Chapter 3

Semidefinite Programming (SDP)

You should all be familiar with linear programs, the theory of duality, and the use of linear programs to solve a variety of problems in polynomial time; flow or routing problems are typical examples. While LPs are very useful, and can be solved very efficiently, they don't cover all possible situations.

In these lectures we study an extension of linear programming, semidefinite programming, which is much more general but still solvable in polynomial time.

3.1 Motivation: MAXCUT

Consider an undirected graph $G = (V, E)$. Then

$$\text{MAXCUT}(G) := \max_{x_i \in \{\pm 1\}, i \in V} \sum_{(i,j) \in E} \frac{1 - x_i x_j}{2}$$

is precisely the number of edges that are cut by the largest cut in the graph (a cut is a partition of the vertices into two sets). Replacing the constraints $x_i \in \{\pm 1\}$ by $x_i \in [-1, 1]$ does not change the optimum. This makes all the constraints linear, but the objective function itself is not; it is quadratic. A natural idea is to introduce auxiliary variables z_{ij} and reformulate the problem as

$$\begin{aligned} \max \quad & \sum_{(i,j) \in E} \frac{1 - z_{ij}}{2} \\ \text{s.t.} \quad & z_{ij} = x_i x_j \\ & -1 \leq x_i \leq 1. \end{aligned}$$

Now the objective function has been linearized, but the constraints are quadratic! Of course it shouldn't be too much of a surprise that we're having a hard time finding a simple form for MAXCUT(G), as the problem is NP-hard.

In particular this is not even a convex program. To see why, observe that both $z = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ and $z' = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ are feasible (for the first, take $x_1 = x_2 = 1$; for the second take $x_1 = 1, x_2 = -1$), but $(z + z')/2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ is not (for instance, because it has rank 2). But here's an interesting observation. What if we allow x_i to be a vector $\vec{x}_i$, of norm at most 1? Then we can factor $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = (\vec{e}_i \cdot \vec{e}_j)_{i,j}$. More generally, if $z_{ij} = \vec{x}_i \cdot \vec{x}_j$ and $z'_{ij} = \vec{x}'_i \cdot \vec{x}'_j$ then $\frac{1}{2}(z_{ij} + z'_{ij}) = \frac{1}{\sqrt{2}} \begin{pmatrix} \vec{x}_i & \vec{x}'_i \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} \vec{x}_j \\ \vec{x}'_j \end{pmatrix}$. This may sound strange, but let's see what we get. Since we allow a larger set for the variables we have a *relaxation*

$$\begin{aligned} \max \quad & \sum_{(i,j) \in E} \frac{1 - z_{ij}}{2} \\ \text{s.t.} \quad & z_{ij} = \vec{x}_i \cdot \vec{x}_j \\ & \|\vec{x}_i\|^2 \leq 1 \quad \forall i \\ & \vec{x}_i \in \mathbb{R}^d. \end{aligned} \tag{3.1}$$

What did we do? We relaxed the constraint that $x_i \in [-1, 1]$, i.e. x_i is a one-dimensional vector of norm at most 1, to allowing a vector $\vec{x}_i$ of arbitrary dimension (we use the $\vec{x}$ notation to emphasize the use of vectors, but soon we'll omit the arrow and simply write $x \in \mathbb{R}^d$ as usual). Note in particular that the feasible region for this problem is unbounded, and a priori it could be that higher and higher-dimensional vectors would give better and better objective values.

To see that this is not the case, we're going to rewrite (3.1) as an optimization problem that only involves a finite number of variables using the theory of positive semidefinite (PSD) matrices. Recall the following equivalent definitions for a PSD matrix Z:

Lemma 3.1. *A symmetric matrix $Z \in \mathbb{R}^{n \times n}$ is positive semidefinite if and only if any of the following equivalent conditions holds:*

(1) $\forall x \in \mathbb{R}^n$, $x^T Z x \geq 0$;

(2) $\exists X \in \mathbb{R}^{d \times n}$ such that $Z = X^T X$;

(3) All eigenvalues of Z are nonnegative;

(4) $Z = UDU^T$, where U is orthogonal and D is diagonal with nonnegative entries;

(5) $Z = \sum_i \lambda_i u_i u_i^T$, $\lambda_i \geq 0$, $\{u_i\}$ orthonormal basis of $\mathbb{R}^n$.

We denote $\mathcal{P}(\mathbb{R}^n) := \{Z \in \mathbb{R}^{n \times n} | Z \succeq 0\}$ the positive semidefinite cone (a cone is a convex set C such that $x \in C, \lambda \geq 0 \implies \lambda x \in C$).

Exercise 1. Prove that the five conditions in the definition are equivalent. You may use the spectral theorem for real symmetric matrices.

For $A, B \in \mathbb{R}^{n\times n}$ symmetric we'll write $A \succeq B$ when $A - B \succeq 0$. For $Z \in \mathbb{R}^{n\times n}$, we will use the notation

$$Y \bullet Z := \mathrm{Tr}(Y^T Z) = \sum_{i,j} Y_{ji} Z_{ji},$$

which defines an inner product on real $n \times n$ matrices.

Exercise 2. Show that if A, B and C are any three real symmetric matrices (of the same size) such that $A \succeq B$ and $C \succeq 0$, then $A \bullet C \geq B \bullet C$.

Using this notation we can write

$$\sum_{(i,j)\in E} \frac{1 - z_{ij}}{2} = \frac{|E|}{2} - \frac{1}{2} G \bullet Z,$$

where $G \in \{0,1\}^{n\times n}$ is the symmetrized adjacency matrix of the graph (G has a coefficient $1/2$ in each position (i,j) such that $\{i,j\}$ is an edge, and zeros elsewhere). The constraints $z_{ij} = \vec{x}_i \cdot \vec{x}_j$ are equivalent to the factorization $Z = X^T X$, where $X \in \mathbb{R}^{d\times n}$ has the $\vec{x}_i$ as its columns. The constraint $\|\vec{x}_i\|^2 \leq 1$ is equivalent to $E_i \bullet Z \leq 1$, where E_{ii} is a matrix with a 1 in the (i,i)-th entry and 0 everywhere else. Using the characterization of PSD matrices given in Lemma 3.1, the problem (3.1) can be reformulated as

$$\begin{aligned} \max \quad & \frac{|E|}{2} - \frac{1}{2} G \bullet Z \\ \text{s.t.} \quad & Z \succeq 0 \\ & E_{ii} \bullet Z \leq 1 \quad \forall i \\ & Z \in \mathbb{R}^{n\times n}, \end{aligned} \tag{3.2}$$

where here $Z \succeq 0$ means that Z is PSD (we'll soon simply write $Z \geq 0$ instead of $Z \succeq 0$ to mean that Z is PSD). Note a key consequence of the rewriting of (3.1) we have just performed: there is no longer an unbounded parameter d! The dimension of the matrix Z is fixed to n, which is a parameter of the problem.

Exercise 3. Let C_k be the cycle graph on k vertices. Find the smallest value of k for which the optimum of (3.1) (equivalently, of (3.2)), for G the adjacency matrix of C_k, is strictly larger than the size of the largest cut in C_k. How much larger is it? In a few lectures we will see that this example is not far from tight: the optimum of (3.1) can never be more than ~ 1.176 times larger than the size of the largest cut.

3.2 Semidefinite programs

In general a semidefinite program is the optimization of a linear function under linear and semidefinite constraints. Let's see some examples.

1. For any symmetric matrix B, its largest eigenvalue can be expressed as

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & x_1 \mathbb{I} - B \succeq 0 \end{aligned}$$

2. The following SDP

$$\begin{aligned} \inf \quad & x_1 \\ \text{s.t.} \quad & \begin{pmatrix} x_1 & 1 \\ 1 & x_2 \end{pmatrix} \succeq 0 \end{aligned}$$

is equivalent to $x_1, x_2 \geq 0$ and $x_1 x_2 \geq 1$. The optimum is 0, but this optimal value is not attained at any feasible point. This is an important difference with LPs. From now on we'll have to be careful and write "inf" or "sup" instead of "min" or "max" whenever we're writing an SDP for which we're not sure whether the optimum is attained.

3. This SDP

$$\begin{aligned} \inf \quad & x_n \\ \text{s.t.} \quad & x_0 \geq 2 \\ & \begin{pmatrix} 1 & x_0 \\ x_0 & x_1 \end{pmatrix} \succeq 0 \\ & \begin{pmatrix} 1 & x_1 \\ x_1 & x_2 \end{pmatrix} \succeq 0 \\ & \vdots \\ & \begin{pmatrix} 1 & x_{n-1} \\ x_{n-1} & x_n \end{pmatrix} \succeq 0 \end{aligned}$$

evaluates to 2^{2^n}. Here even writing down the optimum requires a number of bits (2^n) that is exponential in the instance size ($O(n)$ bits). This could not happen for LPs either.

The last example demonstrates that an SDP cannot always be solved exactly in polynomial time, even if it is both feasible and bounded. Two additional conditions will allow us to give polynomial-time algorithms. First, we will only solve SDPs approximately. This takes care of the second example: we will only require the solver to return a feasible point that achieves an objective value at least opt$-\varepsilon$, for any $\varepsilon > 0$ (the running time will depend on ε). Second, the SDP solver will require as input an a priori bound on the size of the solution. This gets rid of the third example. Finally, we will also need to require that the SDP is *strictly feasible*, meaning that there is a feasible point X that is strictly positive.

Under these three conditions it is possible to show that SDPs can be solved efficiently. An algorithm that works well is the ellipsoid algorithm — the same used to solve LPs. In a lecture or two we'll see another algorithm based on a matrix variant of the MW algorithm. For now let's state one of the best results known:

Theorem 3.2. *For any $\epsilon > 0$ and any SDP such that the feasible region K is such that $\exists r, R > 0$, $\vec{O}$ with*

$$B(\vec{O}, r) \subset K \subset B(\vec{O}, R),$$

a feasible X such that $B \bullet X \geq \text{OPT}(SDP) - \epsilon$ can be computed in time $\text{poly}(\log \frac{R}{r} + |SDP| + \log \frac{1}{\epsilon})$, where $|SDP|$ denotes the number of bits required to completely specify the SDP instance.

3.2.1 Dual of a SDP

Just as linear programs, every SDP has a *canonical form* as follows:

$$\begin{aligned} (\mathcal{P}) \qquad \sup \quad & B \bullet X \\ \text{s.t.} \quad & A_i \bullet X \leq c_i \quad \forall i \in \{1, \ldots, m\} \\ & X \succeq 0, \end{aligned} \tag{3.3}$$

where $B \in \mathbb{R}^{n \times n}$, $A_1, \ldots, A_m \in \mathbb{R}^{n \times n}$, $c_i \in \mathbb{R}$.

Exercise 4. Write each of the three SDPs from the previous section in canonical form (i.e. specify what the matrices B, A_i, and the reals c_i should be).

Let's develop the duality theory for SDPs. What is the dual of $(\mathcal{P})$ given in (3.3)? Let's proceed in the same way as one derives the dual of an LP: form linear combinations of the constraints in order to prove upper bounds on the objective value. More precisely, for any $y_1, \ldots, y_m \geq 0$, if

$$y_1 A_1 + \ldots + y_m A_m \succeq B$$

then for any primal feasible X

$$B \bullet X \preceq (y_1 A_1 + \ldots + y_m A_m) \bullet X = y^T c,$$

where the second inequality uses $A \leq Z \implies A \bullet X \leq Z \bullet X$ for any $X \geq 0$. We obtain the dual

$$\begin{aligned} (\mathcal{D}) \qquad \inf \quad & y^T c \\ \text{s.t.} \quad & y_1, \ldots, y_m \geq 0 \\ & y_1 A_1 + \ldots + y_m A_m - B \succeq 0, \end{aligned}$$

and we just showed:

Theorem 3.3 (Weak Duality)**.** *If both the primal and the dual problems are feasible and bounded, then*

$$\text{OPT}(\mathcal{P}) \leq \text{OPT}(\mathcal{D}).$$

While weak duality always holds under the same conditions as for LPs, strong duality can fail dramatically!

Example 3.4. Consider the optimization problem

$$\begin{array}{ll} \inf & -y_1 \\ \text{s.t.} & \begin{pmatrix} 0 & y_1 & 0 \\ y_1 & y_2 & 0 \\ 0 & 0 & 1-y_1 \end{pmatrix} \succeq 0 \qquad\qquad y_1, y_2 \geq 0. \end{array} \tag{3.4}$$

A block matrix is PSD if and only if each block is PSD. The determinant of a PSD matrix should be no less than 0, thus $0 \times y_2 - y_1^2 \geq 0$, and the optimum of the above SDP is 0. You can check that its dual is given by

$$\begin{array}{ll} \sup & -X_{33} \\ \text{s.t.} & X_{12} + X_{21} - X_{33} \leq -1 \\ & X_{22} \leq 0 \\ & X \succeq 0. \end{array}$$

Since $X_{22} \leq 0$, for X to be PSD it must be 0. The PSD condition then implies $X_{12} = X_{21} = 0$, so $-X_{33} \leq -1$ and the optimum is -1.

In spite of this strong duality does hold as long as both the primal and dual SDPs are *strictly feasible*:

Theorem 3.5 (Strong Duality)**.** *Suppose both the primal $\mathcal{P}$ and the dual $\mathcal{D}$ are strictly feasible and bounded, then*

$$\mathrm{OPT}(\mathcal{P}) = \mathrm{OPT}(\mathcal{D}).$$

Proof. Suppose for contradiction that $\alpha = \mathrm{OPT}(\mathcal{P}) < \mathrm{OPT}(\mathcal{D}) = \beta$. Let $\alpha' = \frac{\alpha+\beta}{2}$ be the mid-point. Define the set

$$K = \left\{ \begin{pmatrix} \sum_i y_i A_i - B \\ \alpha' - c^\top y \end{pmatrix} \in \mathbb{R}^{n^2+1}, y \in \mathbb{R}^m \right\}$$

Then K does not intersect the positive cone $\mathrm{Pos}(\mathbb{R}^n) \times \mathbb{R}_+$. By definition K is an affine subspace, hence there exists an affine hyperplane $\begin{pmatrix} Z \\ \mu \end{pmatrix} \in \mathbb{R}^{n^2+1}, \delta \in \mathbb{R}$ that includes K but does not intersect $\mathrm{Pos}(\mathbb{R}^n) \times \mathbb{R}_+$:

- $\forall y \in \mathbb{R}^m, Z \bullet (\sum y_i A_i - B) + \mu(\alpha' - c^\top y) = \delta$;
- $\forall Q \succeq 0, \forall q \geq 0, Z \bullet Q + \mu q \geq \delta$.

Taking $Q = 0, q = 0$ in the second constraint shows that necessarily $\delta \leq 0$. Suppose Z has a negative eigenvalue. For any value of δ, taking $q = 0$ and Q as a large enough multiple of the projection on the eigenvector of Z corresponding to this eigenvalue would contradict the second inequality. Thus $Z \succeq 0$.

Setting $y = 0$ in the first constraint gives $Z \bullet B = \alpha'\mu - \delta$. Setting $y = e_i$, we obtain $Z \bullet A_i = \mu c_i$. Finally, taking y to be any strictly feasible point, meaning $\begin{cases} \sum_i y_i A_i - B \succ 0 \\ \alpha' - c^\top y < 0 \end{cases}$ (since any feasible solution will satisfy $c^\top y \geq \beta > \alpha'$), gives $\mu > 0$. (To see this, note that if $Z = 0$ then necessarily $\mu \neq 0$ so we are done; if $Z \neq 0$ then $Z \bullet (\sum_i y_i A_i - B) > 0$ and we are done as well.)

Define $Z' = \frac{Z}{\mu}$. Then Z' is primal feasible: $Z' \succeq 0$ since both $Z \succeq 0$ and $\mu > 0$, and $Z' \bullet A_i = c_i$ for each i. We can also compute the objective value $Z' \bullet B = \alpha' - \frac{\delta}{\mu} > \alpha$ since $\delta < 0$, $\mu > 0$. This is a contradiction with our original assumption that the primal optimum was α. □

3.3 Quantum multiplayer games

In the previous lecture we have been studying the quantity

$$\mathrm{OPT}(A) = \max_{x_i, y_j \in \{\pm 1\}} \sum_{i,j} A_{ij} x_i y_j \tag{3.5}$$

as an optimization problem. In this lecture we're going to re-interpret (3.5) as the optimum of a two-player game. This is a useful perspective on optimization / constraint satisfaction problems in general. It will also let us make a connection with quantum multiplayer games.

Consider a game between a referee and two players, Alice and Bob. (Alice and Bob cooperate to win the game "against" the referee.)

(1) Select $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$ according to some distribution π.

(2) Send i to Alice and j to Bob. Alice and Bob reply with signs $x_i, y_j \in \{\pm 1\}$ respectively.

(3) The payoff is $x_i y_j c_{ij}$, where $c_{ij} \in \{\pm 1\}$.

The goal of the players is to provide answers whose product is a certain target c_{ij}. However, Alice only knows i and Bob only knows j, which is what can make the game challenging. Consider the following example:

Example 3.6. Let $m = n = 2$ and $c_{11} = c_{12} = c_{21} = 1$, $c_{22} = -1$. Then

$$w(G) = \max_{a_i, b_j \in \{\pm 1\}} \sum_{i,j} \pi(i,j) c_{ij} a_i b_j = \max \frac{1}{4}(a_1 b_1 + a_1 b_2 + a_2 b_1 - a_2 b_2) = \frac{1}{2}.$$

So the maximum payoff is $1/2$ – it is impossible to coordinate perfectly to always win in this game.

Given a game of the form described above, introduce a matrix $A \in \mathbb{R}^{n\times m}$ with coefficients $A_{i,j} = \pi(i,j)c_{ij}$. Then the maximum expected payoff for the players is

$$w(G) = \max_{x_i,y_j\in\{\pm 1\}} \sum_{i,j} A_{ij}x_iy_j = \mathrm{OPT}(A).$$

Conversely, for any $A \in \mathbb{R}^{m\times n}$, we can define $c_{ij} = \mathrm{sgn}(A_{ij})$ and $\pi(i,j) = \frac{A_{ij}}{\sum_{k,l}|A_{kl}|}$ to transform any optimization problem of the form (3.5) into a game. In particular, our results so far imply that finding the optimal strategy in such a game is NP-hard (as it implies that one should be able to solve MAXCUT), but can also be approximated within a constant factor in polynomial time. This surprising connection, between quadratic optimization and games, lies at the heart of many recent results in complexity theory (PCP theorem, anyone?).

Remark 3.7. In general one may allow the players to use randomized strategies, including both private and shared randomness, to select their answers. It is not hard to see that this cannot help in general: if on average (over their random coins) the players achieve a certain payoff, then there must exist some fixing of the random coins that lets them achieve at least the average payoff, and they might as well fix this choice of coins as part of their strategy.

Now let's consider *quantum* players. Compared to the classical setting, quantum players have a new resource known as *entanglement*. This means that they can be in a joint quantum state, $|\psi\rangle$, and by making measurements on their part of $|\psi\rangle$ they can get answers that are correlated in a way that could not happen classically.

In general, a measurement is an *observable*, which is a Hermitian matrix $X \in \mathbb{C}^{d\times d}$ such that $X = X^\dagger$, $X^2 = \mathbb{I}$ (in other words, all eigenvalues of X are $\in \{\pm 1\}$). (If you are not comfortable with complex matrices you can simply think of X as a real symmetric matrix.) Any measurement produces an outcome $a \in \{\pm 1\}$. The laws of quantum mechanics state that if Alice and Bob measure their own half of a certain state using X and Y, then the product of the outcomes on expectation satisfies

$$\mathbf{E}[a\cdot b] = \frac{1}{d}\mathrm{Tr}(X^\dagger Y) \in [-1,1].$$

If we use $d = 1$, the above reduces to the classical setting. Quantum mechanics allows us to explore higher dimensions. Let's consider the following example:

Example 3.8. If $X = Y = \mathbb{I}$, then the expectation is 1. If $X = \mathbb{I}, Y = -\mathbb{I}$, then the expectation is -1. If $X = \mathbb{I}$, $Y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, then the expectation is 0. Beyond these simple examples, we can get richer things. Consider for instance

$$X_1 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, Y_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, X_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ -1 & -1 \end{pmatrix}.$$

Then $\frac{1}{2}X_1\cdot Y_1 = \frac{1}{2}X_1\cdot Y_2 = \frac{1}{2}X_2\cdot Y_1 = \frac{\sqrt{2}}{2}$ and $\frac{1}{2}X_2\cdot Y_2 = -\frac{\sqrt{2}}{2}$. Such correlations are impossible for classical players! How do we see this? Plugging in this "strategy" in our

example game from earlier, we see that the expected payoff achieved by the quantum players is

$$\frac{1}{4} \cdot 4 \cdot \frac{\sqrt{2}}{2} = \frac{\sqrt{2}}{2} \approx 0.73,$$

which is strictly larger than the value $1/2$ that we proved was optimal for classical players.

From this example we see that quantum players strictly outperform their classical peers. How well can they do? The optimal expected payoff for quantum players is given by

$$w^*(G) = \sup_{\substack{X_i, Y_j \in \mathbb{C}^{d \times d} \\ X_i^2 = Y_j^2 = \mathbb{I} \\ X_i = X_i^\dagger \\ Y_j = Y_j^\dagger}} \sum_{i,j} A_{ij} \cdot \frac{1}{d} \mathrm{Tr}(X_i^\dagger Y_j) \leq \sup_{\substack{\vec{u}_i, \vec{v}_j \in \mathbb{C}^{d^2} \\ \|\vec{u}_i\| = \|\vec{v}_j\| = 1}} \sum_{i,j} A_{ij} \vec{u}_i \cdot \vec{v}_j = \mathrm{SDP}(A).$$

This inequality holds because we can set $\vec{u}_i = \frac{1}{\sqrt{d}} \mathrm{vec}(X_i)$ and $\vec{v}_j = \frac{1}{\sqrt{d}} \mathrm{vec}(Y_j)$, where $\mathrm{vec}(\cdot)$ returns the vector concatenating all the columns of the input matrix. (Note that this way we get complex vectors, but it is possible to split the real and imaginary parts (and double the dimension) to obtain real vectors.) Under such choice, one can verify that

$$\|\vec{u}_i\| = \frac{1}{d} X_i \bullet X_i = \frac{1}{d} \mathrm{Tr}(\mathbb{I}) = 1,$$

$$\|\vec{v}_i\| = \frac{1}{d} Y_j \bullet Y_j = \frac{1}{d} \mathrm{Tr}(\mathbb{I}) = 1,$$

$$u_i \cdot v_j = \frac{1}{d} X_i \bullet Y_j = \frac{1}{d} \mathrm{Tr}(X_i^\dagger Y_j).$$

Amazingly, the converse inequality is also true: from vectors one can define observables, hence the optimal value for the quantum players is *exactly* the optimum of the SDP. This is quite a surprising connection, and it has been very helpful in the study of quantum games and nonlocality. Some simple consequences are:

- The maximum expected payoff of quantum players can be computed efficiently (recall that for classical players it is NP-hard),
- The best quantum strategy can be found efficiently, as the transformation from vectors to observables is efficient,
- Quantum players can only achieve a payoff that is a constant factor larger than the best classical.

3.4 The Matrix Multiplicative Weights Algorithm

We will see how a large class of semidefinite programs can be solved very efficiently using an extension of the Multiplicative Weights Algorithm to the case of matrices. Recall the set-up of the algorithm:

- There are n experts.
- At each step $t = 1, \cdots, T$, the player chooses a distribution $p^{(t)}$ over experts.
- A cost vector $m^{(t)} \in [-1, 1]^n$ is supplied by the environment.
- The player suffers a loss of $p^{(t)} \bullet m^{(t)}$.

Now suppose there is a continuous set of experts, each associated with a unit vector $v \in \mathbb{R}^n, \|v\| = 1$. Let the distribution over experts be a distribution $\mathcal{D}$ on the set of all unit vectors, the unit sphere $\mathcal{S}^{n-1}$. Additionally let the loss be specified by a symmetric matrix $M \in \mathbb{R}^{n\times n}, 0 \preceq M \preceq \mathbb{I}$ (all eigenvalues of M are between 0 and 1) such that by definition the loss associated with expert v is given by $v^\top M v$. Our assumption on M implies this loss will always fall in the interval $[0, 1]$. The expected loss under distribution $\mathcal{D}$ is given by

$$\begin{aligned}\mathbb{E}_{v\sim\mathcal{D}}[v^\top M v] &= \mathbb{E}_{v\sim\mathcal{D}}[M \bullet vv^\top] \\ &= M \bullet \underbrace{\mathbb{E}_{v\sim\mathcal{D}}[vv^\top]}_{\rho\succeq 0},\end{aligned}$$

where $\rho \succeq 0$ satisfies $\mathrm{Tr}(\rho) = \mathbb{E}_{v\sim\mathcal{D}}\|v\|^2 = 1$. ρ is called a *density matrix*. (As a notation reminder, $A \bullet B = \mathrm{Tr}(A^\top B)$.)

We then have the following extension of MWA:

Definition 3.9. (**MMWA**) Let $\eta > 0$. Initialize $W^{(1)} = \mathbb{I}$. For $t = 1, \cdots, T$:

- Observe a loss matrix $0 \preceq M^{(t)} \preceq \mathbb{I}$.
- Set $X^{(t)} = \frac{W^{(t)}}{\mathrm{Tr}(W^{(t)})}$. Suffer a loss $X^{(t)} \bullet M^{(t)}$.
- Update $W^{(t+1)} = \exp(-\eta(M^{(1)} + \cdots + M^{(t)}))$.

Remark 3.10. Note that the normalization condition $0 \preceq M^{(t)} \preceq \mathbb{I}$ and $X^{(t)} \bullet \mathbb{I} = \mathrm{Tr}(X^{(t)}) = 1$ implies that the losses are always in $[0, 1]$.

Remark 3.11. The matrix algorithm is a strict generalization of MWA. We can recover the experts framework by considering the special case where all loss matrices are diagonal:

$$M^{(t)} = \begin{pmatrix} m_1^{(t)} & & & \\ & \ddots & 0 & \\ & 0 & \ddots & \\ & & & m_n^{(t)} \end{pmatrix}.$$

Indeed in this case all the $X^{(t)}$ matrices are diagonal as well, and their diagonal coefficients represent a probability distribution over the experts.

Remark 3.12. The algorithm calls for taking the exponential of a symmetric matrix. There are many equivalent ways in which this can be defined. The simplest is to diagonalize A as $A = UDU^\top$, where U is orthogonal and D is diagonal, and define $\exp(A) = U\exp(D)U^\top =$

$$U \begin{pmatrix} e^{D_{11}} & & & \\ & \ddots & 0 & \\ & 0 & \ddots & \\ & & & e^{D_{nn}} \end{pmatrix} U^\top.$$

More generally, for any function f we can define $f(A)$ in this way. Taking $f(x) = x^k$ for some integer k, you can verify that this definition agrees with the usual definition of the matrix product. For instance, $A^2 = (UDU^T)(UDU^T) = UD^2U^T$ since U is orthogonal.

In practice, to avoid explicit diagonalization one can use the Taylor expansion of the exponential: $\exp(A) = \mathbb{I} + A + \frac{A^2}{2!} + \cdots + \frac{A^k}{k!} + \cdots$, which converges exponentially fast. Since the exponential needs to be computed at every step of the algorithm it is important to implement it as efficiently as possible.

Exercise 5. Show that if A and B are symmetric matrices that commute ($AB = BA$) then $\exp(A + B) = \exp(A)\exp(B)$. Find two real 2×2 symmetric matrices A and B such that $\exp(A + B) \neq \exp(A)\exp(B)$.

The exercise shows that the definition of $W^{(t)}$ given in the matrix algorithm is *not* equivalent in general to the update rule $W^{(t)} := \exp(-\eta M^{(t)})W^{(t-1)}$ that we saw in the experts framework; the two rules are identical only if all matrices $M^{(t)}$ commute. In general, we only know the inequality $\mathrm{Tr}(e^{A+B}) \leq \mathrm{Tr}(e^A e^B)$, which is known as the Golden-Thompson inequality and will be used in the analysis of the convergence of the MMWA.

The following theorem is a direct extension of the experts theorem:

Theorem 3.13. *For any $M^{(1)}, \ldots, M^{(T)} \in \mathbb{R}^{n\times n}$ such that $0 \preceq M^{(t)} \preceq \mathbb{I}$ for all $t = 1, \ldots, T$, and $0 < \epsilon \leq 1/2$, let $\eta = -\ln(1 - \epsilon)$. Then the following relation holds for $M^{(t)}$ and $X^{(t)}$ defined as in the Matrix Multiplicative Weights Algorithm with parameter η:*

$$\sum_{t=1}^{T} M^{(t)} \bullet X^{(t)} \leq (1 + \epsilon) \inf_{\rho \in \mathrm{Pos}(\mathbb{R}^n),\, \mathrm{Tr}(\rho)=1} \Big(\sum_{t=1}^{T} M^{(t)} \bullet \rho \Big) + \frac{\ln n}{\epsilon}. \tag{3.6}$$

Proof. You will see the proof in your homework. □

3.5 Solving SDPs using the MMWA

We will see how the MMWA can be used to approximately solve an arbitrary semidefinite program. We first reduce the optimization problem to a feasibility problem. Suppose we are given an SDP in standard form:

$$
\begin{aligned}
(\mathcal{P}): \quad \sup \quad & B \bullet X \\
\text{s.t.} \quad & A_i \bullet X \leq c_i \qquad \forall i \in \{1, \ldots, m\} \\
& X \succeq 0
\end{aligned}
$$

$$
\begin{aligned}
(\mathcal{D}): \quad \inf \quad & c^\top y \\
\text{s.t.} \quad & \sum_i y_i A_i - B \succeq 0 \\
& y_i \geq 0 \qquad \forall i \in \{1, \ldots, m\}
\end{aligned}
$$

Assume $A_1 = \mathbb{I}, c_1 = R$, which gives the constraint that $\mathrm{Tr}(X) \leq R$, effectively imposing an a priori bound on the size of X. Assume also that the optimal α of the primal problem is non-negative (if not, we can always shift the optimum by adding a positive multiple of the identity to B to make it PSD). To perform a reduction from deciding optimality to deciding feasibility we perform a binary search over $\alpha \in [0, \|B\|R]$ (where $\|B\|$ is the largest eigenvalue of B, so $|B \bullet X| \leq \|B\|\mathrm{Tr}(X) \leq \|B\|R$), at each step deciding feasibility of the following problem:

$$
\begin{aligned}
\exists ? \, X \quad \text{s.t.} \quad & B \bullet X > \alpha \\
& A_i \bullet X \leq c_i \\
& X \succeq 0.
\end{aligned} \tag{3.7}
$$

We will use the MMWA presented in the previous section to answer this decision problem. The idea is to start with a "random" guess of X, such as $X = R\mathbb{I}/n$. Then we iteratively "improve" X to either turn it into a feasible solution, or somehow obtain a proof that the problem is *not* feasible, in which case we know the objective value of the original SDP is smaller than α. In the latter case we repeat the search between 0 and α; in the former we search between α and $\|B\|R$. The number of iterations required will be logarithmic in $\|B\|R/\delta$, where δ is the precision we'd like to achieve.

The following is the main claim that underlies the "improvement" subroutine for X that we will present afterwards:

Claim 3.14. *Let $X \in \mathbb{R}^{n \times n}$ be PSD. The following are equivalent:*

(i) $\exists y \in \mathbb{R}^m_+$ such that $c^\top y \leq \alpha$ and $X \bullet (\sum_i y_i A_i - B) \geq 0$;

(ii) For any scaling factor $\lambda \in \mathbb{R}$, either (λX) is infeasible or $B \bullet (\lambda X) \leq \alpha$.

Note that the claim is saying that, if we are not happy (the current X is either infeasible, or it is feasible but does not satisfy the condition on the objective value), then there is a "reason" for this: there is a positive linear combination of the constraints, specified by the vector y, which achieves a dual objective value less than α and is such that the associated dual constraint is nonnegative "in the direction of X". We will use this y as an indication as to how X can be "improved".

Proof. Assume (i) and suppose λX is feasible, for some scaling $\lambda \in \mathbb{R}$; we need to show that $B \bullet (\lambda X) \leq \alpha$. From the assumption in (i) (which is scale-invariant), we have that

$$\begin{aligned} 0 &\leq \sum_i y_i A_i \bullet (\lambda X) - (\lambda X) \bullet B \\ &\leq y^\top c - (\lambda X) \bullet B \\ &\leq \alpha - (\lambda X) \bullet B, \end{aligned}$$

where the second inequality uses the assumption that (λX) is feasible and the third the assumption on y. Thus $(\lambda X) \bullet B \leq \alpha$, as desired.

We prove the converse implication by contrapositive. Assume (i) does not hold; let's show that (ii) does not hold either. Consider the following pair of primal and dual *linear* programs, that depend on X:

$$\begin{aligned} (P_X): \quad \max \quad & \sum_i (A_i \bullet X) y_i \\ \text{s.t.} \quad & c^T y \leq \alpha \\ & y_i \geq 0 \qquad \forall i \end{aligned}$$

$$\begin{aligned} (D_X): \quad \min \quad & \alpha z \\ \text{s.t.} \quad & c_i z \geq A_i \bullet X \qquad \forall i \\ & z \geq 0 \end{aligned}$$

The negation of (i) implies that the optimum of the primal is less than $B \bullet X$. In particular, the primal is bounded, so strong duality (for linear programs!) holds, and the optimum to the dual equals the optimum to the primal. Let z^* be the optimum solution to the dual. Then $\alpha z^* < B \bullet X$. Let $X^* = \frac{1}{z} X$. Then by the dual constraints X^* is feasible, and by construction $B \bullet X^* > \alpha$. This shows that (ii) does not hold, as desired. □

The equivalence stated in the claim shows that the existence of a y such that condition (i) is satisfied is a proof that we have not yet reached our goal of finding a good feasible solution. The idea is that solving (i) is much easier than solving the full SDP. In particular y does not need to be dual feasible and in fact, (i) is an LP feasibility problem.

We will show how to solve the SDP feasibility problem under the following assumption:

Assumption: There exists an oracle $\mathcal{O}$ such that given a PSD matrix $X \in \mathbb{R}^{n \times n}$ as input, $\mathcal{O}$ returns either:

(a) A vector y such that (i) in Claim 3.14 above holds, or

(b) A statement that no such y exists.

The "quality" of $\mathcal{O}$ is measured through its "width" σ, which is the largest possible value that $\|\sum_i y_i A_i - B\|$ can take over all vectors y that the oracle might return. When designing an oracle, we want it to be fast and have small width. We will see how to do this in some

special cases (such as the SDP for MAXCUT we saw in the previous lecture). Assuming the oracle is given, consider the following algorithm:

Algorithm for solving SDP: Let $X^{(1)} = \frac{R\mathbb{I}}{n}, \varepsilon = \frac{\delta\alpha}{2\sigma R}, \eta = -\ln(1-\varepsilon)$, where $\delta > 0$ is an accuracy parameter. For $t = 1, \cdots, T$, do:

(1) Execute $\mathcal{O}$ on $X^{(t)}$.

- If case (b) happens, then it follows that (i) does not hold, and by Claim 3.14 (ii) does not hold either, and $X^{(t)}$ is feasible such that $B \bullet X^{(t)} > \alpha$: we are done. Stop and return $X^{(t)}$.

- If instead case (a) happens, let $y^{(t)}$ be the vector returned by $\mathcal{O}$. Define a loss matrix
$$M^{(t)} = \frac{\sum_i y_i^{(t)} A_i - B + \sigma\mathbb{I}}{2\sigma},$$
so that the assumption on the width of $\mathcal{O}$ implies $0 \leq M^{(t)} \leq \mathbb{I}$.

(2) Update $X^{(t+1)}$ as in MMWA:
$$W^{(t+1)} = e^{-\eta\sum_{i=1}^{t} M^{(i)}}, \qquad X^{(t+1)} = R\frac{W^{(t+1)}}{\mathrm{Tr}(W^{(t+1)})}.$$

The following theorem states the guarantee for this algorithm.

Theorem 3.15. *Assume case (b) does not happen for $T = \frac{8\sigma^2R^2}{\delta^2\alpha^2}\ln(n)$ steps. Then $\bar{y} = \frac{\delta\alpha}{R}e_1 + \frac{1}{T}\sum_{t=1}^{T} y^{(t)}$ is a feasible solution to $(\mathcal{D})$ with objective value less than or equal to $(1+\delta)\alpha$.*

We can make the following remarks on the running time of this algorithm:

- The overall performance depends on the running time of the oracle, which needs to be executed once per iteration.

- We need to compute a matrix exponential to update X and it is important to do this efficiently. In general, diagonalizing the loss matrices will take $O(n^3)$ time. Often it is possible to do much better by observing that the only information the oracle requires is the inner product of $X^{(t)}$ with the constraint matrices, but not the full matrix $X^{(t)}$ itself. So it is sometimes possible to do the update much more efficiently by keeping only a "low-dimensional sketch" of the matrix exponential. This can be done using the Johnson-Lindenstrauss lemma for dimensionality reduction.

- We would like to have R and the width of the oracle to be not too large, otherwise the algorithm will require lots of iterations. We will see how to do this for the special case of the MAXCUT SDP in the next lecture.

- The algorithm depends inverse polynomially on the approximation error $\delta\alpha$. This is not great, and in general we could hope for an algorithm that depends only polylogarithmically on δ^{-1}. For example this is the case for the ellipsoid algorithm.

- If all these parameters, $R, \sigma, \delta\alpha$ are constants, or such that $\sigma R/(\delta\alpha)$ is not too large, then the overhead $\ln(n)$ in the number of iterations for the algorithm is very small. This is the main strength of the MMWA, and we will see how to take advantage of it for the case of MAXCUT.

Proof. First let's check the claim on the objective value:

$$\begin{aligned} c^T y &= \frac{\delta\alpha}{R} c^T e_1 + \frac{1}{T}\sum_i c^T y^{(t)} \\ &\leq \delta\alpha + \alpha = (1+\delta)\alpha, \end{aligned}$$

where we used $c_1 = R$ and the guarantee $c^T y^{(t)} \leq \alpha$ for any $y^{(t)}$ returned by the oracle.

Next we need to verify that the returned solution is feasible. From condition (a) we know that $y \geq 0$, so it remains to check that $\sum_i y_i A_i - B \geq 0$. From the MMWA theorem we know that for any unit vector v, and in particular the eigenvector of $\sum_t M^{(t)}$ associated with its smallest eigenvalue λ_n,

$$\begin{aligned} \frac{T}{2} \leq& \sum_{t=1}^{T} M^{(t)} \bullet X^{(t)} \\ \leq& (1+\epsilon)\sum_{t=1}^{T} v^T M^{(t)} v + \frac{\ln n}{\epsilon} \\ =& (1+\epsilon)\lambda_n\Big(\sum_{t=1}^{T} \frac{\sum_i y_i^{(t)} A_i - B + \sigma I}{2\sigma}\Big) + \frac{\ln n}{\epsilon} \\ =& \frac{1+\epsilon}{2\sigma}\lambda_n\Big(\sum_i \Big(\sum_t y_i^{(t)}\Big) A_i - TB\Big) + (1+\varepsilon)\frac{T\sigma}{2\sigma} + \frac{\ln n}{\epsilon}. \end{aligned}$$

Here the first line holds by definition of $M^{(t)}$ and using guarantee (i) for the oracle, the third line holds by our choice of v, and the fourth uses that for any PSD A, $\lambda_i((A+b\mathbb{I})/c) = (\lambda_i(A)+b)/c$ for any b and any $c > 0$ (where λ_i is the i-th largest eigenvalue). Rearranging terms,

$$\begin{aligned} \Big(-\frac{\varepsilon T}{2} - \frac{\ln n}{\epsilon}\Big)\frac{2\sigma}{(1+\epsilon)T} &\leq \lambda_n\Big(\sum_i \Big(\frac{1}{T}\sum_t y_i^{(t)}\Big) A_i - B\Big) \\ &= \lambda_n\Big(\sum_i \bar{y}_i A_i - B\Big) - \frac{\delta\alpha}{R}, \end{aligned}$$

where the equality follows from the definition of $\bar{y}$. Using that by definition of T and ε we have $\frac{\varepsilon T}{2} = \frac{\ln n}{\varepsilon}$, it follows that

$$\frac{-4\sigma \ln n}{\epsilon(1+\epsilon)T} \leq \lambda_n\Big(\sum_i \bar{y}_i A_i - B\Big) - \frac{\delta\alpha}{R}.$$

Given the choice of parameters made in the theorem you can check that $\frac{\delta\alpha}{R} - \frac{4\sigma \ln n}{\epsilon(1+\epsilon)T} > 0$, so the smallest eigenvalue of $\sum_i \bar{y}_i A_i - B$ is positive, meaning this is a PSD matrix and $\bar{y}$ is feasible, as required. □

3.6 Application: solving the MAXCUT SDP using MMWA

In this lecture we apply the MMW-based algorithm introduced in the previous lecture to obtain a quasilinear time algorithm for solving the SDP relaxation of the MAXCUT problem introduced a couple lectures ago.

Recall that we saw that for a given undirected graph $G = (V, E)$, the size of the largest cut could be written as

$$\text{MAXCUT}(G) = \frac{|E|}{2} + \max_{x_i \in \{\pm 1\}} \frac{1}{2} \sum_{\{i,j\} \in E} -x_i x_j \leq \frac{|E|}{2} + \sup_{\substack{u_i \in \mathbb{R}^n \\ \|u_i\|=1}} \frac{1}{2} \sum_{i,j} -A_{i,j} u_i . u_j = \text{SDP}(G),$$

where A is the (symmetrized) adjacency matrix of the graph G, which has $1/2$ for every entry (i, j) and (j, i) associated to an edge $\{i, j\}$ and zeros elsewhere. For simplicity let's assume G is d-regular (i.e. each vertex has degree exactly d). Then the adjacency matrix A has at most d non-zero entries, each equal to $1/2$, in every row, so $\|A\| \leq d/2$.

The SDP can be written in canonical primal form as

$$\begin{aligned} \text{SDP}(G) = \sup \quad & B \bullet X \\ s.t. \quad & E_{ii} \bullet X \leq 1 \quad \forall i = 1, \ldots, n \\ & X \succcurlyeq 0, \end{aligned}$$

where $B = \frac{d}{4}\mathbb{I} - \frac{A}{2}$ and E_{ii} is a matrix whose i^{th} diagonal entry is 1 and the others are 0. Given our bound $\|A\| \leq \frac{d}{2}$, we have $0 \leq B \leq \frac{d}{2}\mathbb{I}$.

Note that if α is the optimal value for the SDP we have $\frac{|E|}{2} = \frac{nd}{4} \leq \alpha \leq |E| = \frac{nd}{2}$. The first inequality follows since there is always a cut of size $|E|/2$ (a random cut will cut half the edges), and the second follows from the bound on the norm of B.

Now our goal is to design an oracle $\mathcal{O}$ that we can use for the algorithm that we introduced to solve SDP using MMW. In other words, given $X \succcurlyeq 0$ such that $\text{Tr}(X) = n$ (n plays the role of R in our algorithm), find $y \in \mathbb{R}^n_+$ such that $c^T y = \sum_i y_i \leq \alpha$ and $X \bullet (\sum_i y_i E_{ii} - B) \geq 0$ (if such a y exists).

We design the oracle by distinguishing the following cases:

(i) If $B \bullet X \leq \alpha$, let $y_i = \frac{\alpha}{n} \geq 0$ for all i. Then $c^T y = \sum_i y_i = n\frac{\alpha}{n} = \alpha$. Moreover,

$$X \bullet (\sum_i y_i E_{ii} - B) = \sum_i y_i X_{ii} - X \bullet B = \frac{\alpha}{n}\mathrm{Tr}(X) - X \bullet B = \alpha - X \bullet B \geq 0,$$

and we are done.

(ii) Suppose $B \bullet X = \lambda\alpha > \alpha$ ($\lambda > 1$). We also have $\lambda \leq 2$ because

$$B \bullet X \;\leq\; \|B\|\mathrm{Tr}(X) \;\leq\; \frac{d}{2}n \;\leq\; 2\alpha\;.$$

Since $B \bullet X > \alpha$, if X is feasible then we are in case (b) for the oracle and we are done: we found a good feasible solution. Otherwise define

$$S = \{i : X_{ii} > \lambda\}, \qquad K = \sum_{i \in S} X_{ii}\;.$$

S is the set of indices corresponding to constraints $E_{ii} \bullet X \leq 1$ that are violated by a large amount (since $\lambda > 1$), and K is the sum of violating diagonal entries of X. Note that by definition, $|S| \leq K/\lambda$. Let

$$y_i = \begin{cases} \frac{\lambda\alpha}{K} & \text{if } i \in S\;, \\ 0 & \text{if } i \notin S\;. \end{cases}$$

Then obviously $y \geq 0$ and

$$c^T y \;=\; \sum_i y_i \;=\; \frac{\lambda\alpha}{K}|S| \;\leq\; \frac{\lambda\alpha}{K}\frac{K}{\lambda} \;=\; \alpha\;.$$

Besides,

$$X \bullet \Big(\sum_i y_i E_{ii} - B\Big) = \sum_{i \in S} \frac{\lambda\alpha}{K} X_{ii} - X \bullet B = \frac{\lambda\alpha}{K}K - X \bullet B = \lambda\alpha - X \bullet B = 0\;.$$

This completes a description of the oracle. How good is it? First note that it runs very fast. We have only two cases to distinguish between, and in each one we check a linear constraint. Thus the running time is linear in the number of edges m (to compute $B \bullet X$) and vertices n (to set all the values of y) of the graph.

Next we need to bound the width of the oracle.

$$\Big\|\sum_i y_i E_{ii} - B\Big\| \leq \left\| \begin{pmatrix} y_1 & & 0 \\ & \ddots & \\ 0 & & y_n \end{pmatrix} \right\| + \|B\| \leq \max_i |y_i| + \frac{d}{2}.$$

In order to find $\max_i |y_i|$ we should check all of the cases. In case (i) it is $\frac{\alpha}{n} \leq \frac{d}{2}$. In case (ii), for each i

$$|y_i| \;\leq\; \frac{\lambda\alpha}{K} \;\leq\; \frac{2}{K}\frac{nd}{2}\;.$$

So in this case it all depends on the size of K. If $K = \Omega(\delta n)$ then we obtain $\sigma = O(d/\delta)$ in both cases. Recall that the MMWA algorithm for solving SDPs required $T = \frac{8\sigma^2 R^2}{\delta^2 \alpha^2} \ln(n)$ iterations. At each iteration there is one call to the oracle made. Using $R/\alpha \leq 4/d$, our running time is then $O(mT) = O(m\delta^{-4} \log n)$, which is a quasi-linear-time algorithm! This is much better than a "generic" SDP solver, which would, at best, scale like $O(n^3)$.

But what if K is much smaller? In this case there is a relatively small number of violating coordinates: $|S| = O(\delta n)$ as well. The following claim shows that we can erase those coordinates, and scale the others, to obtain a feasible solution with objective value "almost α". So, in case the oracle observes such a small K it will simply abort and return "almost feasible".

Claim 3.16. *In case (ii) of the description of the oracle, assume that $K \leq \frac{\delta \lambda n}{4}$. Assume without loss of generality (permuting the rows and columns if necessary) that the first $|S|$ diagonal entries of X correspond to those $i \in S$, and define*

$$\bar{X} = \begin{pmatrix} 0 & 0 \\ 0 & \frac{1}{\lambda} X_{\bar{S},\bar{S}} \end{pmatrix}, \qquad \text{where we wrote} \qquad X = \begin{pmatrix} X_{S,S} & X_{S,\bar{S}} \\ X_{\bar{S},S} & X_{\bar{S},\bar{S}} \end{pmatrix}.$$

Then $\bar{X}$ is primal feasible with objective value $B \bullet \bar{X} \geq (1 - 3\delta/2)\alpha$.

Proof. A diagonal block extracted from a PSD matrix is PSD so $\bar{X} \succcurlyeq 0$. Moreover by definition $\bar{X}_{ii} \leq 1$ for every i, thus $\bar{X}$ is primal feasible. It remains to evaluate its objective value. Using the definition of λ,

$$\begin{aligned} \alpha - B \bullet \bar{X} &= B \bullet \left(\frac{1}{\lambda} X - \bar{X} \right) \\ &= \frac{1}{\lambda} \left(B_{S,S} \bullet X_{S,S} + B_{S,\bar{S}} \bullet X_{S,\bar{S}} + B_{\bar{S},S} \bullet X_{\bar{S},S} \right). \end{aligned} \tag{3.8}$$

Let $\{u_i\}$ be vectors from a Cholesky factoriation of X, so $X_{ij} = \langle u_i, u_j \rangle$. Using the definition of the matrix B, we have

$$\begin{aligned} \left| B_{S,\bar{S}} \bullet X_{S,\bar{S}} \right| &= \frac{1}{4} \left| \sum_{(i,j) \in (S \cap \bar{S}) \cap E} \langle u_i, u_j \rangle \right| \\ &\leq \frac{d}{4} \sqrt{\lambda} \sum_{i \in S} \|u_i\| \\ &\leq \frac{d}{4} \sqrt{\lambda |S|} \sqrt{\sum_{i \in S} \|u_i\|^2} \\ &\leq \frac{d}{4} K . \end{aligned}$$

Here for the first inequality we used that the degree is at most d, the Cauchy-Schwarz inequality, and $\|u_j\| \leq \lambda$ for $j \in \bar{S}$; for the second inequality we used Cauchy-Schwarz; for the last we used $\sum_{i \in S} \|u_i\|^2 = K$ and $|S| \leq K/\lambda$.

Using similar bounds on the other two terms in (3.8), the assumption $K \leq \frac{\delta \lambda n}{4}$, and $\frac{nd}{4} \leq \alpha$, we finally get $B \bullet \bar{X} \geq (1 - 3\delta/2)\alpha$. □

As a final comment, note that we cheated a little bit in our evaluation of the overall complexity of our algorithm for the MAXCUT SDP — at each iteration, we need not only call the oracle, but also perform the update for $X^{(t)}$, and this requires computing a matrix exponential... which can take time $O(n^3)$! Note however that here $X^{(t)}$ itself is never needed: the only information the oracle requires is the dot product $X^{(t)} \bullet B$, as well as the diagonal entries of X. It is possible to compute those very efficiently, in time $O(m \log n)$, using dimension reduction techniques such as the Johnson-Lindenstrauss lemma that we saw in a previous lecture.

3.7 Rounding semidefinite programs

3.7.1 General quadratic programs

Consider the following problem

$$\mathrm{MAXQP}(A) = \sup_{\substack{x_i, y_j \in \{\pm 1\} \\ i=1,\dots,n \\ j=1,\dots,m}} \sum_{i,j} A_{i,j} x_i y_j,$$

where $A \in \mathbb{R}^{n \times m}$.

Exercise 6. Show that this problem is NP-hard by performing a reduction from MAXCUT.

We will see how a good approximation can be obtained in polynomial time. For this we propose the following relaxation:

$$\mathrm{MAXQP}(G) \leq \mathrm{SDP}(G) = \sup_{\substack{u_i, v_j \in \mathbb{R}^{m+n} \\ \|u_i\| = \|v_j\| = 1}} \sum_{i,j} A_{i,j} u_i \cdot v_j. \tag{3.9}$$

Since the u_i and v_j are different sets of vectors it may not be immediately obvious that this program is an SDP, but it is. To check this, let's make sure we can write $\mathrm{SDP}(A)$ in the primal canonical form

$$\begin{aligned} \sup \quad & B \bullet Z \\ s.t. \quad & A_i \bullet Z \leq c_i \\ & Z \succcurlyeq 0. \end{aligned}$$

Let U be a matrix whose columns are the u_i, and V whose columns are the v_j; define

$$Z = (UV)^T (UV) = \begin{pmatrix} [u_i \cdot u_j] & [u_i \cdot v_j] \\ [v_i \cdot u_j] & [v_i \cdot v_j] \end{pmatrix} \in \mathbb{R}^{(m+n) \times (m+n)}.$$

Clearly Z is a PSD matrix (it is a Gram matrix), and its diagonal elements are the squared norms $\|u_i\|^2$ and $\|v_j\|^2$, which should be at most one. Thus we let $c_i = 1$ and $A_i = E_{ii}$ for

$i = 1, \ldots, n+m$, where E_{ii} is a $(n+m) \times (n+m)$ matrix whose i^{th} diagonal entry is 1 and the others are 0. Finally for the objective value, we define

$$B = \frac{1}{2} \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}.$$

Then the corresponding SDP is equivalent to the definition of SDP(A) given in (3.9).

3.7.2 Analysis of the SDP relaxation

In this section we analyze the performance of the relaxation (3.9). The main result is that we can bound the ratio SDP(A)/MAXQP(A) by a constant factor, regardless of the choice of A.

Theorem 3.17. *There exists a universal constant K such that $\forall A$, $\mathrm{SDP}(A) \leq K \cdot \mathrm{MAXQP}(A)$. Moreover, given unit vectors u_i and v_j achieving the optimum in SDP(A), there exists a deterministic polynomial-time algorithm that produces x_i and y_j in $\{\pm 1\}$ such that*

$$\sum_{i,j} A_{i,j} x_i y_j \;\geq\; K \cdot \mathrm{MAXQP}(A)\,.^{1}$$

Remark 3.18. Different proof techniques for the theorem yield different values of K. In your homework you will develop an algorithm to achieve $K \approx 0.56$. The best value for K is called Grothendieck's constant K_G and can be defined as

$$K_G = \inf \left\{ C : \forall m, n, \forall A \in \mathbb{R}^{n \times m}, \quad \sup \sum_{i,j} A_{i,j} u_i \cdot v_j \leq K_G \sup \sum_{i,j} A_{i,j} x_i y_j \right\}$$

Let's prove Theorem 3.17. Let $u_i, v_j \in \mathbb{R}^d$ achieving SDP(A) be given (note that technically we should assume they achieve SDP(A)$-\varepsilon$, as we don't know if the supremum is attained; for clarity we'll put this issue aside). We can always assume $d \leq m+n$ since this is the number of vectors. Observe that

$$\sum_{i,j} A_{ij} u_i \cdot v_j = \frac{1}{d} \sum_{k=1}^{d} \left(\sum_{i,j} A_{ij} \left(\sqrt{d} (u_i)_k \right) \left(\sqrt{d} (v_j)_k \right) \right).$$

This implies we can find $k \in \{1, \ldots, d\}$ such that

$$\sum_{i,j} A_{ij} \left(\sqrt{d} (u_i)_k \right) \left(\sqrt{d} (v_j)_k \right) \geq \sum_{i,j} A_{ij} u_i \cdot v_j = \mathrm{SDP}(A) \geq \mathrm{OPT}(A).$$

How large are the $|(u_i)_k|$? From $\|u_i\| \leq 1$ we know $|(u_i)_k| \leq 1$, which implies $\sqrt{d}(u_i)_k \leq \sqrt{d}$. This naive bound is not good enough: we are looking for an assignment of values in the

[1] In fact, we'll show a bit more, we'll prove that $\sum_{i,j} A_{i,j} x_i y_j \geq K \cdot \mathrm{SDP}(A)$, which of course is at least $C \cdot \mathrm{MAXQP}(A)$.

range $[-1, 1]$, so we'd have to divide all coordinates by $\sqrt{d}$, but then we'd lose a factor d in the objective value.

Now, if everything was "well-behaved", i.e. the vectors are random, we would expect $|(u_i)_k| \simeq 1/\sqrt{d}$ because $\|u_i\| \leq 1$ — in this case barely any renormalization is needed. This heuristic suggests the approach for the proof, which is to identify a rotation (in fact, an embedding in a space of slightly larger dimension) that guarantees the coordinates are well-balanced, so that most of them are not too large, no larger than some constant. We'll then "truncate" the coordinates that are too large, and argue that the loss in objective value suffered through this truncation is not too large.

3.7.3 Randomized rounding

Before giving a deterministic algorithm, as is claimed in the theorem, let's consider the following simple randomized rounding procedure. Let $\vec{g}$ be uniformly distributed over $\{\pm\frac{1}{\sqrt{d}}\}^d$. For any $\vec{u} \in \mathbb{R}^d$, define $h(\vec{u}) = \vec{g} \cdot \vec{u} \in \mathbb{R}$. For $M > 0$ define

$$h^M(\vec{u}) = \begin{cases} h(\vec{u}) & \text{if } |h(\vec{u})| \leq M, \\ M & \text{if } h(\vec{u}) > M, \\ -M & \text{if } h(\vec{u}) < -M. \end{cases}$$

Then we have the following lemma:

Lemma 3.19. *For any $\vec{u}, \vec{v} \in \mathbb{R}^d$, with $\|\vec{u}\| = \|\vec{v}\| = 1$,*

(1) $\mathbf{E}_{\vec{g}}[h(\vec{u}) \cdot h(\vec{v})] = \vec{u} \cdot \vec{v}$.

(2) $\mathbf{E}_{\vec{g}} |h(\vec{u})|^2 = 1$.

(3) $\mathbf{E}_{\vec{g}} |h^M(\vec{u})|^2 \leq 1$.

(4) $\mathbf{E}_{\vec{g}} |h(\vec{u}) - h^M(\vec{u})|^2 \leq \frac{3}{M^2}$.

We'll prove the lemma later, first let's use it to finish the proof of the theorem. The last property gives a precise trade-off between the size of M and the quality of the approximation

of the projection $h(\vec{u})$ by its truncation $h^M(\vec{u})$. Using the lemma, we can write

$$\begin{aligned}
&\mathrm{SDP}(A)\\
&= \sum_{i,j} A_{ij} \vec{u}_i \cdot \vec{v}_j\\
&= \mathop{\mathbf{E}}_{\vec{g}} \Big[\sum_{i,j} A_{ij} h(\vec{u}_i) \cdot h(\vec{v}_j) \Big]\\
&= \mathop{\mathbf{E}}_{\vec{g}} \Big[\sum_{i,j} A_{ij} h^M(\vec{u}_i) \cdot h^M(\vec{v}_j) \Big] + \mathop{\mathbf{E}}_{\vec{g}} \Big[\sum_{i,j} A_{ij} h^M(\vec{u}_i) \cdot (h(\vec{v}_j) - h^M(\vec{v}_j)) \Big]\\
&\qquad + \mathop{\mathbf{E}}_{\vec{g}} \Big[\sum_{i,j} A_{ij} (h(\vec{u}_i) - h^M(\vec{u}_i)) h^M(\vec{v}_j) \Big]\\
&\le M^2 \mathop{\mathbf{E}}_{\vec{g}} \Big| \sum_{i,j} A_{ij} \Big(\frac{h^M(\vec{u}_i)}{M} \Big) \Big(\frac{h^M(\vec{v}_j)}{M} \Big) \Big| + \frac{2\sqrt{3}}{M} \cdot \mathrm{SDP}(A) \,.
\end{aligned}$$

Here the key step happened in the last line. We obtained it by bounding each of the second and third terms in the line above by $\frac{\sqrt{3}}{M}\mathrm{SDP}(A)$. To justify this we interpreted e.g. $h(\vec{u}_i)$ as a vector indexed by all possible realizations of the random vector $\vec{g}$. This is a vector of exponential dimension, but its norm is bounded by Lemma 3.19. Thus scaling the vectors so that they all have norm 1, the quantity cannot exceed the value $\mathrm{SDP}(A)$, by definition of that quantity as a supremum. This gives the claimed bound.

To conclude the analysis of the randomized rounding procedure, note that there must exist a $\vec{g}$ such that

$$\sum_{i,j} A_{ij} \Big(\frac{h^M(\vec{g} \cdot \vec{u}_i)}{M} \Big) \Big(\frac{h^M(\vec{g} \cdot \vec{v}_j)}{M} \Big) \ge \frac{1}{M^2} \Big(\mathrm{SDP}(A) - \frac{2\sqrt{3}}{M} \mathrm{SDP}(A) \Big) \,.$$

To conclude, we set $M = 4$, $x_i = (h^M(\vec{g} \cdot \vec{u}_i))/M$, $y_j = (h^M(\vec{g} \cdot \vec{v}_j))/M$ and get

$$\sum_{i,j} A_{ij} x_{ik} y_{jk} \ge \mathrm{SDP}(A) \cdot \Big(\frac{M - 2\sqrt{3}}{M^3} \Big) \approx 0.01 \cdot \mathrm{SDP}(A).$$

Here $x_i, y_j \in [-1, 1]$, and we can always find values in $\{\pm 1\}$ that are at least as good (to see how, fix all the x_i and observe that there is always an optimal setting of each individual y_j that is either $+1$ or -1).

It only remains to prove the lemma.

Proof of Lemma 3.19. (1) We check that

$$\begin{aligned}
\mathop{\mathbf{E}}_{\vec{g}} \big[h(\vec{u}) \cdot h(\vec{v}) \big] &= \mathop{\mathbf{E}}_{\vec{g}} \Big[\sum_{i,j} g_i u_i g_j v_j \Big]\\
&= \vec{u} \cdot \vec{v} \,,
\end{aligned}$$

since $\mathbf{E}[g_i g_j] = \delta_{ij}$.

(2) Using (1),

$$\mathop{\mathbf{E}}_{\vec{g}} |h(\vec{u})|^2 = \vec{u} \cdot \vec{u} = \|\vec{u}\|^2 = 1.$$

(3) Using (2),

$$\mathop{\mathbf{E}}_{\vec{g}} |h^M(\vec{u})|^2 \leq \mathop{\mathbf{E}}_{\vec{g}} \|h(\vec{u})\|^2 = 1.$$

(4) From the definition of $h^M(\cdot)$, we have

$$\begin{aligned}
\mathop{\mathbf{E}}_{\vec{g}} |h(\vec{u}) - h^M(\vec{u})|^2 &\leq \mathop{\mathbf{E}}_{\vec{g}} \left[|h(\vec{u})|^2 \, 1_{|\vec{u}\cdot\vec{g}|>M} \right] \\
&\leq \Big(\mathop{\mathbf{E}}_{\vec{g}} |h(\vec{u})|^4 \Big)^{1/2} \Big(\mathop{\mathbf{Pr}}_{\vec{g}} \left(|\vec{u} \cdot \vec{g}| > M \right) \Big)^{1/2} \\
&\leq \sqrt{3} \frac{\sqrt{3}}{M^2} \\
&= \frac{3}{M^2},
\end{aligned}$$

where the second line is by the Cauchy-Schwarz inequality and the third uses the fourth moment bound

$$\begin{aligned}
\mathop{\mathbf{E}}_{\vec{g}} \left[|\vec{u} \cdot \vec{g}|^4 \right] &= \mathop{\mathbf{E}}_{\vec{g}} \Big[\sum_{i,j,l,m} \vec{g}_i \vec{g}_j \vec{g}_l \vec{g}_m \vec{u}_i \vec{u}_j \vec{u}_l \vec{u}_m \Big] \\
&= \sum_i (\vec{u}_i)^4 + 3 \sum_{i \neq l} (\vec{u}_i)^2 (\vec{u}_l)^2 \\
&\leq 3 \big(\|\vec{u}_i\|^2 \big)^2 \\
&= 3,
\end{aligned}$$

where the second line follows by observing that $\mathbf{E}_{\vec{g}} \, \vec{g}_i \vec{g}_j \vec{g}_l \vec{g}_m = 0$ unless $i = j$ and $l = m$, or $i = l$ and $j = m$, or $i = m$ and $j = l$. This bound the first term; to bound the second we apply Chebyshev's inequality.

□

Remark 3.20. The same guarantees for the rounding procedure can be obtained by taking random projections on Gaussian vectors.

3.7.4 A deterministic rounding procedure

Our analysis so far assumes a uniformly random choice of $\vec{g} \in \{\pm 1\}^d$. But do we need $\vec{g}$ to be uniformly random? The only part where the distribution of $\vec{g}$ is used is in the proof of Lemma 3.19. Looking at the proof closely, we see that only 4-wise independence is used, to prove (4).

Recall from Lecture 1 that we call a family of vectors $\vec{g}_1, \ldots, \vec{g}_t \in \mathbb{R}^d$ k-wise independent if for any k distinct coordinates $j_1, \ldots, j_k \in \{1, \ldots, d\}$, the random variables $(X_{j_1}, \ldots, X_{j_k})$

defined by selecting a random $i \in \{1, \ldots, t\}$ and returning $((\vec{g}_i)_{j_1}, \ldots, (\vec{g}_i)_{j_k})$ are independent. Say furthermore that the vectors are *uniformly* k-wise independent if the random variables we just defined are uniformly distributed.

In your homework you proved the following claim using a construction based on Vandermonde matrices:

Claim 3.21. *In dimension d, there exists $t = O(d^2)$ vectors $\vec{g}_1, \ldots, \vec{g}_t \in \{\pm 1\}^d$, such that the $\vec{g}_i$ are uniformly 4-wise independent.*

Thus the analysis in the previous section goes through exactly unchanged if, instead of choosing $\vec{g}$ uniformly at random, we choose it uniformly at random over the family of t vectors guaranteed by the claim. But we can construct these t vectors explicitly, perform the rounding for each one of then, and choose the best solution found.