1 Semidefinite programming

Today we will discuss semidefinite programming (SDP) in particular; it is a special case of conic programming. The standard form of the primal is.

\[
\begin{align*}
\text{Min } & \quad C \cdot X \\
\text{s.t. } & \quad A_k \cdot X = b_k, \quad k = 1, \ldots, m \\
& \quad X \succeq 0 \quad (\equiv X \text{ positive semi-definite: } V^T X V \geq 0 \ \forall V)
\end{align*}
\]

We are here assuming that \(X\) is symmetric. The following fact is well-known from Linear Algebra:

**Fact 1** For symmetric \(X \in \mathbb{R}^{n \times n}\), the following statements are equivalent:

1. \(X \succeq 0\);
2. \(X\) has non-negative eigenvalues ;
3. \(X = V^T V\) for some \(V \in \mathbb{R}^{m \times n}, \ m \leq n\).

The dual of (1) is given by:

\[
\begin{align*}
\text{Max } & \quad b^T y \\
& \quad \sum_{k=1}^{m} y_k A_k + S = C, \\
& \quad S \succeq 0.
\end{align*}
\]

2 SDP and the central path

We now show how some of the interior-point methods for LP can be used for SDP as well. We need the following definitions:

**Definition 1**

\[
\begin{align*}
\mathcal{F}^0(P) & = \{X \in \mathbb{R}^{n \times n} : A_k \cdot X = b_k, \ k = 1, \ldots, m, \ X \succ 0\} \\
\mathcal{F}^0(D) & = \{(y, S) : \sum_{k=1}^{m} y_k A_k + S = C, \ S \succ 0\}
\end{align*}
\]

where \(X \succ 0 \equiv X \text{ positive definite, } v^T X v > 0, \ \forall v \in \mathbb{R}^n\).
Recall the barrier function for LP:

\[ B_\mu(X) = c^T x - \mu \sum_i \ln(x_i) = c^T x - \mu \ln \left( \prod_i x_i \right). \]

Recall that minimizing the function trades off minimizing the objective function versus staying in the interior of the feasible region (in particular, staying away from the constraints \( x_i = 0 \)).

We would like to have the same sort of function for SDP. The corresponding barrier function is

\[ B_\mu(X) = C \cdot X - \mu \ln(\det X). \]

Once again, minimizing the barrier function trades off minimizing the objective function versus staying in the interior of the feasible region: since \( \det X \) is the product of the eigenvalues, it stays away from zero precisely when the eigenvalues of \( X \) stay away from zero.

We can prove a theorem analogous to the one we proved for linear programming about how to find the minimizer of the barrier function. We will skip the proof.

**Theorem 1** If \( \mathcal{F}^0(P) \) and \( \mathcal{F}^0(D) \) are non-empty, a necessary and sufficient condition for \( X \in \mathcal{F}^0(P) \) to be the unique minimizer of \( B_\mu \) is that \( \exists (y, S) \in \mathcal{F}^0(D) \) such that:

1. \( \sum_{k=1}^m y_k A_k + S = C \)
2. \( A_k \cdot X = b_k, \quad k = 1, ..., m \)
3. \( XS = \mu I \)

Again, as in the case of linear programming, we will apply Newton’s method to find the minimizer of the barrier function \( B_\mu \). In particular, we want to find a zero of the function

\[ F(X, y, S) = \begin{pmatrix} \sum_{k=1}^m y_k A_k + S - C \\ A_1 \cdot X - b_1 \\ \vdots \\ A_k \cdot X - b_k \\ XS - \mu I \end{pmatrix}. \]

We apply Newton’s method by finding the Jacobian \( J(X, y, S) \) and repeatedly solving the following system for \( (\Delta X, \Delta y, \Delta S) \):

\[ J(X, y, S) \begin{pmatrix} \Delta X \\ \Delta y \\ \Delta S \end{pmatrix} + F(X, y, S) = 0. \]

Finding the Jacobian yields the following system:

\[ \sum_{k=1}^m (\Delta y_k) A_k + \Delta S = 0 \]
\[ A_k \cdot (\Delta X) = 0 \]
\[ S(\Delta X) + (\Delta S)X = -SX + \mu I \]
Thus we get the following algorithm exactly analogous to that for linear programming:

**Primal-Dual Interior-Point for SDP**

\[
(X^0, y^0, S^0) \leftarrow \text{initial feasible point } (x^0, s^0 > 0) \\
\mu^0 \leftarrow \frac{1}{n} X^0 \cdot S^0 \\
k \leftarrow 0 \\
\text{While } \mu^k > \epsilon \\
\begin{align*}
\sum_{i=1}^{m} (\Delta y^k_i) A_i + \Delta S^k &= 0 \\
S^k (\Delta X^k) + (\Delta S^k) X^k &= -S^k X^k + \sigma^k \mu^k I \\
(X^{k+1}, y^{k+1}, S^{k+1}) &\leftarrow (X^k, y^k, S^k) + \alpha^k (\Delta X^k, \Delta y^k, \Delta S^k) \\
\mu^{k+1} &\leftarrow \frac{1}{n} X^{k+1} \cdot S^{k+1} \\
k &\leftarrow k + 1
\end{align*}
\]

where \( \mu = \frac{1}{n} X \cdot S \) and \( \sigma \in [0, 1] \) is a centering parameter. As before, this algorithm leads to an \( O(\sqrt{n} \ln \frac{C}{\epsilon}) \) iteration algorithm to get from duality gap of \( C \) down to \( \epsilon \). Note that the iteration count depends on \( n \), even though the number of variables in the matrix is \( n^2 \).

### 3 An application: MAX CUT

We now show how to use semidefinite programming to obtain an approximation algorithm for finding a maximum cut (MAX CUT). Given input \( G = (V, E) \), weights \( w_{ij} \forall (i, j) \in E \), our goal is to find \( S \subseteq V \) that maximizes \( \sum_{(i, j) \in \delta(S)} w_{ij} \). An example of a cut can be seen in Figure 1.

![Figure 1: Example of a cut.](image-url)
We claim that the following is an SDP relaxation of MAX CUT.

\[
\text{Max } \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - x_{ij})
\]

\[
x_{ii} = 1, \quad \forall i = 1, \ldots, n = |v|
\]

\[
X = (x_{ij}) \succeq 0.
\]

For this to be a relaxation, we need to show that an optimal solution is feasible, and has objective function value equal to the weight of the edges in the optimal solution.

Suppose \( S^* \) is an optimal solution to the MAX CUT problem. Set

\[
z_i = \begin{cases} 
+1 & \text{if } i \in S^* \\
-1 & \text{otherwise}
\end{cases}
\]

If we let \( X^* = zz^T \) this implies that \( X^* \succeq 0 \). Moreover,

\[
X^*_{ij} = z_iz_j = \begin{cases} 
+1 & \text{is } i, j \in S^* \text{ or } i, j \text{ is in } S^* \\
-1 & \text{if exactly one of } i, j \text{ is in } S^*
\end{cases}
\]

Since it holds that \( X^*_{ii} = z_iz_i = 1 \) \( \forall i = 1, \ldots, n \) it follows that \( X^* \) is feasible. The objective function value is

\[
\frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - x^*_{ij}) = \sum_{(i,j) \in \delta(S^*)} w_{ij}
\]

since all edges \((i, j)\) that are not in the cut have \( x^*_{ij} = 1 \), while all edges \((i, j)\) in the cut have \( x^*_{ij} = -1 \). So if we solve SDP, get \( X \) of value \( Z^* \), then

\[
Z^* \geq \sum_{(i,j) \in \delta(S^*)} w_{ij} \equiv OPT.
\]

since the optimal solution to MAX CUT is feasible for the SDP.

Now, we want to solve the SDP relaxation and from it obtain a solution to the MAX CUT problem. W solve the SDP in polynomial time and get a solution \( X \) and let \( V \) be such that \( X = V^T V \). Let \( v_i \) be the \( i \)th column of \( V \), then \( X_{ij} = v_i^T v_j \). Note that \( x_{ii} = 1 \Rightarrow v_i^T v_i = \|v_i\|^2 = 1 \) so that \( v_i \) are a set of vectors in the unit ball. This is illustrated for two dimensions in Figure 2. Next pick a random vector \( r = (r_1, \ldots, r_n) \) where \( r_i \sim \mathcal{N}(0, 1) \). We get a solution to MAX CUT by setting \( i \in S \) iff \( r^T v_i \geq 0 \).

**Fact 2** \( r \) is spherically symmetric.

**Fact 3** Projection of \( r \) onto any 2D plane is still spherically symmetric.

**Lemma 2** \( Pr[(i, j) \in \delta(S)] = \frac{1}{\pi} \arccos(x_{ij}) \).
Figure 2: Example of some vectors $V_i$ in the 2D unit circle.

Figure 3: Example of a plane that cuts the sphere.

**Proof:** Consider the 2D plane containing $v_i$ and $v_j$. Let $r = r' + r''$ where $r'$ is the projection of $r$ onto plane and $r'' \perp$ to the plane. Then

$$
\Pr[\langle i, j \rangle \in \delta(S)] = \Pr[i \in S, j \notin S \text{ or } i \notin S, j \in S] \\
= \Pr[r^Tv_i \geq 0, r^Tv_j < 0 \text{ or } r^T < 0, r'^Tv_j \geq 0] \\
= \Pr[(r')^Tv_i \geq 0, (r')^Tv_j < 0 \text{ or } (r')^Tv_i < 0, (r')^Tv_j \geq 0] \\
= \frac{2\theta}{2\pi} = \frac{\theta}{\pi}
$$

This follows because $r'$ is spherically symmetric in the unit circle in the plane, and of the $2\pi$ possible orientations, $2\theta$ of them correspond to the event we are interested in; for this, the shaded region in Figure 5 is when exactly one of $i, j$ is in $S$.

We know that $v_i^Tv_j = \|v_i\|\|v_j\|\cos \theta = \cos \theta$ so that $\theta = \arccos(v_i^Tv_j) = \arccos(x_{ij})$. □

**Lemma 3** $\frac{1}{2} \arccos(x) \geq 0.87856 \cdot \frac{1}{2}(1 - x)$ for $-1 \leq x \leq 1$

**Theorem 4** In expectation, this algorithm returns a cut of weight $\geq 0.87856 \cdot \text{OPT}$. 28-5
Figure 4: Illustration of the decomposition of $r$ into $r'$ and $r''$. 

Figure 5: Illustration of the regions where exactly one of $i$ and $j$ are in $S$.

Proof:

$$E \left[ \sum_{(i,j) \in \delta(S)} w_{ij} \right] = \sum_{(i,j) \in E} w_{ij} \Pr[(i,j) \in \delta(S)]$$

$$= \sum_{(i,j) \in E} w_{ij} \frac{1}{\pi} \arccos(x_{ij})$$

$$\geq 0.87856 \cdot \frac{1}{2} \sum_{(i,j) \in E} w_{ij} (1 - x_{ij})$$

$$= 0.87856 \cdot Z^* \geq 0.87856 \cdot \text{OPT}.$$ 

It appears that SDP is strictly needed in order to obtain an approximation algorithm with a factor this close to 1. Using linear programming, it does not appear possible to get an approximation algorithm with an approximation factor better than $1/2$. 

$\square$