidefinite Programming** (**SDP**). In the previous lecture, we solved the Min Cut problem in polynomial time using a LP. Now we'll see that the Max Cut problem is "motivation" for SDP. Unlike Min Cut, the Max Cut problem is NP-Hard, so we can't just right a LP for it (in the derandomization lecture we saw that a random cut cut at least half of the edges).

The input to the Max Cut problem is a graph $G = (V, E)$. The goal is to find $f : V \to \{0, 1\}$ to maximize $\mathbf{Pr}_{v \sim w \in E}[f(v) \neq f(w)]$.

Well, we can try to write a LP for it anyway. In the ILP, we'll have variables x_v to indicate whether v is in or out of the cut.

ILP	**LP**	
$x_v \in \{0,1\}$	$0 \leq x_v \leq 1$	$v \in V$
$y_{vw} \in \{0,1\}$	$0 \leq y_{vw} \leq 1$	$v, w \in V$

$$\max \operatorname{avg}_{v \sim w \in E}\{y_{vw}\}$$

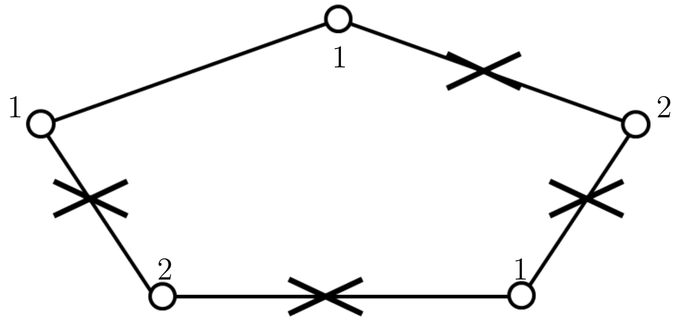


Figure 5: An example input to the Max Cut problem. Here the label of each vertex is the set it is put in, and the X's denote the edges cut. In this case, the max cut is $\frac{4}{5}$.

The intent here is that somehow we want to encode $y_{vw} = \mathbf{1}[x_v \neq x_w]$, i.e., y_{vw} is 1 whenever edge $v \sim w$ is cut. For the Min Cut problem, this is do-able:

$$y_{vw} \geq |x_v - x_w| \iff y_{vw} \geq x_v - x_w \text{ and } y_{vw} \geq x_w - x_v$$

This is successful because the goal is to minimize the y's. But for Max Cut we want $y_{vw} \leq |x_v - x_w|$, but we cannot do the same trick to encode this! So we are kind of stuck.

2.1 Semidefinite Programming Idea

The researchers C. Delorme and S. Poljak [DP93] had a smart idea about this. First, make a notation change to $x_v \in \{-1, 1\}$. Consider the following program:

$$x_v \in \{-1, 1\} \tag{1}$$
$$y_{vw} = x_v x_w \tag{2}$$

$$LP \begin{cases} \max\limits_{v \sim w \in E} \operatorname{avg} \left\{ \frac{1}{2} - \frac{1}{2} y_{vw} \right\} & \text{(3a)} \\ y_{vw} = y_{wv} & \text{(3b)} \\ y_{vv} = 1 & \text{(3c)} \end{cases}$$

Constraints (1) and (2) plus the objective (3a) *exactly* capture the Max Cut problem, since $\frac{1}{2} - \frac{1}{2} y_{vw} = \mathbf{1}[x_v \neq x_w]$. The constraints (3a) and (3b) must clearly hold for any solution. The objective (3a) plus constraints (3b) and (3c) define a LP, but not a very good one since we could, for example, set $y_{vw} = -1000000$ (hereafter (3b) and (3c) are referred to as the LP constraints, and combined with (3a) are the LP). For this LP to work, we want to enforce $y_{vw} = x_v x_w$ using linear constraints. What we really want to enforce is the **MOMENT** constraint:

$$\exists (x_v) \text{ s.t. } y_{v,w} = x_v x_w \quad \forall\, v, w \in V \tag{\textbf{MOMENT}}$$

This would allow us to drop the constraint $x_v \in \{-1, 1\}$, so the LP constraints in addition to MOMENT exactly captures Max Cut too.

Idea: Try to enforce MOMENT with linear inequalities.

Suppose you hand off the LP to Ellipsoid (which we now think of as a black box). The LP is clearly unbounded, so Ellipsoid will return something along the lines of $y_{vw} = -2^{\text{poly}(n)}$ for $v \neq w$ and $y_{vv} = 1$ $\forall v, w \in V$. But then we can say, "Just a moment, I forgot one constraint:"

$$\sum_{v,w \in V} y_{vw} \geq 0. \tag{4}$$

Basically, we just "surprised" Ellipsoid with a new hyperplane constraint, but Ellipsoid can handle that. Why is (4) a valid constraint?

Claim 2.1. *MOMENT implies constraint (4).*

Proof.

$$\sum_{v,w \in V} y_{vw} = \sum_{v,w \in V} x_v x_w = \left(\sum_{v \in V} x_v \right)^2 \geq 0$$

□

Since (4) is a valid constraint implied by MOMENT, so we add that constraint to the LP since we can't add MOMENT directly.

Now imagine Ellipsoid continues running begins to output another solution:

$$\begin{aligned} y_{11} &= 1 \\ y_{12} &= 100 \\ y_{21} &= 100 \\ y_{22} &= 1 \\ y_{13} &= \ldots \end{aligned}$$

but we interrupt it and say "Just a moment, I forgot a constraint:"

$$y_{11} - y_{12} - y_{21} + y_{22} \geq 0 \tag{5}$$

(Of course, now Ellipsoid is getting a little annoyed (sorry Ellipsoid!) because we again sprang a new constraint on it.)

Claim 2.2. *MOMENT implies constraint (5)*

Proof.

$$y_{11} - y_{12} - y_{21} + y_{22} = x_1x_1 - x_1x_2 - x_2x_1 + x_2x_2 = (x_1 - x_2)^2 \geq 0$$

□

Then Ellipsoid responds with "Just kidding! I really meant $y_{12} = -100, y_{21} = -100, \ldots$", but then we respond that we forgot the constraint

$$y_{11} + y_{12} + y_{21} + y_{22} \geq 0 \tag{6}$$

Claim 2.3. *MOMENT implies constraint (6)*

Proof.

$$y_{11} + y_{12} + y_{21} + y_{22} = x_1x_1 + x_1x_2 + x_2x_1 + x_2x_2 = (x_1 + x_2)^2 \geq 0$$

□

Note that

- Adding constraint (5) to the LP implies that $1 - 2y_{12} + 1 \geq 0$, which is true if and only if $y_{12} \leq 1$.
- Adding constraint (6) to the LP implies that $1 + 2y_{12} + 1 \geq 0$, which is true if and only if $y_{12} \geq -1$.

So even constraints (5) and (6) are beginning to constrain the y variables in an interesting way.

Back to the dialogue with Ellipsoid. We could keep surprising Ellipsoid with more constraints, but note that, for general constants $c_v \in \mathbb{R}$ we have that

$$\left(\sum_{v\in V} c_v x_v\right)^2 \geq 0 \implies \sum_{v,w\in V} c_v c_w x_v x_w \geq 0$$

Therefore, it follows from the MOMENT constraint that

$$\sum_{v,w\in V} c_v c_w y_{vw} \geq 0 \qquad \text{(SDP) (7)}$$

and, for constant c's, these are linear constraints in the y's, exactly like those we've been adding. It would be nice if we could add all of these inequalities. There are infinitely many inequalities of the form (7), but this is not necessarily a problem as long as we can solve the separation problem. This is the idea behind Semidefinite Programming. We refer to (7) as the SDP constraints.

2.2 Formal Definition of Semidefinite Programming

First, here are some key facts about Semidefinite Programming

Fact 2.4. *We can solve the LP with the SDP constraints efficiently (i.e., even though there's an infinite number of constraints, we can solve the problem of finding a solution satisfying both LP and SDP in polynomial time).*

Fact 2.5. *MOMENT implies SDP, but SDP does not imply MOMENT (i.e., LP with SDP does not exactly capture the Max Cut problem).*

Fact 2.6. *LP with SDP is a pretty good relaxation for Max Cut (in particular, it finds a Max Cut at least 88% of the optimal).*

We just showed the first part of Fact 2.5 (MOMENT implies SDP), and the second part of Fact 2.5 is an easy exercise. Fact 2.6 will be shown next lecture, so we focus on proving Fact 2.4: Given a set of y's, we need to see if all SDP constraints are satisfied, and find a set of c_v's corresponding to an unsatisfied constraint if not.

Definition 2.7. A symmetric matrix $Y \in \mathbb{R}^{n\times n}$ is *positive semidefinite* (*PSD*), written $Y \succeq 0$, if for all $c \in \mathbb{R}^n$,

$$\sum_{i,j\in[n]} c_i c_j y_{ij} \geq 0$$

i.e.,

$$\begin{bmatrix} \longleftarrow c^T \longrightarrow \end{bmatrix} \begin{bmatrix} \\ & Y & \\ \\ \end{bmatrix} \begin{bmatrix} \uparrow \\ c \\ \downarrow \end{bmatrix} \geq 0$$

For example, the Laplacian of any graph is PSD. We want to check if the solution to Ellipsoid is PSD, and, if not, find c such that $c^T Y c < 0$. The following theorem was proved and referred to in the footnote of Problem 3 of Homework 3.

Theorem 2.8. *There exists a polynomial($\langle Y \rangle$)-time algorithm that finds $c \in \mathbb{R}^n$ such that $c^T Y c < 0$ if such a c exists, and outputs "Y is PSD" if not.*

Proof Sketch. We do "Symmetric" Gaussian Elimination[1] on Y and continue as long as the diagonal is nonnegative. If at any point an entry on the diagonal is negative, we can stop and find c. If not, we have a proof that Y is PSD. More specifically, we are trying to write

$$Y = LDL^T = \underset{L}{\begin{bmatrix} 1 & \nwarrow & \uparrow & \nearrow \\ \nwarrow & 1 & 0 & \rightarrow \\ \leftarrow & * & \ddots & \searrow \\ \swarrow & \downarrow & \searrow & 1 \end{bmatrix}} \underset{D}{\begin{bmatrix} d_1 & \nwarrow & \uparrow & \nearrow \\ \nwarrow & d_2 & 0 & \rightarrow \\ \leftarrow & 0 & \ddots & \searrow \\ \swarrow & \downarrow & \searrow & d_n \end{bmatrix}} \underset{L^T}{\begin{bmatrix} 1 & \nwarrow & \uparrow & \nearrow \\ \nwarrow & 1 & * & \rightarrow \\ \leftarrow & 0 & \ddots & \searrow \\ \swarrow & \downarrow & \searrow & 1 \end{bmatrix}}$$

where the $*$ entries could be anything, and the d_i's are all nonnegative. If we find a negative d_i, since Y was being decomposed in this way, we can easily extract a c such that $c^T Y c < 0$. If we succeed in writing Y this way, then Y is PSD, since $Y = LDL^T$ and we have that

$$\begin{aligned} c^T Y c &= c^T LDL^T c = c^T L\sqrt{D}\sqrt{D}L^T c \\ &= \left(\sqrt{D}L^T c\right)\left(\sqrt{D}L^T c\right)^T = \left|\left|\sqrt{D}L^T c\right|\right|_2^2 \geq 0 \end{aligned}$$

□

[1]This is called "Cholesky Decomposition". The idea is to simultaneously zero out all entries but the first in both the first column *and* the first row at the same time. Then we move on to the second column and row, etc.

Therefore, there exists a polynomial-time separation oracle for whether or not a solution is PSD. Thus, we can use Ellipsoid to solve[2] the LP with PSD constraints (i.e., the LP with the additional constraint that Y is PSD).

Proposition 2.9. *A symmetric matrix Y is PSD if and only if $Y = U^T U$ for some matrix $U \in \mathbb{R}^{n\times n}$.*

Proof. As we saw in the previous proof, if Y is PSD then $Y = LDL^T = L\sqrt{D}\sqrt{D}L^T$, so we can set $U = L\sqrt{D}$. For the other direction, note that $c^T U^T = (Uc)^T$, so the ith entry of $c^T U^T$ is equal to the ith entry of Uc, so $c^T U^T U c$ is just a sum of squares, and hence nonnegative. $\square$

There are many good definitions of PSD. For example:

Proposition 2.10. *A symmetric matrix Y is PSD if and only if all the eigenvalues of Y are nonnegative.*

Recall that we have the following

$$\begin{array}{ccccc} \textbf{MOMENT} & & \not\Longleftarrow & & \textbf{SDP} \\ \exists x_i \in \mathbb{R} \text{ s.t. } y_{ij} = x_i x_j & \Longrightarrow & \sum_{i,j} c_i c_j y_{ij} \geq 0 \;\forall\, c \in \mathbb{R}^n & \Longleftrightarrow & \exists\, U \in \mathbb{R}^{n\times n} \text{ s.t. } Y = U^T U \end{array}$$

Even though SDP does not imply MOMENT, there is a similar condition that SDP is equivalent to (it's probably a good definition to remember).

Proposition 2.11. *A symmetric matrix Y is PSD if and only if there exists jointly random variables $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ such that $y_{ij} = \mathbf{E}[\boldsymbol{X}_i \boldsymbol{X}_j]$.*

Proof. ($\Longleftarrow$) Suppose these random variables exist, and y_{ij} is as defined. Then, for all $c \in \mathbb{R}^n$.

$$\sum_{i,j} c_i c_j y_{ij} = \sum_{i,j} c_i c_j \,\mathbf{E}[\boldsymbol{X}_i \boldsymbol{X}_j] = \mathbf{E}\left[\sum_{i,j} c_i x_i c_j x_j\right] = \mathbf{E}\left[\left(\sum_i c_i x_i\right)^2\right] \geq 0$$

($\Longrightarrow$) Given a symmetric PSD matrix Y, how do we get the random variables? Recall that $Y = U^T U$ for some $U \in \mathbb{R}^{n\times n}$. Now we must define $(\boldsymbol{X}_i)_{i\in[n]}$. Pick $k \sim [n]$ uniformly at random, and set $\boldsymbol{X}_i = \sqrt{n}U^T_{ik} = \sqrt{n}U_{ki}$ for all $i \in [n]$.

$$Y = U^T U = \underset{U^T}{\overset{\longleftarrow\; k \;\longrightarrow}{\begin{bmatrix} \qquad & \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} & \qquad \end{bmatrix}}} \underset{U}{\begin{bmatrix} \qquad & \qquad & \qquad \\ \\ \\ \\ \\ \end{bmatrix}}$$

[2]Of course, Ellipsoid also needs a full-dimensional input (little box of side length r) and a bound on the solution space (R). The setting of $y_{vv} = 1$ is not full-dimensional, so we need to relax the constraints to $1 - 2^{-\mathrm{poly}(\langle A\rangle,\langle b\rangle)} \leq y_{vv} \leq 1 + 2^{-\mathrm{poly}(\langle A\rangle,\langle b\rangle)}$. Henceforth we ignore this detail, but it is a technical point to keep in mind.

Then we have that

$$\mathbf{E}[\boldsymbol{X}_i \boldsymbol{X}_j] = \frac{1}{n} \sum_{k=1}^{n} \sqrt{n} U_{ik}^T \sqrt{n} U_{jk}^T = \sum_{k=1}^{n} U_{ik}^T U_{kj} = \left(U^T U\right)_{ij} = y_{ij}$$

$\square$

Here is one more useful definition of PSD:

Proposition 2.12. *A symmetric matrix Y is PSD if and only if there exists vectors $\vec{u}_1, \ldots, \vec{u}_n \in \mathbb{R}^n$ such that $y_{ij} = \langle \vec{u}_i, \vec{u}_j \rangle$.*

Proof. If Y is symmetric and PSD, then $Y = U^T U$, so let $\vec{u}_i$ be the ith column of U. If such $\vec{u}_i$ exist, then $Y = U^T U$, where $U = [\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_n]$, so Y is PSD. $\square$

References

[DP93] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993.