CS331: Algorithms and Complexity
Part VI: Continuous Algorithms

Kevin Tian

1 Introduction

In this part of the notes, we focus on algorithms for continuous optimization problems. Many algorithms we have studied so far in this course can naturally be phrased as optimization problems over a variable $\mathbf{x}$ living in some constraint set $\mathcal{X}$, i.e., for some $f : \mathcal{X} \to \mathbb{R}$, we want to solve

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \text{ or } \max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}). \tag{1}$$

We call $\mathbf{x}$ the *decision variable.* Essentially every problem in Parts III, IV, and V of the notes had the form (1) for an appropriate choice of $\mathcal{X}$ and f. For example, in scheduling, $\mathcal{X}$ was the set of all non-overlapping subsets of $[n]$, representing n intervals, and f was the subset size. Similarly, in s-t shortest path, $\mathcal{X}$ was the set of all s-t paths in a graph, and f was the total weight.

In all of these examples, $\mathcal{X}$ is discrete, i.e., we are asking for the best object out of a finite set of objects. However, we live in a continuous world, and many optimization problems in applications are naturally continuous, i.e., where $\mathcal{X}$ contains infinitely many points. This unit serves as a gentle introduction to techniques for approaching such problems. In the rest of these notes, the decision variable $\mathbf{x} \in \mathbb{R}^d$ will always be a vector, and $\mathcal{X} \subseteq \mathbb{R}^d$ will always be a continuous set.

To illustrate, consider the s-t maxflow problem on $G = (V, E, \mathbf{c})$, where the dimension $d = |E|$ is the edge count; we identify $E \equiv [d]$ in some order. The vector $\mathbf{x} \in \mathbb{R}^E$ represents a feasible s-t flow, and our goal is to solve the max variant of (1), where $f(\mathbf{x}) := \sum_{e=(s,v)\in E} \mathbf{x}_e$ and

$$\mathcal{X} := \left\{ \mathbf{x} \in \mathbb{R}^E_{\geq 0} \mid \mathbf{x}_e \leq \mathbf{c}_e \text{ for all } e \in E, \sum_{e=(v,u)\in E} \mathbf{x}_e - \sum_{e=(u,v)\in E} \mathbf{x}_e = 0 \text{ for all } v \in V \setminus \{s,t\} \right\}. \tag{2}$$

The problem of maximizing f over the continuous set (2), written in this way, appears quite daunting. However, in Section 4.3, Part V, we were nonetheless able to capitalize upon the structure of graphs to design algorithms for solving s-t maxflow. The purpose of these notes is to develop more general principles for solving structured continuous optimization problems.

Before describing the types of structure we consider, we should address an important issue: what sort of operations are we allowed to perform when solving (1)? Certainly being able to evaluate f at a given point $\mathbf{x}$ is a minimal requirement. In some natural applications, it is reasonable to assume that this is essentially the only form of access to f we have, e.g., when f is an extremely complicated function such as a deep neural network. If the function f is not too irregular, this also lets us approximate derivatives of f through finite differences: e.g., in one dimension,

$$f'(x) \approx \frac{f(x+\delta) - f(x)}{\delta} \tag{3}$$

for small δ, which can be computed in two function evaluations to f. In the more structured examples in these notes, where we have a more explicit description of the function f or constraint set $\mathcal{X}$, we will be afforded additional types of access to the problem.

Let us now give a simple example that demonstrates some pitfalls which can occur when performing continuous optimization in settings that are too unstructured.

Lemma 1. *Let $f : [0,1] \to [0,1]$. There is no algorithm that queries f at a finite number of points and returns a point $\hat{x} \in [0,1]$ such that $f(\hat{x}) - \min_{x\in[0,1]} f(x) \leq 0.99$.*

Proof. Consider a family of functions $f_{\bar{x}} : [0,1] \to [0,1]$, indexed by $\bar{x} \in [0,1]$. We define $f_{\bar{x}}$ as:

$$f_{\bar{x}}(x) = \begin{cases} 0 & x = \bar{x} \\ 1 & x \neq \bar{x} \end{cases}.$$

Suppose f is chosen to be one of the $f_{\bar{x}}$ for an unknown $\bar{x}$. If our algorithm has found an $\hat{x}$ that satisfies $f(\hat{x}) - \min_{x\in[0,1]} f(x) \le 0.99$, then we must have $\hat{x} = \bar{x}$, i.e., we have learned the value of $\bar{x}$. However, this is a contradiction; given any finite set of function queries, there are an infinite number of $\bar{x}$ that make all queries evaluate to 1, preventing us from learning $\bar{x}$. □

At face value Lemma 1 seems quite discouraging. Even if the function f is one-dimensional, with domain and range bounded by 1, and even if our goal is only to produce a point whose value is within 0.99 of the optimum, the problem (1) is still impossible. Nonetheless, we will see that in many interesting cases, that a priori may seem even more challenging than the problem in Lemma 1, we can give efficient algorithms. The key challenge encountered in Lemma 1 was the lack of information; querying suboptimal points gives no clues as to where the minimizer lies.

When can we do better? A simple example is *unimodal functions* in one dimension, i.e., functions $f : [a,b] \to \mathbb{R}$ which are decreasing over $[a,c]$ and increasing over $[c,b]$, for some $a \le c \le b$ (so the minimizer is c). Here, a strategy called ternary search lets us quickly shrink the range in which the minimizer can lie by a constant factor each iteration. More concretely, assuming we know a candidate range $[\ell, r] \subset [a,b]$ containing c, querying $f(\frac{2\ell}{3} + \frac{r}{3})$ and $f(\frac{\ell}{3} + \frac{2r}{3})$ lets us eliminate a third of the candidate range, depending on which query is larger, upon which we can recurse.

The key difference between these two examples is that in the latter case, we were able to use function query information to locally improve and hone in on the minimizer. Indeed, *convexity*, a variant of unimodality which extends to high dimensions, will be a recurring theme in these notes, as an example of structure that gives way to tractability in continuous optimization.

We remark that the author of these notes developed a new course on continuous algorithms. The interested reader can find a much more in-depth discussion of any topic in these notes at [Tia24a].

2 Linear programming

One of the most ubiquitous structured optimization problems that admits efficient algorithms is *linear programming*. Here, we introduce linear programs, focusing on their use as a modeling tool.

Let $\mathbf{x} \in \mathbb{R}^d$ be a vector-valued variable. We call $f(\mathbf{x})$ a *linear function* if $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x} = \sum_{i\in[d]} \mathbf{c}_i \mathbf{x}_i$, for some $\mathbf{c} \in \mathbb{R}^d$. For $\mathbf{a} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, we call a set of the form $\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{a}^\top \mathbf{x} \le b\}$ a *halfspace*. Intuitively, a halfspace is one half of $\mathbb{R}^d$ as bisected by a $(d-1)$-dimensional hyperplane, satisfying the one-dimensional constraint $\mathbf{a}^\top \mathbf{x} = b$. Finally, a *polytope* is an intersection of halfspaces:

$$\left\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{a}_i^\top \mathbf{x} \le b_i, \text{ for all } i \in [n]\right\}. \tag{4}$$

We remark that the polytope (4) can be more concisely described using the notation

$$\left\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{A}\mathbf{x} \le \mathbf{b}\right\},$$

where $\mathbf{b} \in \mathbb{R}^n$ has the thresholds $\{b_i\}_{i\in[n]}$ as coordinates, and $\mathbf{A} \in \mathbb{R}^{n\times d}$ has the vectors $\{\mathbf{a}_i^\top\}_{i\in[n]}$ as rows. Above and throughout the section, when comparing vectors, we treat $\le$ as enforcing coordinatewise constraints. A linear program (LP) asks to optimize a linear function over a polytope:

$$\max_{\substack{\mathbf{x}\in\mathbb{R}^d \\ \mathbf{A}\mathbf{x}\le\mathbf{b}}} \mathbf{c}^\top \mathbf{x}. \tag{5}$$

To gain some intuition for (5), suppose that the linear function satisfies $\mathbf{c} = \mathbf{e}_1$, i.e., $\mathbf{c}^\top \mathbf{x} = \mathbf{x}_1$ and our goal is to make the first coordinate as large as possible. Then (5) can be directly interpreted as asking for the "northernmost point in a polytope," treating the first coordinate axis as north-south for simplicity. More generally, up to rotating the polytope so the direction of interest $\mathbf{c}$ points north, this intuition captures the general geometry of LP objectives.

The fact that all linear programs can be optimized to high precision in polynomial time is incredibly powerful in the design of algorithms. As we explain shortly, almost every problem in Part V of the notes can be written in the form (4). Thus, faster LP algorithms can directly yield faster graph algorithms (and much more). Indeed, many of the fastest known maxflow algorithms adopt this perspective, evolving flow variables continuously rather than via discrete updates along paths.

2.1 Expressivity of LPs

Linear programs are far more expressive than the formulation (5) initially suggests. For example, it admits the following straightforward extensions, among many others.

Negation. Suppose we want to minimize a linear function over a polytope, i.e., to solve (5) with min replacing max. To do so, we can simply replace $\mathbf{c} \leftarrow -\mathbf{c}$; indeed, $(-\mathbf{c})^\top \mathbf{x}$ is still linear, and its maximizer is the minimizer to $\mathbf{c}^\top \mathbf{x}$. Similarly, if we wish to enforce lower bounds with our constraints rather than upper bounds, i.e., a halfspace constraint of the form $\mathbf{a}^\top \mathbf{x} \geq b$, we can rewrite this as the halfspace $(-\mathbf{a})^\top \mathbf{x} \leq (-b)$ and add an appropriate row and entry to $(\mathbf{A}, \mathbf{b})$.

Equality. We can also enforce linear equalities with a similar trick. Suppose we want to add the constraint $\mathbf{a}^\top \mathbf{x} = b$ to the linear program (5). This can be enforced using two inequality constraints of the form $\mathbf{a}^\top \mathbf{x} \leq b$ and $\mathbf{a}^\top \mathbf{x} \geq b$, where we know how to rewrite the latter as $(-\mathbf{a})^\top \mathbf{x} \leq b$. Thus adding these two inequality constraints to the polytope yields an equality constraint.

Absolute values. As a third example, suppose we want to solve a problem of the form

$$\min_{\substack{\mathbf{x} \in \mathbb{R}^d \\ \mathbf{A}\mathbf{x} \leq \mathbf{b}}} \mathbf{c}^\top |\mathbf{x}| := \sum_{i \in [d]} \mathbf{c}_i |\mathbf{x}_i|, \tag{6}$$

i.e., where $|\mathbf{x}|$ is elementwise and $\mathbf{c} \in \mathbb{R}^d_{\geq 0}$ is nonnegative entrywise. It is not a priori clear that this is an LP. To see this, we first rewrite the problem (6): by our earlier discussion of negations, to solve (6), it suffices to solve the maximization variant

$$\max_{\substack{\mathbf{x} \in \mathbb{R}^d \\ \mathbf{A}\mathbf{x} \leq \mathbf{b}}} \sum_{i \in [d]} \mathbf{c}_i (-|\mathbf{x}_i|). \tag{7}$$

Next, we can create a new optimization problem with an auxiliary variable $\mathbf{y} \in \mathbb{R}^d$, so that our new decision variable is the $2d$-dimensional vector $\mathbf{z}$ concatenating $\mathbf{x}$ and $\mathbf{y}$, i.e., with $\mathbf{z}_i = \mathbf{x}_i$ and $\mathbf{z}_{d+i} = \mathbf{y}_i$ for all $i \in [d]$. In addition to $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, we also enforce the linear constraints

$$\mathbf{y}_i \leq \mathbf{x}_i \iff (\mathbf{e}_{i+d} - \mathbf{e}_i)^\top \mathbf{z} \leq 0 \text{ and } \mathbf{y}_i \leq -\mathbf{x}_i \iff (\mathbf{e}_{i+d} + \mathbf{e}_i)^\top \mathbf{z} \leq 0, \tag{8}$$

and make our new objective $\mathbf{c}^\top \mathbf{y} = \sum_{i \in [d]} \mathbf{c}_i \mathbf{y}_i$, which is a linear function of $\mathbf{z}$ (it only depends on the last d coordinates). Thus our new problem asks to solve $\max \mathbf{c}^\top \mathbf{y}$ subject to $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and the constraints (8). As discussed, this is an LP in the concatenated decision variable $\mathbf{z}$.

Now to maximize $\mathbf{c}^\top \mathbf{y}$, since $\mathbf{c}$ was assumed nonnegative, we should take each coordinate of $\mathbf{y}$ as large as possible. Given that our only constraints on each $\mathbf{y}_i$ are $\mathbf{y}_i \leq \mathbf{x}_i$ and $\mathbf{y}_i \leq -\mathbf{x}_i$, we have that at optimality, $\mathbf{y}_i = \min(\mathbf{x}_i, -\mathbf{x}_i) = -|\mathbf{x}_i|$, so our LP fully captures the variant (7) as required.

For reasons that will be clarified later, the nonnegativity condition $\mathbf{c} \in \mathbb{R}^d_{\geq 0}$ is necessary in allowing us to write (6) as an LP. This essentially is due to the fact that LPs are convex problems, and that the function $f(x) = |x|$ is convex but its negation $f(x) = -|x|$ is not.

2.2 Primal and dual LPs

One of the most important facts about LPs is that they have a naturally primal-dual structure. In particular, for every LP, there is a *dual LP* that is equivalent to the original problem. We often call the original LP of interest the *primal LP*. In this section, we introduce and explore this characterization of LPs, along with several illustrative examples.

Forms of duality. Let $\mathbf{A} \in \mathbb{R}^{n \times d}$, $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{c} \in \mathbb{R}^d$ be fixed. The most basic formulation of LP is (5), which has an equivalent dual formulation as follows:

$$\max_{\substack{\mathbf{x} \in \mathbb{R}^d \\ \mathbf{A}\mathbf{x} \leq \mathbf{b}}} \mathbf{c}^\top \mathbf{x} = \min_{\substack{\mathbf{y} \in \mathbb{R}^n_{\geq 0} \\ \mathbf{A}^\top \mathbf{y} = \mathbf{c}}} \mathbf{b}^\top \mathbf{y}. \tag{9}$$

That is, the constraint thresholds $\mathbf{b}$ in the primal LP on the left define the objective function in the dual LP on the right, and vice versa with $\mathbf{c}$, which defined the primal objective function.

One somewhat unsatisfying aspect of (9) is its asymmetry. The primal LP has inequality constraints $\mathbf{A}\mathbf{x} \le \mathbf{b}$, and the dual LP has equality constraints $\mathbf{A}^\top \mathbf{y} = \mathbf{c}$, along with nonnegativity constraints $\mathbf{y} \in \mathbb{R}^n_{\ge 0}$. An easier variant to remember is the symmetric primal-dual formulation

$$\max_{\substack{\mathbf{x}\in\mathbb{R}^d_{\ge 0}\\ \mathbf{A}\mathbf{x}\le\mathbf{b}}} \mathbf{c}^\top \mathbf{x} = \min_{\substack{\mathbf{y}\in\mathbb{R}^n_{\ge 0}\\ \mathbf{A}^\top\mathbf{y}\ge\mathbf{c}}} \mathbf{b}^\top \mathbf{y}. \tag{10}$$

In (10), both the primal and dual decision variables, $\mathbf{x}$ and $\mathbf{y}$, are constrained to be entrywise nonnegative. By our discussion in Section 2.1, we can rewrite the primal LP in (10) in the more basic form (5), using additional constraints, appropriately negated. It is an instructive exercise to check that performing these steps and applying (9) gives a consistent result with (10):

$$\begin{aligned}\max_{\substack{\mathbf{x}\in\mathbb{R}^d_{\ge 0}\\ \mathbf{A}\mathbf{x}\le\mathbf{b}}} \mathbf{c}^\top \mathbf{x} &= \max_{\mathbf{x}\in\mathbb{R}^d} \mathbf{c}^\top \mathbf{x} \text{ subject to } \begin{pmatrix}\mathbf{A}\\ -\mathbf{I}_d\end{pmatrix}\mathbf{x} \le \begin{pmatrix}\mathbf{b}\\ \mathbf{0}_d\end{pmatrix}\\ &= \min_{\substack{\mathbf{y}\in\mathbb{R}^n_{\ge 0}\\ \mathbf{s}\in\mathbb{R}^d_{\ge 0}}} \mathbf{b}^\top \mathbf{y} \text{ subject to } \begin{pmatrix}\mathbf{A}^\top & -\mathbf{I}_d\end{pmatrix}\begin{pmatrix}\mathbf{y}\\ \mathbf{s}\end{pmatrix} = \mathbf{c}\\ &= \min_{\substack{\mathbf{y}\in\mathbb{R}^n_{\ge 0}\\ \mathbf{s}\in\mathbb{R}^d_{\ge 0}}} \mathbf{b}^\top \mathbf{y} \text{ subject to } \mathbf{A}^\top\mathbf{y} = \mathbf{c}+\mathbf{s} = \min_{\substack{\mathbf{y}\in\mathbb{R}^n_{\ge 0}\\ \mathbf{A}^\top\mathbf{y}\ge\mathbf{c}}} \mathbf{b}^\top \mathbf{y}.\end{aligned} \tag{11}$$

Here, $\mathbf{I}_d$ is the $d \times d$ identity matrix (see Section 5, Part I for a review), $\mathbf{0}_d$ is the all-zeroes vector in $\mathbb{R}^d$, and we used that for an arbitrary nonnegative vector $\mathbf{s}$, $\mathbf{A}^\top\mathbf{y} = \mathbf{c} + \mathbf{s} \iff \mathbf{A}^\top\mathbf{y} \ge \mathbf{c}$. The auxiliary variables $\mathbf{s}$ introduced in this derivation are often called *slack variables.*

We discuss and sketch a proof of (9), (10), collectively called *strong LP duality*, in Section 2.3.

Resource allocation. The symmetric relationship (10) is perhaps easiest to motivate when all of $\mathbf{A}, \mathbf{b}, \mathbf{c}$ are entrywise nonnegative. Problems of this form are known as *packing-covering LPs*, and frequently arise as models of resource allocation. Suppose we have n types of raw *materials*, and d types of *products* which can be made from these materials. We can view the primal problem

$$\max_{\substack{\mathbf{x}\in\mathbb{R}^d\\ \mathbf{A}\mathbf{x}\le\mathbf{b}}} \mathbf{c}^\top \mathbf{x}$$

as answering the following question: what is the most profit can we make, by converting available materials into products? The parameters in this problem have the following interpretations.

- For each $j \in [d]$, the column $\mathbf{A}_{:j} \in \mathbb{R}^n$ indicates the "recipe" for product j. That is, one unit of product j requires $\mathbf{A}_{ij}$ units of material i, for all $i \in [n]$.
- $\mathbf{b} \in \mathbb{R}^n_{\ge 0}$ indicates the amount of each material available to us.
- $\mathbf{c} \in \mathbb{R}^d_{\ge 0}$ indicates the selling price of each type of product.
- $\mathbf{x} \in \mathbb{R}^d_{\ge 0}$ indicates the amount of each product we wish to produce.

The constraints $\mathbf{A}\mathbf{x} \le \mathbf{b}$ enforce that the total materials used does not exceed what is available, and the objective function $\mathbf{c}^\top\mathbf{x}$ indicates the profit we will make. Now consider the dual LP,

$$\min_{\substack{\mathbf{y}\in\mathbb{R}^n_{\ge 0}\\ \mathbf{A}^\top\mathbf{y}\ge\mathbf{c}}} \mathbf{b}^\top \mathbf{y}.$$

We can view the dual problem as capturing a competitor who wishes to buy up all of our raw materials so they can start their own factory. The decision variable $\mathbf{y}$ corresponds to the per-unit price offered to us by our competitor, so that $\mathbf{b}^\top\mathbf{y} = \sum_{i\in[n]} \mathbf{b}_i\mathbf{y}_i$ is the amount they must spend to buy our $\mathbf{b}_i$ units of each material $i \in [n]$. Moreover, for each $j \in [d]$ the inequality

$$\left[\mathbf{A}^\top\mathbf{y}\right]_j \ge \mathbf{c}_j \iff \sum_{i\in[n]} \mathbf{A}_{ij}\mathbf{y}_i \ge \mathbf{c}_j$$

enforces that applying the "recipe" for product j does not earn us more profit than selling to our competitor, as otherwise we are not incentivized to sell our materials. Thus, the dual LP computes the minimum our competitor must spend to dissuade us from entering the market ourselves.

LP duality says that no competitor can beat the open market. That is, after buying up $\mathbf{b}$ units of materials, a competitor cannot then flip these materials to make a profit. This is because their maximum profit and minimum spending are respectively captured by the primal and dual LPs.

Zero-sum games. Our resource allocation example is related to an application of LP in playing zero-sum games, which was actually John von Neumann's original motivation for developing LP theory [vN28]. Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ be a *payoff matrix*, interpreted as follows. We view the sets $[n]$ and $[d]$ as indexing *actions* that two competing players, Alice and Bob, can take, e.g., in rock-paper-scissors, $n = d = 3$. Then if Alice selects action $i \in [n]$ and Bob selects action $j \in [d]$, we treat $\mathbf{A}$ as denoting that Alice receives a reward of $\mathbf{A}_{ij}$, and Bob receives a reward of $-\mathbf{A}_{ij}$. As in Section 5.1, Part III, this is a zero-sum game, so Alice and Bob have directly competing goals.

More generally, we can imagine that Alice and Bob independently play distributions over strategies. This extra degree of freedom is clearly helpful; in rock-paper-scissors, any deterministic strategy will always lose to an optimal opponent. Let $\mathbf{y} \in \mathbb{R}^n_{\geq 0}$ with $\sum_{i \in [n]} \mathbf{y}_i = 1$ indicate a probability distribution over $[n]$ governing Alice's strategy, and similarly, let $\mathbf{x} \in \mathbb{R}^d_{\geq 0}$ with $\sum_{j \in [d]} \mathbf{x}_j$ govern Bob's strategy. Then Alice's expected reward, over the random strategies, is

$$\sum_{\substack{i \in [n] \\ j \in [d]}} \mathbf{A}_{ij} \mathbf{y}_i \mathbf{x}_j = \sum_{i \in [n]} \mathbf{y}_i \left(\sum_{j \in [d]} \mathbf{A}_{ij} \mathbf{x}_j \right) = \sum_{i \in [n]} \mathbf{y}_i \left[\mathbf{A}\mathbf{x}\right]_i = \mathbf{y}^\top \mathbf{A}\mathbf{x}. \tag{12}$$

Alice's goal is to choose $\mathbf{y}$ that maximizes (12), and Bob's goal is to choose $\mathbf{x}$ minimizing it.

We could further ask: what if Bob, with full knowledge of Alice's intended strategy $\mathbf{y}$, can then respond with his choice of $\mathbf{x}$? In light of (12), he should choose a distribution $\mathbf{x} \in \mathbb{R}^d$ to minimize $\mathbf{y}^\top \mathbf{A}\mathbf{x}$. To do this he should deterministically select only the action $j \in [d]$ minimizing $[\mathbf{A}^\top \mathbf{y}]_j$, i.e., use the strategy $\mathbf{x} = \mathbf{e}_j$. In this Bob-favored version of the game, Alice can guarantee at most

$$\max_{\substack{\mathbf{y} \in \mathbb{R}^n_{\geq 0} \\ \sum_{i \in [n]} \mathbf{y}_i = 1}} \min_{j \in [d]} \left[\mathbf{A}^\top \mathbf{y}\right]_j$$

in payoff. Similarly, one can define an Alice-favored version of the game, where Alice knows Bob's intended strategy $\mathbf{x}$ and then can choose her action to maximize $\mathbf{y}^\top \mathbf{A}\mathbf{x} = \mathbf{x}^\top \mathbf{A}^\top \mathbf{y}$. Analogously, Alice will only select the largest coordinate of $\mathbf{A}\mathbf{x}$. The best Bob can do is thus captured by

$$\min_{\substack{\mathbf{x} \in \mathbb{R}^d_{\geq 0} \\ \sum_{j \in [d]} \mathbf{x}_j = 1}} \max_{i \in [n]} \left[\mathbf{A}\mathbf{x}\right]_i .$$

Amazingly, an application of LP duality shows that these two values are equal:

$$\max_{\substack{\mathbf{y} \in \mathbb{R}^n_{\geq 0} \\ \sum_{i \in [n]} \mathbf{y}_i = 1}} \min_{j \in [d]} \left[\mathbf{A}^\top \mathbf{y}\right]_j = \min_{\substack{\mathbf{x} \in \mathbb{R}^d_{\geq 0} \\ \sum_{j \in [d]} \mathbf{x}_j = 1}} \max_{i \in [n]} \left[\mathbf{A}\mathbf{x}\right]_i . \tag{13}$$

Thus, the Alice-favored game and the Bob-favored game end in the same result. The counterintuitive fact (13) is a primitive proof of existence of Nash equilibria, specialized to zero-sum games.

Maxflow and mincut. We conclude by writing the maxflow-mincut theorem from Part V as an instance of LP duality. Similar results can be derived for reachability and shortest paths problems, but require some care to simplify, so we defer an extended discussion to Appendix H, [Eri24].

We already wrote s-t maxflow as a continuous optimization problem in (2). Upon observation, the constraint set (2) is a polytope since all constraints are linear functions of $\mathbf{x}$, and the objective function is also linear. One way to encode this problem more concisely as an LP is by using the *incidence matrix* $\mathbf{A}$ of a graph $G = (V, E, \mathbf{w})$, defined as follows. Let $\mathbf{A} \in \{-1, 0, 1\}^{V \times E}$ have a column for every edge $e \in E$. Then for $e = (a, b) \in E$, the column $\mathbf{A}_{:e} \in \mathbb{R}^V$ has

$$\mathbf{A}_{ve} = \begin{cases} -1 & v = b \\ 1 & v = a \\ 0 & v \in V \setminus \{a, b\} \end{cases} . \tag{14}$$

For a flow vector $\mathbf{x} \in \mathbb{R}^E$, we can check that $[\mathbf{A}\mathbf{x}]_v$ encodes the netflow at vertex $v \in V$ (see Section 4.1, Part V), because it sums all flows on edges e where v is the head of e (i.e., v is producing the flow), and removes all flows on edges where v is the tail (i.e., v is consuming the flow).

Then, letting $U := V \setminus \{s, t\}$ be all vertices other than the source and sink, we can rewrite the s-t maxflow problem using the following primal LP:

$$\max_{\substack{\mathbf{x}\in\mathbb{R}^E_{\geq 0} \\ \mathbf{x}\leq\mathbf{c} \\ [\mathbf{A}\mathbf{x}]_U=\mathbf{0}_U}} [\mathbf{A}\mathbf{x}]_s\,,$$

where we let subscripting by U denote restricting to coordinates in U, and $\mathbf{0}_U$ denote the all-zeroes vector in $\mathbb{R}^U$. After some massaging, taking the dual of this primal LP yields

$$\max_{\substack{\mathbf{x}\in\mathbb{R}^E_{\geq 0} \\ \mathbf{x}\leq\mathbf{c} \\ [\mathbf{A}\mathbf{x}]_U=\mathbf{0}_U}} [\mathbf{A}\mathbf{x}]_s = \min_{\substack{\mathbf{y}\in\mathbb{R}^E_{\geq 0},\mathbf{z}\in\mathbb{R}^V \\ \mathbf{z}_s=1,\mathbf{z}_t=0 \\ \mathbf{y}_e\geq\mathbf{z}_u-\mathbf{z}_v \text{ for all } e=(u,v)\in E}} \mathbf{c}^\top\mathbf{y}. \tag{15}$$

The equation (15) exactly encodes maxflow-mincut duality; we already argued the left-hand side computes the s-t maxflow. To demystify the right-hand side, the decision variable $\mathbf{z} \in \mathbb{R}^V$ can be interpreted as assigining vertices in V to the two sides of a cut between S and $V \setminus S$. The constraints $\mathbf{z}_s = 1$ and $\mathbf{z}_t = 0$ encode $s \in S$ and $t \notin S$, and the optimal $\mathbf{z}$ is the 0-1 indicator of the minimum cut S. Moreover, the constraint $\mathbf{y}_e \geq \mathbf{z}_u - \mathbf{z}_v$ is only nontrivial when $\mathbf{z}_u - \mathbf{z}_v \geq 0$. This happens only when $\mathbf{z}_u = 1$ and $\mathbf{z}_v = 0$, i.e., $e = (u, v)$ for $u \in S$ and $v \notin S$. These are exactly the cut edges, so the optimal $\mathbf{y}$ has value 1 for these edges and 0 everywhere else. Thus, the right-hand side of (15) exactly sums the capacities of all edges going from S to $V \setminus S$, i.e., cut(S).

2.3 Weak and strong duality

In this section, we provide justification for why the LP duality relationships (9), (10) hold.

Weak duality. As in the case of maxflow-mincut theorems, there is a weak version of LP duality that says (9), (10) hold with inequality, that is easier to prove, so we begin by giving this argument.

Lemma 2. *For any* $\mathbf{A} \in \mathbb{R}^{n\times d}, \mathbf{b} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^d$, *we have*

$$\max_{\substack{\mathbf{x}\in\mathbb{R}^d_{\geq 0} \\ \mathbf{A}\mathbf{x}\leq\mathbf{b}}} \mathbf{c}^\top\mathbf{x} \leq \min_{\substack{\mathbf{y}\in\mathbb{R}^n_{\geq 0} \\ \mathbf{A}^\top\mathbf{y}\geq\mathbf{c}}} \mathbf{b}^\top\mathbf{y}.$$

Proof. Let $\mathbf{x} \in \mathbb{R}^d_{\geq 0}$ and $\mathbf{y} \in \mathbb{R}^n_{\geq 0}$ respectively satisfy $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{A}^\top\mathbf{y} \geq \mathbf{c}$. We claim that for any such $\mathbf{x}, \mathbf{y}$, we have $\mathbf{c}^\top\mathbf{x} \leq \mathbf{b}^\top\mathbf{y}$; choosing $\mathbf{x}, \mathbf{y}$ optimally to solve their respective LPs then gives the conclusion. We now prove our claim $\mathbf{c}^\top\mathbf{x} \leq \mathbf{b}^\top\mathbf{y}$. Because $\mathbf{x}$ is entrywise nonnegative,

$$\mathbf{A}^\top\mathbf{y} \geq \mathbf{c} \implies \sum_{j\in[d]} \left[\mathbf{A}^\top\mathbf{y}\right]_j \mathbf{x}_j \geq \sum_{j\in[d]} \mathbf{c}_j\mathbf{x}_j \iff \mathbf{y}^\top\mathbf{A}\mathbf{x} \geq \mathbf{c}^\top\mathbf{x},$$

where we used the derivation (12) in the last equality. Similarly, nonnegativity of $\mathbf{y}$ yields

$$\mathbf{A}\mathbf{x} \leq \mathbf{b} \implies \sum_{i\in[n]} \mathbf{y}_i \left[\mathbf{A}\mathbf{x}\right]_i \leq \sum_{i\in[n]} \mathbf{b}_i\mathbf{y}_i \iff \mathbf{y}^\top\mathbf{A}\mathbf{x} \leq \mathbf{b}^\top\mathbf{y}.$$

Combining the above expressions proves $\mathbf{c}^\top\mathbf{x} \leq \mathbf{b}^\top\mathbf{y}$ as claimed. □

Lemma 2 hence establishes the weak form of (10). The weak form of (9) similarly follows by establishing $\mathbf{y}^\top\mathbf{A}\mathbf{x} = \mathbf{c}^\top\mathbf{x}$ and $\mathbf{y}^\top\mathbf{A}\mathbf{x} \leq \mathbf{b}^\top\mathbf{y}$, the latter using nonnegativity of $\mathbf{y}$.

Strong duality. We next briefly sketch a proof of strong duality. Here it is convenient to start with the asymmetric form (9), so the primal problem is $\max_{\mathbf{x}\in\mathbb{R}^d|\mathbf{A}\mathbf{x}\leq\mathbf{b}} \mathbf{c}^\top\mathbf{x}$. Let $\mathbf{x}^\star \in \mathbb{R}^d$ be the optimal solution, and let $I \subseteq [n]$ denote the set of *tight constraints*, i.e., the constraints $i \in [n]$ satisfying $\mathbf{a}_i^\top\mathbf{x} = \mathbf{b}_i$ with equality rather than inequality. In summary,

$$\mathbf{a}_i^\top\mathbf{x} = \mathbf{b}_i \text{ for all } i \in I, \text{ and } \mathbf{a}_i^\top\mathbf{x} < \mathbf{b}_i \text{ for all } i \in [n] \setminus I. \tag{16}$$

Our key claim is that the objective direction, $\mathbf{c} \in \mathbb{R}^d$, is a *nonnegative linear combination* of the tight constraints $\{\mathbf{a}_i\}_{i \in I}$. The set of vectors expressible in this way is called a *cone*:

$$C := \left\{ \mathbf{v} \in \mathbb{R}^d \mid \mathbf{v} = \sum_{i \in I} \mathbf{y}_i \mathbf{a}_i \text{ for } \mathbf{y} \in \mathbb{R}^I_{\geq 0} \right\}. \tag{17}$$

Why is $\mathbf{c} \in C$? Intuitively, if $\mathbf{c}$ did not lie in this set, then some coefficient, say that of the constraint $\mathbf{a}_i$, is negative. By moving in the $-\mathbf{a}_i$ direction, we can then locally improve our correlation with the objective $\mathbf{c}$ (because the contribution of $\mathbf{a}_i$ is negative), while staying feasible for $\mathbf{a}_i^\top \mathbf{x} \leq \mathbf{b}_i$. This contradicts the optimality of $\mathbf{x}$. Making this intuition rigorous, i.e., quantifying the effect of this movement on other constraints, is the most challenging part of the strong duality proof, and we omit it for brevity in this proof sketch. The formal version is known as *Farkas' lemma.*

Now suppose that $\mathbf{c}$ belongs to the cone (17). By padding the coefficient vector $\mathbf{y} \in \mathbb{R}^I$ with zeroes appropriately, we can extend it to $\mathbf{y}^\star \in \mathbb{R}^n$ such that

$$\mathbf{c} = \mathbf{A}^\top \mathbf{y}^\star = \sum_{i \in [n]} \mathbf{y}_i^\star \mathbf{a}_i = \sum_{i \in I} \mathbf{y}_i^\star \mathbf{a}_i. \tag{18}$$

This is consistent with (17) because $\mathbf{y}_i^\star = 0$ for all $i \in [n] \setminus I$. Finally, we can finish the proof:

$$\mathbf{b}^\top \mathbf{y}^\star = \sum_{i \in I} \mathbf{b}_i \mathbf{y}_i^\star = \sum_{i \in I} [\mathbf{A}\mathbf{x}^\star]_i \, \mathbf{y}_i^\star = \left(\sum_{i \in I} \mathbf{y}_i^\star \mathbf{a}_i \right)^\top \mathbf{x}^\star = \mathbf{c}^\top \mathbf{x}^\star.$$

Here the first equality used $\mathbf{y}_i^\star = 0$ outside I, the second used (16), the third rearranged the expression $[\mathbf{A}\mathbf{x}^\star]_i = \mathbf{a}_i^\top \mathbf{x}^\star$, and the last used (18). We have thus given a pair $(\mathbf{x}^\star, \mathbf{y}^\star)$ that are feasible for the two sides of (9), with equal objective values. This shows that the left-hand side of (9), a maximization over feasible $\mathbf{x}^\star$, is at least the right-hand side, a minimization over feasible $\mathbf{y}^\star$. Combined with our weak duality result, this establishes that both sides are equal.

Feasibility and boundedness. It is worth remarking at this point that in general, the polytope $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ may not be very well-behaved, especially with regards to a specific LP, with objective $\mathbf{c}^\top \mathbf{x}$. There are two major pathological types of behavior that could happen.

If there are no points such that $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, i.e., the constraints are impossible to simultaneously satisfy, we call the LP *infeasible.* A simple example of an infeasible LP is the two constraints $\mathbf{e}_1^\top \mathbf{x} \leq 0$ and $-\mathbf{e}_1^\top \mathbf{x} \leq -1$; recall the latter constraint is equivalent to $1 \leq \mathbf{e}_1^\top \mathbf{x}$.

If the objective function can be arbitrarily maximized or minimized, i.e., the feasible region $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ includes points going off to infinity, we call the LP *unbounded.* A simple example of an unbounded LP is the problem $\max -\mathbf{e}_1^\top \mathbf{x}$, subject to the single constraint $\mathbf{e}_1^\top \mathbf{x} \leq 0$.

We extend our definitions of (9) and (10) so that if an LP involving maximization is infeasible, its value is $-\infty$, and if it involves minimization, its value is ∞. As a consequence, we can check that this implies that the dual of an unbounded LP is always infeasible.

3 Gradient descent

In this section, we develop the most famous continuous algorithm: gradient descent. Our focus in this section will be on motivating and analyzing gradient descent in the one-dimensional setting,[1] but we briefly discuss high-dimensional variants in Section 3.4.

3.1 Structural assumptions

As illustrated in Lemma 1, unstructured continuous optimization, i.e., solving $\min_{x \in \mathcal{X}} f(x)$, is impossible, even when $\mathcal{X} \subseteq \mathbb{R}$ is one-dimensional. However, under *structural assumptions* on f that hold in important practical applications, it turns out that we can optimize f efficiently assuming nothing more than value query access, i.e., the ability to evaluate $f(x)$ at a given $x \in \mathcal{X}$.

Convexity. Perhaps the most natural structural assumption to impose on f to admit efficient optimization algorithms is *convexity.* We begin by defining this property.

[1] We will hence stop bolding decision variables, i.e., a vector $\mathbf{x}$, as all variables are scalars.

Definition 1 (Convexity)**.** *We say that a function $f : \mathbb{R} \to \mathbb{R}$ is convex if for any two $x, y \in \mathbb{R}$,*

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y), \text{ for all } \lambda \in [0,1]. \tag{19}$$

We say f is strictly convex *if* (19) *holds with strict inequality, i.e., $<$ rather than $\leq$.*

Why is convexity helpful? One reason is that convex minimization is unambiguous; as you will explore in the homework, strictly convex functions are unimodal (see Section 1). Thus, all local minima are global minima, so we cannot be tricked into terminating an optimization algorithm.

Another major reason is that convexity implies helpful bounds, which can guide algorithms and measure their progress. For example, (19) has a simple interpretation as providing a function *upper bound*. In one dimension, this is interpretable as f lying below the line between $(x, f(x))$ and $(y, f(y))$, over the interval $[x, y]$. We can further rearrange (19) to obtain the *lower bound*

$$f(y) \geq \frac{f((1-\lambda)x + \lambda y) - (1-\lambda)f(x)}{\lambda} = f(x) + \frac{f(x + \lambda(y-x)) - f(x)}{\lambda}.$$

If f is differentiable, then by taking a limit as $\lambda \to 0$,

$$f(y) \geq f(x) + \lim_{\lambda \to 0} \frac{f(x + \lambda(y-x)) - f(x)}{\lambda} = f(x) + f'(x)(y-x). \tag{20}$$

You may recall from calculus that the Euler approximation to a function, centered at x, is

$$f(y) \approx f(x) + f'(x)(y-x). \tag{21}$$

The consequence (20) has a simple interpretation: for convex f, the Euler approximation is always an underestimate. One implication is that the derivative $f'(x)$ gives us a search direction for convex functions. In particular, let $x^\star := \text{argmin}_{x \in [0,1]} f(x)$, and suppose that for some $x \in [0,1]$, $f'(x) > 0$. Then we claim that the minimizer $x^\star$ cannot lie right of x: indeed,

$$f(y) \geq f(x) + f'(x)(y-x) > f(x) \text{ for all } y > x.$$

The first inequality was (20), and the second used $f'(x) > 0$. Analagously, if $f'(x) < 0$, we can conclude that the minimizer $x^\star$ cannot lie left of x. This implies that if we can query the derivative $f'(x)$, we can perform binary search to quickly narrow an interval containing $x^\star$.

Finally, we mention without proof that if f is twice-differentiable, a quick way to verify that f is convex is by establishing $f''(x) \geq 0$ pointwise. This implies that the gradient f' is an increasing function of x, which should match our intuition when, e.g., f is a quadratic.

Smoothness. In one dimension, the ability to query the gradient is already enough to search for the minimizer via binary search. However, in high dimensions, our strategy must change, because there are infinitely many potential search directions, so binary search is not well-defined.

This is where gradient descent comes into play: rather than use the sign of $f'(x)$ to eliminate a portion of our search space, we instead use it to gradually guide an iterate x towards $x^\star$. More concretely, for a *step size* $\eta > 0$, gradient descent repeatedly iterates the following update:

$$x \leftarrow x - \eta f'(x). \tag{22}$$

As a sanity check, for any $\eta > 0$, the update proceeds in the correct direction: if $f'(x) > 0$, then (22) moves x leftwards, where the minimizer $x^\star$ must lie. However, we can get into trouble if (22) overshoots $x^\star$, i.e., η is too large, in which case we must backtrack. On the other hand, if we set η too small, then we can make very little progress in each iteration, leading to a slow algorithm.

Why does (22) scale our movement proportionally to $f'(x)$? One reason is that the further away from the minimizer $x^\star$ we are, the larger $|f'(x)|$ is. This is a consequence of the fact that $f'' \geq 0$ pointwise, e.g., $f'(x^\star) = 0$, and as we increase x from $x^\star$, $f'(x)$ grows.

What η should we choose in (22)? A common structural property that can be used to prescribe a step size is *smoothness*. Informally, smoothness means that the gradient f' changes slowly. The intuition behind this is that if f' is stable, then it continues to be a good descent direction for some time, and hence we do not need to worry about overshooting the minimizer for an appropriate choice of η. To make this intuition quantitative, we introduce two more definitions.

Definition 2 (Lipschitzness)**.** *For $\mathcal{X} \subseteq \mathbb{R}$ and $L > 0$, we say that $f : \mathcal{X} \to \mathbb{R}$ is* L-Lipschitz *if*

$$|f(x) - f(y)| \le L|x - y| \text{ for all } x, y \in \mathcal{X}.$$

Definition 2 says that if x, y are nearby points, $f(x), f(y)$ are also close. This condition can be used to argue that if we have localized the minimizer $x^\star$ in a small interval, then all points in that interval are approximate minimizers, because their function values are close to $f(x^\star)$. Finally, we can use Lipschitzness to define smoothness, i.e., the property of having a slowly-changing gradient.

Definition 3 (Smoothness)**.** *For $\mathcal{X} \subseteq \mathbb{R}$ and $L > 0$, we say that differentiable $f : \mathcal{X} \to \mathbb{R}$ is* L-smooth *if its derivative f' is L-Lipschitz.*

Gradient access. Why are we allowed to make the gradient descent update (22) at all? In particular, how do we compute $f'(x)$? Although we will not prove this, it turns out that the intuition (3) can be quantified for smooth functions, such that the approximation error is negligible if δ is chosen, say, at machine precision level. Thus, under smoothness it is reasonable to assume that we can access the gradient $f'(x)$, using just two function evaluations, using the formula (3).

3.2 Quadratics

The canonical example of a smooth, convex function is a *quadratic function*, i.e., $q(x) = ax^2 + bx + c$. Analyzing the behavior of gradient descent on quadratics gives us a good understanding of its behavior on general smooth, convex functions, so we begin by fully characterizing this simple case.

We assume that $a > 0$. Observe that all quadratics q with $a > 0$ are convex: $q''(x) = 2a > 0$. Further, recall that the minimizer of a quadratic can be computed by completing the square:

$$\begin{aligned} x^\star &:= \text{argmin}_{x \in \mathbb{R}} \{q(x)\} = \text{argmin}_{x \in \mathbb{R}} \left\{ax^2 + bx + c\right\} \\ &= \text{argmin}_{x \in \mathbb{R}} \left\{a\left(x + \frac{b}{2a}\right)^2 + c - \left(\frac{b}{2a}\right)^2\right\} \\ &= \text{argmin}_{x \in \mathbb{R}} \left\{a\left(x + \frac{b}{2a}\right)^2\right\} = -\frac{b}{2a}. \end{aligned} \tag{23}$$

Note also that the gradient of a quadratic has a clean interpretation, in light of (23):

$$q'(x) = 2ax + b = 2a\left(x - x^\star\right). \tag{24}$$

The gradient (24) is thus directly proportional to the distance from our current point x to the true minimizer, $x^\star$. Moreover, it is straightforward to check that q is $L = 2a$-smooth:

$$|q'(x) - q'(y)| = 2a|(x - x^\star) - (y - x^\star)| = 2a|x - y|.$$

Therefore, for quadratic functions, it is optimal to choose the step size $\eta = \frac{1}{L}$ in the gradient descent update (22), because such a step size will take us to the minimizer in a single step:

$$x - \frac{1}{L}q'(x) = x - \frac{1}{2a}(2a(x - x^\star)) = x - (x - x^\star) = x^\star. \tag{25}$$

More generally, choosing a step size $\eta < \frac{1}{L}$ will cause us to stay on the current side of $x^\star$, whereas choosing $\eta > \frac{1}{L}$ overshoots the target. We can thus view $\eta = \frac{1}{L}$ as a "sweet spot" for quadratics.

We observe that the fact that q is $2a$-smooth, and $q'' = 2a$, is no coincidence. More generally, if f is L-smooth, we can take limits to show that

$$f''(x) = \lim_{y \to x} \frac{f'(y) - f'(x)}{y - x} \le \lim_{y \to x} \frac{|f'(y) - f'(x)|}{|y - x|} \le L. \tag{26}$$

Thus, the second derivative of an L-smooth, convex function is always in the range $[0, L]$.

Finally, we mention one additional interpretation of the progress of gradient descent on quadratics, that extends to more general functions. In particular, we can rewrite the amount of function progress made by the update $x \leftarrow x - \frac{1}{L}q'(x)$ as follows:

$$q(x) - q\left(x - \frac{1}{L}q'(x)\right) = q(x) - q(x^\star) = a\left(x - x^\star\right)^2 = \frac{1}{2L}(q'(x))^2. \tag{27}$$

The first equality used that for quadratics, (25) holds, the second equality used our earlier completion of the square in (24), and the last equality used the gradient calculation in (24). Thus, the amount of progress made by the optimal gradient step is proportional to the squared gradient size. This is intuitive, as larger gradients means there is more progress to be made.

3.3 Smooth functions

In fact, the step size $\eta = \frac{1}{L}$ is an excellent choice for smooth functions beyond quadratics.[2] The main reason for is that smooth functions can be upper bounded by quadratics. This lets us use the analysis in Section 3.2 as a proxy for the progress of gradient descent.

More specifically, just as the Euler approximation in (21) uses a linear function to approximate f, we can obtain sharper and sharper characterizations using higher-degree polynomials. For example, we can approximate f with a degree-2 polynomial, i.e., a quadratic:

$$f(y) \approx f(x) + f'(x)(y-x) + \frac{1}{2}f''(x)(y-x)^2. \tag{28}$$

The reason there is a coefficient of $\frac{1}{2}$ in front of the quadratic term is so that the two sides match when f is exactly a quadratic. Indeed, if we take two derivatives of both sides at $y = x$, we indeed obtain $f''(x) = f''(x)$, because the second derivative of $\frac{1}{2}f''(x)(y-x)^2$ is $f''(x)$.

Intuitively, because $f''(x) \le L$ pointwise, we should expect that in light of (28), the upper bound

$$f(y) \le f(x) + f'(x)(y-x) + \frac{L}{2}(y-x)^2 \text{ for all } x, y \in \mathbb{R} \tag{29}$$

holds. With a bit of calculus, we can show that this is the case even if f is not twice-differentiable.

Lemma 3. *Let $L > 0$, and let $f : \mathbb{R} \to \mathbb{R}$ be L-smooth. Then (29) holds.*

Proof. By the fundamental theorem of calculus,

$$f(y) = f(x) + \int_x^y f'(z)\mathrm{d}z.$$

However, we also have $f'(z) \le f'(x) + L|z-x|$ by smoothness. Thus,

$$\begin{aligned} f(y) &\le f(x) + \int_x^y f'(x)\mathrm{d}z + L\int_x^y |z-x|\mathrm{d}z \\ &= f(x) + f'(x)(y-x) + L\int_x^y |z-x|\mathrm{d}z \\ &= f(x) + f'(x)(y-x) + \frac{L}{2}(z-x)^2\Big|_x^y = f(x) + f'(x)(y-x) + \frac{L}{2}(y-x)^2, \end{aligned}$$

where we can verify by casework on the sign of $y - x$ that the last line holds. □

In summary, (20) and (29) show that for smooth, convex functions,

$$f(x) + f'(x)(y-x) \le f(y) \le f(x) + f'(x)(y-x) + \frac{L}{2}(y-x)^2 \text{ for all } x, y \in \mathbb{R}.$$

In other words, for any choice of centerpoint x, we can extrapolate a linear function and a quadratic function from x that sandwich the function f everywhere.

Critical points. One reasonable termination criterion for an optimization algorithm is to find a point where the gradient is small. We call a point where $|f'(x)| \le \epsilon$ an *ϵ-critical point.* This definition is motivated by the fact for smooth f, $f'(x^\star) = 0$ at the minimizer $x^\star$, so it is reasonable to expect that points with small gradients lie close to $x^\star$ (though formally, only the reverse implication follows from smoothness, because we can only bound gradients by distances, not vice versa).

There is a simple argument showing gradient descent quickly returns an approximate critical point.

[2] One smooth convex function which sees significant use in machine learning is the *logit function*, $f(x) = \log(1 + \exp(x)))$. The derivative of the logit is always in $[0, 1]$, so it can be interpreted as a probability in binary classification, e.g., if we want to label an image, it can represent our estimated likelihood the image is a dog rather than a cat.

Lemma 4. *Let $L, \Delta, \epsilon > 0$, let $f : \mathbb{R} \to \mathbb{R}$ be L-smooth, and let $x^\star := \text{argmin}_{x \in \mathbb{R}} f(x)$. Let $x_0 \in \mathbb{R}$ satisfy $f(x_0) - f(x^\star) \le \Delta$. Then for $T \ge \frac{2L\Delta}{\epsilon^2}$, iterating the update*

$$x_t \leftarrow x_{t-1} - \frac{1}{L} f'(x_{t-1}), \text{ for all } t \in [T],$$

returns iterates $\{x_t\}_{t=0}^T$ such that for at least one iterate, $|f'(x_t)| \le \epsilon$.

Proof. Fix an iteration $t \in [T]$, and let $x := x_t$ for now. Let $q(y) := f(x) + f'(x)(y-x) + \frac{L}{2}(y-x)^2$ be the quadratic upper bound in (29), such that $q(y) \ge f(y)$ for all $y \in \mathbb{R}$. We make two observations regarding q. First, $q'(y) = f'(x) + L(y-x)$, so evaluating this at x, we have that $q'(x) = f'(x)$. Second, q is a quadratic with smoothness $L = q''$, so it is minimized by the update $x \leftarrow x - \frac{1}{L} q'(x)$. Using these facts in conjunction with our earlier calculation (27), we have

$$f\left(x - \frac{1}{L} f'(x)\right) \le q\left(x - \frac{1}{L} q'(x)\right) = q(x) - \frac{1}{2L}(q'(x))^2 = f(x) - \frac{1}{2L}(f'(x))^2. \tag{30}$$

In particular, this shows that for all $t \in [T]$,

$$f(x_t) \le f(x_{t-1}) - \frac{1}{2L}(f'(x_{t-1}))^2.$$

Now suppose it were the case that $|f'(x_t)| > \epsilon$ in every iteration $0 \le t \le T$. Then, we have

$$f(x^\star) \le f(x_T) \le f(x_{T-1}) - \frac{\epsilon^2}{2L} \le f(x_{T-2}) - \frac{2\epsilon^2}{2L} \le \ldots \le f(x_0) - \frac{T\epsilon^2}{2L}.$$

However, this would be a contradiction for $T \ge \frac{2L\Delta}{\epsilon^2}$, because we assumed $f(x_0) - f(x^\star) \le \Delta$. □

Another way of phrasing Lemma 4 is that after T iterations, we will find a point x_t with

$$|f'(x_t)| \le \sqrt{\frac{2L\Delta}{T}}. \tag{31}$$

Remarkably, Lemma 4 never used convexity. This shows gradient descent is a reasonable strategy for finding approximate *local minima* of smooth, but potentially nonconvex, functions. This fact is often cited as an explanation for the empirical performance of gradient descent for training neural networks, whose associated loss functions are generally very complicated, nonconvex functions.

Global minimization. If we additionally have convexity, we can obtain much stronger global optimality guarantees. We require one helper lemma whose proof can be skipped on a first read.

Lemma 5. *Let $L > 0$, let $f : \mathbb{R} \to \mathbb{R}$ be L-smooth and convex, and let $x^\star := \text{argmin}_{x \in \mathbb{R}} f(x)$. Then,*

$$\left|\left(x - \frac{1}{L} f'(x)\right) - x^\star\right| \le |x - x^\star|, \text{ for all } x \in \mathbb{R}.$$

Proof. By squaring both sides and canceling terms, we see that it suffices to prove

$$(x - x^\star)^2 - \frac{2}{L} f'(x)(x - x^\star) + \frac{1}{L^2}(f'(x))^2 \le (x - x^\star)^2$$
$$\iff \frac{1}{2L}(f'(x))^2 \le f'(x)(x - x^\star).$$

However, the latter inequality follows from

$$f'(x)(x - x^\star) \ge f(x) - f(x^\star) \ge f(x) - f\left(x - \frac{1}{L} f'(x)\right) \ge \frac{1}{2L}(f'(x))^2.$$

Here, the first inequality used the convexity lower bound $f(x) + f'(x)(x^\star - x) \le f(x^\star)$, the second used that f is minimized at $x^\star$, and the last used our earlier calculation (30). □

Lemma 5 says steps (22) with $\eta = \frac{1}{L}$ never drift further away from the minimizer. With this fact, we can can prove that gradient descent efficiently *globally minimizes* smooth, convex functions.

Lemma 6. *Let $L > 0$, let $f : \mathbb{R} \to \mathbb{R}$ be L-smooth and convex, and let $x^\star := \text{argmin}_{x\in\mathbb{R}} f(x)$. Let $x_0 \in \mathbb{R}$ satisfy $|x_0 - x^\star| \le R$, and let $\epsilon > 0$. Then for $T \ge \frac{L^2R^4}{\epsilon^2}$, iterating the update*

$$x_t \leftarrow x_{t-1} - \frac{1}{L} f'(x_{t-1}), \text{ for all } t \in [T]$$

returns iterates $\{x_t\}_{t=0}^T$ such that for at least one iterate, $f(x_t) \le f(x^\star) + \epsilon$.

Proof. We first claim that it suffices to choose $\Delta = \frac{LR^2}{2}$ in Lemma 4. Indeed, because $f'(x^\star) = 0$, the upper bound (29) applied at $x \leftarrow x^\star$ and $y \leftarrow x$ shows that

$$f(x_0) \le f(x^\star) + \frac{L}{2}(x_0 - x^\star)^2 \le f(x^\star) + \frac{LR^2}{2}.$$

Hence, after T iterations, Lemma 4 (in particular, (31)) implies we have an iterate x_t satisfying

$$|f'(x_t)| \le \sqrt{\frac{2L\Delta}{T}} \le \sqrt{2 \cdot \frac{L^2R^2}{2} \cdot \frac{\epsilon^2}{L^2R^4}} = \frac{\epsilon}{R}.$$

We conclude by applying convexity, which shows

$$f(x_t) - f(x^\star) \le f'(x_t)(x_t - x^\star) \le |f'(x_t)||x_t - x^\star| \le \epsilon.$$

In the last inequality, we used that repeatedly applying Lemma 5 yields $|x_t - x^\star| \le |x_0 - x^\star| \le R$. □

We remark that because gradient descent makes monotone function progress, i.e., $f(x_t) \le f(x_{t-1})$ for each iteration t (due to (30)), we can upgrade Lemma 6 to read $f(x_T) \le f(x^\star) + \epsilon$.

3.4 ...and beyond

We briefly overview improvements and generalizations of our gradient descent analysis.

Improvements. A tighter analysis of the exact same gradient descent algorithm in Lemma 6 establishes that only $T \ge \frac{LR^2}{\epsilon}$ iterations are needed to obtain an ϵ-approximate minimizer. We refer the reader to Theorem 3, [Tia24b], for a proof of this quadratic factor improvement.

By changing the algorithm, it turns out that it is possible to achieve yet another quadratic factor improvement: $T \ge (\frac{LR^2}{\epsilon})^{1/2}$ iterations suffice for smooth convex optimization [Nes83]. The key idea behind this modified algorithm is the concept of *momentum*: updating using a weighted average of gradients encountered. Intuitively, smoothness causes gradients to change slowly, so we can use the history of our algorithm as additional information to guide our iterates. Momentum is a powerful tool even in nonconvex settings, and indeed, most optimization packages used for training modern machine learning models include a tunable momentum parameter.

High dimensions. As discussed earlier in these notes, and as you will explore in your homework, much less sophisticated strategies than gradient descent (e.g., binary and ternary search) suffice for minimizing appropriately structured one-dimensional functions. However, one appealing aspect of gradient descent is that essentially the entire analysis in Section 3.3 extends to high dimensions. Concretely, if $\mathbf{x} \in \mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$, we can define the high-dimensional gradient

$$\nabla f(\mathbf{x}) := \begin{pmatrix} \frac{\partial f}{\partial \mathbf{x}_1}(\mathbf{x}) & \frac{\partial f}{\partial \mathbf{x}_2}(\mathbf{x}) & \dots & \frac{\partial f}{\partial \mathbf{x}_d}(\mathbf{x}) \end{pmatrix} \in \mathbb{R}^d. \tag{32}$$

This direction can then guide our iterate, via the update $\mathbf{x} \leftarrow \mathbf{x} - \frac{1}{L}\nabla f(\mathbf{x})$. For L-smooth, convex $f : \mathbb{R}^d \to \mathbb{R}$, minimized at $\mathbf{x}^\star$, initializing gradient descent (with momentum) at $\mathbf{x}_0 \in \mathbb{R}^d$ satisfying $\|\mathbf{x}_0 - \mathbf{x}^\star\|_2 \le R$ implies $T \ge (\frac{LR^2}{\epsilon})^{1/2}$ iterations again yields an ϵ-approximate minimizer. Analogs of Lemma 4 for finding critical points of nonconvex functions also hold.

That the quantity T does not grow in the dimension d has been enormously useful in machine learning, where decision variables (corresponding to the parameters of a deep neural network model) live in extremely high-dimensional spaces, e.g., ChatGPT is estimated to have $d \approx 10^{12}$.

4 Principal component analysis

In this section, we give an introduction to *principal component analysis* (PCA), a core algorithm in data analysis and unsupervised machine learning. Intuitively, PCA is a way to take a dataset $\mathbf{A} \in \mathbb{R}^{n \times d}$, thought of as n points in $\mathbb{R}^d$, and return the "most important directions" in the dataset. We provide some motivation for why this is a useful primitive in Section 4.1, followed by some general theory on low-rank approximations in Section 4.2. We conclude in Section 4.3 by overviewing the power method, a simple and efficient way to approximately perform PCA.

4.1 Low-rank approximation: motivation

Consider the following matrix, with some omitted entries.[3] What are the missing entries?

$$\begin{pmatrix} 7 & ? & ? & ? & 21 \\ ? & 8 & 12 & ? & 6 \\ ? & ? & 6 & 2 & ? \end{pmatrix}$$

At first, this problem seems horribly ill-posed – with the information given, these entries could be anything. However, if you encountered such a matrix in a setting where you expect entries to be correlated in some way, you may have a guess what the remaining entries are. Indeed, all observed entries are consistent with the first row being 7× the third row, and the second being 2× the third. This is equivalent to all columns being multiples of $\begin{pmatrix} 7 & 2 & 1 \end{pmatrix}^\top$. The completed matrix is:

$$\begin{pmatrix} 7 & 28 & 42 & 14 & 21 \\ 2 & 8 & 12 & 4 & 6 \\ 1 & 4 & 6 & 2 & 3 \end{pmatrix}.$$

Such a matrix, where all columns are multiples of each other, is called *rank-one*. Another way of phrasing this condition is that a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ is rank-one iff it can be written as $\mathbf{A} = \mathbf{u}\mathbf{v}^\top$, for some $\mathbf{u} \in \mathbb{R}^n$, $\mathbf{v} \in \mathbb{R}^d$. We can verify that for all $j \in [d]$, this means that column $\mathbf{A}_{:j}$ is $\mathbf{v}_j \cdot \mathbf{u}$.

More generally, $\mathbf{A} \in \mathbb{R}^{n \times d}$ is said to be rank-r if there are matrices $\mathbf{U} \in \mathbb{R}^{n \times r}$ and $\mathbf{V} \in \mathbb{R}^{d \times r}$ such that $\mathbf{A} = \mathbf{U}\mathbf{V}^\top$, or equivalently, denoting the columns of $\mathbf{U}, \mathbf{V}$ as $\{\mathbf{u}_k\}_{k \in [r]}$, $\{\mathbf{v}_k\}_{k \in [r]}$ respectively,

$$\mathbf{A} = \mathbf{U}\mathbf{V}^\top = \sum_{k \in [r]} \mathbf{u}_k \mathbf{v}_k^\top. \tag{33}$$

In the regime $r \ll \min(n, d)$, we call a decomposition of the form (33) a *low-rank decomposition*. We focus on this regime, because it is typically the interesting setting; for instance, every matrix has a rank-$\min(n, d)$ decomposition, by taking $\mathbf{U} = \mathbf{A}$ and $\mathbf{V} = \mathbf{I}_d$ (or vice versa if $n \leq d$).

Applications. Why do we care about low-rank matrices? One reason is that such matrices, or approximations thereof, often appear "in the wild." The matrix completion example we used to introduce this section is a well-known example; this task was famously popularized by the Netflix prize [BL07]. In this challenge, participants were given partial information about how a variety of users had rated movies (over 10^8 ratings in total). The goal was to use this information to predict all unknown user ratings. In a toy model where there is an absolute ranking of movie quality, and the only variability in ratings is how much users enjoy movies in general (e.g., Bob gives every movie a rating $\frac{1}{2}$ as large as Alice's), this user-rating matrix is indeed rank-one.

More generally, we can imagine that all enjoyers of action movies have similar preferences, as do all enjoyers of horror movies, etc. In the model (33), where each row corresponds to a Netflix user and each column to a movie, we can let each $k \in [r]$ describe a "type" of movie enjoyer. Then, each column $\mathbf{v}_k \in \mathbb{R}^d$ corresponds to how a purely type-k user would rate each movie $j \in [d]$, and each $\mathbf{u}_k \in \mathbb{R}^n$ specifies how much each user $i \in [n]$ is explained by the type k. By observation, (33) has

$$\mathbf{A}_{i:} = \sum_{k \in [r]} [\mathbf{u}_k]_i \mathbf{v}_k = \sum_{k \in [r]} \mathbf{U}_{ik} \mathbf{V}_{:k}, \tag{34}$$

i.e., it decomposes the ratings $\mathbf{A}_{i:}$ by user i into r explanatory factors. The k^{th} explanatory factor corresponds to the fact that user i is associated a $[\mathbf{u}_k]_i$ amount with the k^{th} type (i.e., the contents of $\mathbf{U}_{ik}$), and that purely type-k users would rate the d movies scaling with the vector $\mathbf{V}_{:k} \in \mathbb{R}^d$.

[3] Thanks to Tim Roughgarden and Greg Valiant for this example.

In this sense, low-rank decompositions are explanatory models. In the Netflix example, under the premise that there are r explanatory factors $\{\mathbf{v}_k\}_{k\in[r]}$ of user ratings, (33) seeks to learn these factors and their associated user-specific coefficients, to decompose each rating $\mathbf{A}_{i:}$ as in (34).

Another famous application of low-rank decompositions are *topic models* in natural language processing. Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ correspond to the word frequencies of n documents, i.e., $\mathbf{A}_{ij}$ is how much word $j \in [d]$ appeared in document i, and d is the dictionary size. In an extreme example where every document randomly samples words according to their distribution in the English language, every row $\mathbf{A}_{i:}$ is roughly a multiple of $\mathbf{f} \in \mathbb{R}^d$, the average English word distribution. More generally, each document could correspond to a mixture of r *topics*, e.g., young adult fiction, biography, cookbooks, and so on, each with their own associated frequency vector $\mathbf{v}_k \in \mathbb{R}^d$. The model (34) hence again uses low-rank decomposition to learn the topics driving document word usage, as well as the amounts of each topic that a document $i \in [n]$ is associated with.

Denoising. Of course, it is unlikely that naturally-occurring matrices are exactly low-rank. This could be due to a variety of reasons: for instance, in the Netflix challenge example, it could be because ratings are only allowed to be integers, whereas the "true" preference vector is non-integer valued. In the topic modeling example, additional error could be introduced due to finite-sample issues: even if words are sampled based on some frequency vector $\mathbf{f}$, for a finite sample of say, 1000 words, it is unlikely that the observed frequencies are exactly a multiple of $\mathbf{f}$.

This motivates *low-rank approximations* (LRAs), the central focus of this section. In the LRA model, our goal is to decompose $\mathbf{A} \in \mathbb{R}^{n\times d}$ as

$$\mathbf{A} = \mathbf{U}\mathbf{V}^\top + \mathbf{N}, \text{ for } \mathbf{U} \in \mathbb{R}^{n\times r},\ \mathbf{V} \in \mathbb{R}^{d\times r}, \text{ and } \mathbf{N} \in \mathbb{R}^{n\times d}. \tag{35}$$

Here, $\mathbf{N}$ models the *noise matrix*, which captures, e.g., discretization issues, sampling error, or other considerations arising from collecting data in practice. The hope of the model (35) is that the noise matrix $\mathbf{N}$ is significantly smaller than the low-rank explanatory factors $\mathbf{U}\mathbf{V}^\top$. We will make the notion of "largeness" of a matrix more precise in Section 4.2.

Computational considerations. Finally, we mention one additional motivation for LRAs, from the perspective of efficient algorithms. As discussed in Part II, Section 5, computing a matrix-vector product $\mathbf{A}\mathbf{v}$ for $\mathbf{v} \in \mathbb{R}^d$ requires $O(nd)$ arithmetic operations in general, as we must compute all n inner products $\mathbf{A}_{i:}^\top\mathbf{v}$. However, if $\mathbf{A}$ is given as an approximate low-rank decomposition $\mathbf{U}\mathbf{V}^\top$ for $\mathbf{U} \in \mathbb{R}^{n\times r}$, $\mathbf{V} \in \mathbb{R}^{d\times r}$, then we can compute $\mathbf{U}\mathbf{V}^\top\mathbf{v} = \mathbf{U}(\mathbf{V}^\top\mathbf{v})$ in just $O((n+d)r)$ arithmetic operations. If $r \ll \min(d,n)$, this could be a significant savings: e.g., if $d \approx n$ and $r = O(1)$, this reduces matrix-vector multiplication costs from $O(d^2)$ to $O(d)$.

4.2 Low-rank approximation: theory

In this section, we discuss theoretical aspects of low-rank approximations. A very fortunate fact of linear algebra is that the optimal LRA is a well-understood object. More precisely, suppose we are given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ that we wish to approximate with a low-rank decomposition $\mathbf{U}\mathbf{V}^\top$, for $\mathbf{U} \in \mathbb{R}^{n\times r}$, $\mathbf{V} \in \mathbb{R}^{d\times r}$. It turns out we have a precise characterization of

$$\mathbf{U}, \mathbf{V} = \text{argmin}_{\mathbf{U}\in\mathbb{R}^{n\times r}, \mathbf{V}\in\mathbb{R}^{d\times r}} \left\|\mathbf{A} - \mathbf{U}\mathbf{V}^\top\right\|, \tag{36}$$

in the entire range $1 \le r \le \min(d,n)$. Here, $\|\cdot\|$ is any *unitarily-invariant norm*. Many reasonable ways of bounding the sizes of matrices, such as the Frobenius norm (entrywise Euclidean norm, i.e., the ℓ_2 norm of a matrix if it is treated as a vector), are unitarily-invariant norms. Other examples of unitarily-invariant norms popularly used in applications are the "operator norm" and "nuclear norm," for which our characterization also applies. Another view of (36) is that our goal is to make the noise matrix $\mathbf{N}$ in (35) as small as possible, according to $\|\cdot\|$. Intuitively, this aligns with our goal of "explaining" as much of $\mathbf{A}$ as possible using the LRA $\mathbf{U}\mathbf{V}^\top$.

Singular value decomposition. Characterizing (36) is entirely in terms of the *singular value decomposition* (SVD) of $\mathbf{A}$. The SVD is described in Fact 4, Part I, but we recall it here.

First, we remind the reader that $\mathbf{U} \in \mathbb{R}^{d\times d}$ is called *orthonormal* if $\mathbf{U}^\top\mathbf{U} = \mathbf{I}_d$, where $\mathbf{I}_d$ is the $d \times d$ identity matrix, i.e., has ones along the diagonal and zeroes elsewhere. Observe that if we let

$\{\mathbf{u}_i\}_{i\in[d]}$ be the columns of $\mathbf{U}$, the $(i,j)^{\text{th}}$ entry of $\mathbf{U}^\top\mathbf{U}$ is simply $\mathbf{u}_i^\top\mathbf{u}_j$. Thus $\mathbf{U}^\top\mathbf{U} = \mathbf{I}_d$ implies

$$\mathbf{u}_i^\top\mathbf{u}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \text{ for all } (i,j) \in [d] \times [d]. \tag{37}$$

This is naturally interpreted as all of the $\mathbf{u}_i$ having unit length, and each pair of unequal $(\mathbf{u}_i, \mathbf{u}_j)$ being perpendicular. Such a set $\{\mathbf{u}_i\}_{i\in[d]}$ is called an *orthonormal basis.* The most famous orthonormal basis is the *standard basis* $\{\mathbf{e}_i\}_{i\in[d]}$, where $\mathbf{e}_i$ has a one in the i^{th} position and zeroes elsewhere. The orthonormal matrix associated with the standard basis is $\mathbf{U} = \mathbf{I}_d$. More generally, $\mathbf{U}$ is a *rotation matrix* that transforms the standard basis into the basis given by $\{\mathbf{u}_i\}_{i\in[d]}$. A useful fact is that if $\mathbf{U}^\top\mathbf{U} = \mathbf{I}_d$, then $\mathbf{U}\mathbf{U}^\top = \mathbf{I}_d$ also, so one can make sense of "inverse rotations."

It is also possible for a rectangular matrix $\mathbf{U} \in \mathbb{R}^{n\times d}$ to be orthonormal, i.e., satisfy $\mathbf{U}^\top\mathbf{U} = \mathbf{I}_d$, if $n \geq d$. This can be interpreted as the columns $\{\mathbf{u}_i\}_{i\in[d]} \subset \mathbb{R}^n$ each having unit norm and being pairwise perpendicular. These vectors form a basis for a linear subspace, the subspace $\text{Span}(\mathbf{U}) \subseteq \mathbb{R}^n$ (for a review, see Section 5.2, Part I). As a simple example, if $\mathbf{U}$ is some subset of the columns of $\mathbf{I}_n$, i.e., some subset $\{\mathbf{e}_i\}_{i\in S}$ for $S \subseteq [n]$, then it is straightforward to verify $\mathbf{U}^\top\mathbf{U} = \mathbf{I}_d$. In this case, $\text{Span}(\mathbf{U})$ is simply all vectors whose nonzero coordinates are in S.

Now we are ready to define the singular value decomposition. We focus on the case of $\mathbf{A} \in \mathbb{R}^{n\times d}$ with $n \geq d$, i.e., "tall" matrices, because otherwise we can apply Fact 1 to $\mathbf{A}^\top$.

Fact 1 (Singular value decomposition)**.** *Let* $\mathbf{A} \in \mathbb{R}^{n\times d}$ *with* $n \geq d$. *There exist orthonormal* $\mathbf{U} \in \mathbb{R}^{n\times d}$, $\mathbf{V} \in \mathbb{R}^{d\times d}$, *and diagonal* $\mathbf{\Sigma} = \mathbf{diag}\left(\boldsymbol{\sigma}\right) \in \mathbb{R}^{d\times d}_{\geq 0}$, *for* singular values $\boldsymbol{\sigma} \in \mathbb{R}^d_{\geq 0}$, *such that*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{i\in[d]} \boldsymbol{\sigma}_i \mathbf{u}_i \mathbf{v}_i^\top, \tag{38}$$

where $\{\mathbf{u}_i\}_{i\in[d]} \subseteq \mathbb{R}^n$ *are columns of* $\mathbf{U}$ *and* $\{\mathbf{v}_i\}_{i\in[d]} \subseteq \mathbb{R}^d$ *are columns of* $\mathbf{V}$.

The SVD has a very intuitive interpretation. Suppose for now that $\mathbf{V} = \mathbf{I}_d$. Then, any vector $\mathbf{x} \in \mathbb{R}^d$ is transformed through multiplication by $\mathbf{A}$ as follows:

$$\mathbf{x} = \sum_{i\in[d]} \mathbf{x}_i \mathbf{e}_i \implies \mathbf{A}\mathbf{x} = \mathbf{U}\mathbf{\Sigma}\mathbf{x} = \sum_{i\in[d]} (\boldsymbol{\sigma}_i \mathbf{x}_i)\mathbf{u}_i.$$

In other words, $\mathbf{x}_i$ units of the standard basis vector $\mathbf{e}_i \in \mathbb{R}^n$ are transformed into $\boldsymbol{\sigma}_i\mathbf{x}_i$ units of the basis vector $\mathbf{u}_i \in \mathbb{R}^d$. In the general case of arbitrary orthonormal $\mathbf{V}$, we can again consider $\mathbf{x} = \mathbf{V}\mathbf{c}$ for some coefficient vector $\mathbf{c}$, that puts $\mathbf{x}$ in the basis given by $\mathbf{V}$'s columns. Then,

$$\mathbf{x} = \sum_{i\in[d]} \mathbf{c}_i \mathbf{v}_i \implies \mathbf{A}\mathbf{x} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{V}\mathbf{c} = \sum_{i\in[d]} (\boldsymbol{\sigma}_i \mathbf{c}_i)\mathbf{u}_i. \tag{39}$$

That is, again the action of multiplication by $\mathbf{A} \in \mathbb{R}^{n\times d}$ transforms each unit of $\mathbf{v}_i$ into $\boldsymbol{\sigma}_i$ units of $\mathbf{u}_i$. This gives a geometric interpretation of how this multiplication transforms $\mathbb{R}^d$ into $\mathbb{R}^n$.

There also is a natural interpretation of the *sparsity* of $\boldsymbol{\sigma}$, i.e., the number of singular values that are nonzero. This turns out to just be the *rank* of $\mathbf{A}$, which we recall is the maximum number of linearly independent columns of $\mathbf{A}$, a.k.a. the dimension of $\text{Span}(\mathbf{A})$ (again, see Section 5.2, Part I). Intuitively, any zero singular value $\boldsymbol{\sigma}_i$ makes it so that $\mathbf{u}_i$ cannot appear in the output $\mathbf{A}\mathbf{x}$, effectively projecting out $\mathbf{v}_i$ from $\mathbf{x}$ and reducing the dimension of the output space by one.

Now we are ready to explain the solution to the LRA problem (36), due to [EY36, Mir60].

Fact 2 (Eckart-Young-Mirsky)**.** *Let* $\mathbf{A} \in \mathbb{R}^{n\times d}$ *with* $n \geq d$ *have SVD* $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, *and suppose without loss of generality that* $\mathbf{\Sigma} = \mathbf{diag}\left(\boldsymbol{\sigma}\right)$ *is sorted so that* $\boldsymbol{\sigma}_1 \geq \boldsymbol{\sigma}_2 \geq \ldots \geq \boldsymbol{\sigma}_d$.

For $r \in [d]$, *let* $\mathbf{U}_{[r]:} \in \mathbb{R}^{n\times r}$ *and* $\mathbf{V}_{[r]:} \in \mathbb{R}^{d\times r}$ *consist of the first* r *columns of* $\mathbf{U}$ *and* $\mathbf{V}$ *respectively. For any unitarily invariant norm* $\|\cdot\|$, *the optimal rank-*r *LRA in* (36) *is*

$$\mathbf{U}_{[r]:}\mathbf{diag}\left(\boldsymbol{\sigma}_{[r]}\right)\mathbf{V}_{[r]:}^\top = \sum_{k\in[r]} \boldsymbol{\sigma}_k \mathbf{u}_k \mathbf{v}_k^\top.$$

Thus, the optimal LRA of rank r keeps only the r components with the largest singular values in the SVD. Fact 1 is often used to justify the use of SVD to compute the optimal explanatory factors in LRAs (35). The rest of the section discusses computational issues concerning the SVD.

Principal component analysis. Algorithms for SVD are better explained via their symmetric counterpart, *principal component analysis* (PCA). When $\mathbf{M} \in \mathbb{R}^{d\times d}$ is symmetric, i.e., $\mathbf{M}_{ij} = \mathbf{M}_{ji}$ for all $(i,j) \in [d] \times [d]$, the *spectral theorem* (Fact 3, Part I) says that for some diagonal $\boldsymbol{\Lambda} = \mathbf{diag}\left(\boldsymbol{\lambda}\right) \in \mathbb{R}^{d\times d}$ and orthonormal $\mathbf{U} \in \mathbb{R}^{d\times d}$ with columns $\{\mathbf{u}_i\}_{i\in[d]}$, the *eigendecomposition*

$$\mathbf{M} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top = \sum_{i\in[d]} \boldsymbol{\lambda}_i \mathbf{u}_i \mathbf{u}_i^\top \tag{40}$$

holds. Here $\boldsymbol{\lambda}$ are the eigenvalues of $\mathbf{A}$. To reconcile this with Fact 1, let $\boldsymbol{\sigma} = |\boldsymbol{\lambda}|$ applied entrywise. Then $\mathbf{V}$ has columns equal to μ's columns, except possibly negated in any column $i \in [d]$ where $\boldsymbol{\sigma}_i = \boldsymbol{\lambda}_i$. This is to ensure that the singular values $\boldsymbol{\sigma}$ are nonnegative as in Fact 1.

There is a particularly clean interpretation of everything we have discussed thus far when the eigenvalues $\boldsymbol{\lambda}$ of a symmetric $\mathbf{M} \in \mathbb{R}^{d\times d}$ are already all nonnegative. In this case, the eigendecomposition (40) exactly matches the SVD (38). In light of (39), consider the case where $\mathbf{U} = \mathbf{I}_d$. In this case, multiplication by $\mathbf{M} = \boldsymbol{\Lambda}$ amounts to scaling the i^{th} standard basis axis by $\boldsymbol{\lambda}_i$. More generally, (39) shows multiplication by $\mathbf{M}$ performs this axis rescaling in the basis given by $\mathbf{U}$.

Symmetric matrices $\mathbf{M} \in \mathbb{R}^{d\times d}$ for which $\boldsymbol{\Lambda} = \mathbf{diag}\left(\boldsymbol{\lambda}\right) \in \mathbb{R}^{d\times d}_{\geq 0}$ in (40) are called *positive semidefinite* (PSD). Every PSD can visually be associated with an ellipsoid centered at the origin in $\mathbb{R}^{d\times d}$, with its d principal axes given by columns of $\mathbf{U}$, and their associated radii given by $\boldsymbol{\lambda}$.

PSD matrices are much easier to analyze and work with algorithmically than general matrices. Moreover, the SVD reduces to computing the eigendecomposition (40) of PSD matrices:

$$\begin{aligned}
\mathbf{A}^\top\mathbf{A} &= \left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\right)^\top \left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}\right) = \mathbf{V}\boldsymbol{\Sigma}\underbrace{\left(\mathbf{U}^\top\mathbf{U}\right)}_{=\mathbf{I}_d}\boldsymbol{\Sigma}\mathbf{V} = \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^\top, \\
\mathbf{A}\mathbf{A}^\top &= \left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\right)\left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\right)^\top = \mathbf{U}\boldsymbol{\Sigma}\underbrace{\left(\mathbf{V}^\top\mathbf{V}\right)}_{=\mathbf{I}_d}\boldsymbol{\Sigma}\mathbf{U}^\top = \mathbf{U}\boldsymbol{\Sigma}^2\mathbf{U}^\top.
\end{aligned} \tag{41}$$

Thus the right singular vectors $\mathbf{V}$ of $\mathbf{A}$ are simply eigenvectors of the PSD matrix $\mathbf{A}^\top\mathbf{A}$, and similarly the left singular vectors $\mathbf{U}$ are eigenvectors of PSD $\mathbf{A}\mathbf{A}^\top$. In Section 4.3, we hence focus on computing the top r eigenvectors of a PSD matrix $\mathbf{M} \in \mathbb{R}^{d\times d}$. This task is known as the r-principal component analysis (r-PCA) problem, and directly gives optimal LRAs.

4.3 Power method

Let $\mathbf{M} \in \mathbb{R}^{d\times d}$ be PSD, and suppose (40) holds, where $\boldsymbol{\lambda}_1 \geq \boldsymbol{\lambda}_2 \geq \ldots \boldsymbol{\lambda}_d \geq 0$ without loss of generality. We focus on approximating $\mathbf{u}_1$, the *top eigenvector* of $\mathbf{M}$, and further, we assume $\boldsymbol{\lambda}_1 \geq (1+\gamma)\boldsymbol{\lambda}_2$, where $\gamma > 0$ is referred to as the *gap*. Intuitively, this setting is simpler because the top eigenvector is more "obvious" compared to all other eigenvectors, by a factor of $\geq 1+\gamma$.

Interestingly, the techniques in this section extend to the case of recovering the top r eigenvectors, and the *gap-free* setting (where we do not assume $\gamma > 0$), once we have the right definitions in place. This is beyond the scope of these notes, and we refer the reader to [MM15] for such topics.

We first give an analysis of a simple algorithm in an extreme case where γ is very large. We need two helper facts about random Gaussian vectors which we state without proof.

Fact 3. *Let $\mathbf{g} \in \mathbb{R}^d$ have each coordinate $\mathbf{g}_i$ drawn independently from a standard Gaussian (normal) distribution $\mathcal{N}(0,1)$. Then with probability ≥ 0.99, $|\mathbf{g}_i| \leq 200d|\mathbf{g}_1|$ for all $i \in [d] \setminus \{1\}$.*

Fact 4. *Let $\mathbf{h} = \sum_{i\in[d]} \mathbf{g}_i\mathbf{u}_i \in \mathbb{R}^d$, for $\mathbf{g}$ is as in Fact 3 and orthonormal $\mathbf{U} \in \mathbb{R}^{d\times d}$. Then $\mathbf{h}$ and $\mathbf{g}$ are distributionally identical, i.e., we can sample $\mathbf{h}$ by drawing $\mathbf{g}$ as in Fact 3 and setting $\mathbf{h} \leftarrow \mathbf{g}$.*

Intuitively, Fact 3 is because Gaussians are flat near 0, so it is reasonably likely that $|\mathbf{g}_1| \geq \frac{1}{200}$. Further, Gaussians concentrate, so it is extremely likely that all other coordinate magnitudes are at most d. We will see tools needed to understand this statement better in Part VII of the notes.

Fact 4 is a bit more complicated, but follows from the fact that Gaussian densities are $\propto \exp(-\frac{1}{2}x^2)$, so in high dimensions this is $\exp(-\frac{1}{2}\sum_{i\in[d]}\mathbf{x}_i^2) = \exp(-\frac{1}{2}\|\mathbf{x}\|_2^2)$. Multiplication through $\mathbf{U}$ does not change this density, because

$$\|\mathbf{U}\mathbf{x}\|_2^2 = (\mathbf{U}\mathbf{x})^\top \mathbf{U}\mathbf{x} = \mathbf{x}^\top \left(\mathbf{U}^\top\mathbf{U}\right)\mathbf{x} = \mathbf{x}^\top\mathbf{x} = \|\mathbf{x}\|_2^2.$$

Finally we are ready to state and analyze the one-step power method.

Lemma 7 (One-step power method)**.** *Let $\boldsymbol{\lambda}$ have nondecreasing coordinates in (40) and let $\boldsymbol{\lambda}_1 > (1+\gamma)\boldsymbol{\lambda}_2$ for $\gamma > 2000d^2$. Let $\mathbf{h} = \sum_{i\in[d]}\mathbf{g}_i\mathbf{u}_i$ where $|\mathbf{g}_i| \le 200d|\mathbf{g}_1|$ for all $i \in [d]\setminus\{1\}$. Then,*

$$\frac{|(\mathbf{M}\mathbf{h})^\top\mathbf{u}_1|}{\|\mathbf{M}\mathbf{h}\|_2\,\|\mathbf{u}_1\|_2} \ge 0.99.$$

Proof. First, observe that by orthonormality (37),

$$\mathbf{M}\mathbf{h} = \left(\sum_{i\in[d]}\boldsymbol{\lambda}_i\mathbf{u}_i\mathbf{u}_i^\top\right)\left(\sum_{i\in[d]}\mathbf{g}_i\mathbf{u}_i\right) = \sum_{i\in[d]}(\boldsymbol{\lambda}_i\mathbf{g}_i)\mathbf{u}_i.$$

Therefore, $|(\mathbf{M}\mathbf{h})^\top\mathbf{u}_1| = |\mathbf{g}_1\boldsymbol{\lambda}_1|$ and hence from the given assumptions,

$$|(\mathbf{M}\mathbf{h})^\top\mathbf{u}_1| = |\mathbf{g}_1\boldsymbol{\lambda}_1| > 10d|\mathbf{g}_i\boldsymbol{\lambda}_i| = 10d|(\mathbf{M}\mathbf{h})^\top\mathbf{u}_i|, \text{ for all } i \in [d]\setminus\{1\}.$$

However, we also have

$$\|\mathbf{M}\mathbf{h}\|_2^2 = \sum_{i\in[d]}((\mathbf{M}\mathbf{h})^\top\mathbf{u}_i)^2 = \sum_{i\in[d]}(\mathbf{g}_i\boldsymbol{\lambda}_i)^2 \le (\mathbf{g}_1\boldsymbol{\lambda}_1)^2 + \frac{1}{100d^2}\sum_{i\in[d]\setminus\{1\}}(\mathbf{g}_1\boldsymbol{\lambda}_1)^2 \le 1.01(\mathbf{g}_1\boldsymbol{\lambda}_1)^2.$$

Putting together the above displays with $\|\mathbf{u}_1\|_2 = 1$, we have the claim:

$$\frac{|(\mathbf{M}\mathbf{h})^\top\mathbf{u}_1|}{\|\mathbf{M}\mathbf{h}\|_2\,\|\mathbf{u}_1\|_2} = \frac{|\mathbf{g}_1\boldsymbol{\lambda}_1|}{\sqrt{\sum_{i\in[d]}(\mathbf{g}_i\boldsymbol{\lambda}_i)^2}} \ge \frac{1}{1.01} \ge 0.99.$$

□

Lemma 7 has an extremely strong guarantee. It says that the vector $\mathbf{M}\mathbf{h}$ has a correlation of at least 0.99 in the *cosine distance* $\cos(\theta(\mathbf{M}\mathbf{h},\mathbf{u}_1))$, where $\theta(\mathbf{M}\mathbf{h},\mathbf{u}_1)$ is the angle between the two given vectors after normalization. The largest correlation is $1 = \cos(0)$, so a cosine distance of 0.99 is almost the best possible. However, Lemma 7 seems to be almost useless: it requires a gap parameter of at least $\gamma > 2000d^2$. This is very unlikely to be the case in applications.

Nonetheless, even in more reasonable regimes, e.g., $\gamma \approx 0.1$, a simple trick lets us use Lemma 7: exponentiation. In particular, for $\gamma \in (0,1)$, let

$$p = \left\lceil \frac{\log_2(2000d^2)}{\gamma} \right\rceil \implies (1+\gamma)^p \ge 2^{\log_2(2000d^2)} = 2000d^2. \tag{42}$$

The reason why this is relevant is because $\mathbf{M}^p$ is a matrix with the exact same eigenvectors as $\mathbf{M}$, but its eigenvalues are $\boldsymbol{\lambda}^p$ where exponentiation is entrywise. This is most easily seen when $p = 2$:

$$\mathbf{M}^2 = (\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top)(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top) = \mathbf{U}\boldsymbol{\Lambda}(\mathbf{U}^\top\mathbf{U})\boldsymbol{\Lambda}\mathbf{U}^\top = \mathbf{U}\boldsymbol{\Lambda}^2\mathbf{U}^\top.$$

Thus, to recover the top eigenvector $\mathbf{u}_1$ to cosine distance 0.99, it is enough to apply Lemma 7 to the matrix $\mathbf{M}^p$, which in light of the calculation (42) has eigenvalues that actually do satisfy the enormous gap assumption in Lemma 7! Moreover, we can compute $\mathbf{M}^p\mathbf{h}$ for a $\mathbf{h}$ meeting the assumptions in Lemma 7 with probability ≥ 0.99, in time

$$O\left(d^2p\right) = O\left(\frac{d^2\log(d)}{\gamma}\right),$$

which is nearly-linear in d^2 (the size of $\mathbf{M}$), as long as γ is not too small. To do so, we draw $\mathbf{h}$ with independent normal coordinates (see Facts 3, 4) and multiply by $\mathbf{M}$, p times in total.

This is an early demonstration of the surprising power of randomness in designing efficient algorithms, a theme we will explore in much more depth in Part VII of the notes.

5 High-dimensional continuous optimization

We conclude these notes with some anecdotes on continuous optimization in high dimensions. Our goal is to unify many of the disparate ideas in Sections 2, 3, and 4 under a common lens.

5.1 Convexity in high dimensions

Just as convexity in one dimension enabled binary search and gradient descent to find approximate global minima, there is a related notion of high-dimensional convexity enabling efficient algorithms.

Definition 4 (Convexity in $\mathbb{R}^d$). *We say that a set $\mathcal{X} \subseteq \mathbb{R}^d$ is convex if for any two $\mathbf{x}, \mathbf{y} \in \mathcal{X}$,*

$$(1-\lambda)\mathbf{x} + \lambda\mathbf{y} \in \mathcal{X}, \text{ for all } \lambda \in [0,1]. \tag{43}$$

We say that a function $f : \mathcal{X} \to \mathbb{R}$ is convex if $\mathcal{X}$ is convex, and for any two $\mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$f((1-\lambda)\mathbf{x} + \lambda\mathbf{y}) \le (1-\lambda)f(\mathbf{x}) + \lambda f(\mathbf{y}), \text{ for all } \lambda \in [0,1]. \tag{44}$$

Comparing Definitions 1 and 4, the difference is simply that Definition 4 applies to high-dimensional vectors, and allows for restriction of the decision variable $\mathbf{x}$ by a convex *constraint set* $\mathcal{X}$. According to (43), we call a set $\mathcal{X}$ convex if the line between any two points in $\mathcal{X}$ never leaves the set. Intuitively, the definition (44) allows us to make function value progress by taking weighted averages of points. The role of (43) is to enforce that these weighted averages stay in the set.

We have already seen two examples of functions satisfying Definition 4. The simplest is the subject of Section 3, where we had $\mathcal{X} = \mathbb{R}$ and $d = 1$, i.e., we were performing unconstrained one-dimensional convex optimization. Perhaps less obviously, linear programs, the subject of Section 2, also satisfy Definition 4. Linear objective functions $f(\mathbf{x}) = \mathbf{c}^\top\mathbf{x}$ satisfy (44) with equality:

$$\mathbf{c}^\top((1-\lambda)\mathbf{x} + \lambda\mathbf{y}) = \sum_{j\in[d]} \mathbf{c}_j((1-\lambda)\mathbf{x}_j + \lambda\mathbf{y}_j) = (1-\lambda)\mathbf{c}^\top\mathbf{x} + \lambda\mathbf{c}^\top\mathbf{y}, \text{ for all } \lambda \in [0,1].$$

Moreover, if two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ satisfy $\mathbf{A}\mathbf{x} \le \mathbf{b}$ and $\mathbf{A}\mathbf{y} \le \mathbf{b}$ for $\mathbf{A} \in \mathbb{R}^{n\times d}$, then for all $i \in [n]$,

$$[\mathbf{A}((1-\lambda)\mathbf{x} + \lambda\mathbf{y})]_i = (1-\lambda)\mathbf{A}_{i:}^\top\mathbf{x} + \lambda\mathbf{A}_{i:}^\top\mathbf{y} \le (1-\lambda)\mathbf{b}_i + \lambda\mathbf{b}_i = \mathbf{b}_i, \text{ for all } \lambda \in [0,1],$$

i.e., all weighted averages $(1-\lambda)\mathbf{x} + \lambda\mathbf{y}$ also lie in the polytope $\mathcal{X} := \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{A}\mathbf{x} \le \mathbf{b}\}$.

Convexity and PSD matrices. Interestingly, there is a close connection between Definition 1 and our definition of PSD matrices in Section 4.2. Recall that a symmetric matrix $\mathbf{M} \in \mathbb{R}^{d\times d}$ is PSD if, in the eigendecomposition (40), all eigenvalues are nonnegative, i.e., $\boldsymbol{\lambda}_i \ge 0$ for all $i \in [d]$. There is another definition of being PSD that is somewhat easier to apply in applications.

Lemma 8. *Symmetric matrix $\mathbf{M} \in \mathbb{R}^{d\times d}$ is PSD iff $\mathbf{v}^\top\mathbf{M}\mathbf{v} \ge 0$ for all $\mathbf{v} \in \mathbb{R}^d$.*

Proof. Recall from (40) that all symmetric $\mathbf{M} \in \mathbb{R}^{d\times d}$ can be written as $\mathbf{M} = \sum_{j\in[d]} \boldsymbol{\lambda}_j\mathbf{u}_j\mathbf{u}_j^\top$, for some set of orthonormal $\{\mathbf{u}_j\}_{j\in[d]} \subset \mathbb{R}^d$ and eigenvalues $\boldsymbol{\lambda} \in \mathbb{R}^d$, not necessarily nonnegative. Our goal is to argue that $\boldsymbol{\lambda} \in \mathbb{R}^d_{\ge 0}$, i.e., all eigenvalues are nonnegative, iff $\mathbf{v}^\top\mathbf{M}\mathbf{v} \ge 0$ for all $\mathbf{v} \in \mathbb{R}^d$.

First, we give an interpretation of the expression $\mathbf{v}^\top\mathbf{M}\mathbf{v}$ when $\mathbf{v} = \mathbf{u}_i$ is an eigenvector of $\mathbf{M}$. Using the above eigendecomposition, and the facts that $\mathbf{u}_i^\top\mathbf{u}_i = 1$ and $\mathbf{u}_j^\top\mathbf{u}_j = 0$ for $j \ne i$,

$$\mathbf{u}_i^\top\mathbf{M}\mathbf{u}_i = \mathbf{u}_i^\top\left(\sum_{j\in[d]} \boldsymbol{\lambda}_j\mathbf{u}_j\mathbf{u}_j^\top\right)\mathbf{u}_i = \mathbf{u}_i^\top\left(\sum_{j\in[d]} \boldsymbol{\lambda}_j\mathbf{u}_j(\mathbf{u}_j^\top\mathbf{u}_i)\right) = \mathbf{u}_i^\top(\boldsymbol{\lambda}_i\mathbf{u}_i) = \boldsymbol{\lambda}_i.$$

In other words, when $\mathbf{v}$ is an eigenvector, $\mathbf{v}^\top\mathbf{M}\mathbf{v}$ is just the corresponding eigenvalue.

Now one direction of the proof is straightforward. If $\mathbf{M}$ is not PSD, then let $\boldsymbol{\lambda}_i$ be any negative eigenvalue. Choosing $\mathbf{v} = \mathbf{u}_i$ shows that it cannot be the case that $\mathbf{v}^\top\mathbf{M}\mathbf{v} \ge 0$ for all $\mathbf{v} \in \mathbb{R}^d$, because $\mathbf{u}_i^\top\mathbf{M}\mathbf{u}_i = \boldsymbol{\lambda}_i < 0$. The contrapositive is that if $\mathbf{v}^\top\mathbf{M}\mathbf{v} \ge 0$ for all $\mathbf{v} \in \mathbb{R}^d$, then $\mathbf{M}$ is PSD.

The other direction of the proof is essentially a direct expansion. If $\mathbf{M}$ is PSD, i.e., $\boldsymbol{\lambda} \in \mathbb{R}^{\geq 0}$,

$$\mathbf{v}^\top \mathbf{M} \mathbf{v} = \mathbf{v}^\top \left(\sum_{j \in [d]} \boldsymbol{\lambda}_j \mathbf{u}_j \mathbf{u}_j^\top \right) \mathbf{v} = \sum_{j \in [d]} \boldsymbol{\lambda}_j (\mathbf{u}_j^\top \mathbf{v})^2 \geq 0.$$

□

We can now connect convexity and PSD matrices. An interpretation of Definition 4 is that for any direction $\mathbf{v} \in \mathbb{R}^d$ and "centerpoint" $\bar{x} \in \mathbb{R}^d$, the one-dimensional restriction

$$f_{1\mathrm{d}}(x) := f\left(\bar{\mathbf{x}} + x\mathbf{v}\right)$$

is convex. Pictorially, if we force $\mathbf{x} \in \mathcal{X}$ to lie on some fixed line passing through $\bar{x}$ in the direction $\mathbf{v}$, $f_{1\mathrm{d}}$ is the resulting restriction along this line, where $f_{1\mathrm{d}}(0) = f(\bar{\mathbf{x}})$.

Now we can use our understanding of convexity in one dimension to interpret Definition 4. We know at least two things about $f_{1\mathrm{d}}$, assuming it is twice-differentiable for now. The first is that the Euler approximation (20) holds, i.e., $f_{1\mathrm{d}}(y) \geq f_{1\mathrm{d}}(x) + f'_{1\mathrm{d}}(x)(y - x)$ for all $x, y \in \mathbb{R}$. The second is that the derivative grows left-to-right, i.e., $f''_{1\mathrm{d}}(x) \geq 0$ for all $x \in \mathbb{R}$.

What do these conditions mean in high dimensions? We first need to understand what the first and second derivatives of $f_{1\mathrm{d}}$ are. By the multivariate chain rule, letting $\mathbf{x} := \bar{\mathbf{x}} + x\mathbf{v}$ be the high-dimensional point $f_{1\mathrm{d}}(x)$ is evaluated at, and recalling our definition of $\nabla f(\mathbf{x})$ in (32),

$$\frac{\mathrm{d}}{\mathrm{d}x} f_{1\mathrm{d}}(x) = \frac{\mathrm{d}}{\mathrm{d}x} f(\bar{\mathbf{x}} + x\mathbf{v}) = \sum_{i \in [d]} \frac{\partial}{\partial \mathbf{x}_i} f(\mathbf{x}) \cdot \mathbf{v}_i = \nabla f(\mathbf{x})^\top \mathbf{v}.$$

Completely analogously, we can write the second derivative as

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2} f_{1\mathrm{d}}(x) = \mathbf{v}^\top \nabla^2 f(\mathbf{x}) \mathbf{v}.$$

Here, $\nabla^2 f(\mathbf{x})$ is a symmetric, $d \times d$ matrix known as the *Hessian.* Just as the gradient generalizes first derivatives to high dimensions, the Hessian generalizes second derivatives. The $(i,j)^{\text{th}}$ entry of the Hessian matrix $\nabla^2 f(\mathbf{x})$ is the second partial derivative $\frac{\partial^2}{\partial \mathbf{x}_i \partial \mathbf{x}_j} f(\mathbf{x})$, for all $(i,j) \in [d] \times [d]$.

Now we can explain how to generalize our first and second derivative characterizations of convexity in $\mathbb{R}$. Regarding the Euler approximation underestimate (20), the analog in $\mathbb{R}^d$ is that

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}), \text{ for all } \mathbf{x}, \mathbf{y} \in \mathcal{X}. \tag{45}$$

Pictorially, what (45) means is that if we project $\mathbf{x}$ and $\mathbf{y}$ in the direction given by $\nabla f(\mathbf{x}) \in \mathbb{R}^d$, then the linear extrapolation of $f(\mathbf{x})$ along this line underestimates $f(\mathbf{y})$.

Similarly, regarding the second derivative bound $f''_{1\mathrm{d}}(x) \geq 0$, the high-dimensional analog is

$$\mathbf{v}^\top \nabla^2 f(\mathbf{x}) \mathbf{v} \geq 0 \text{ for all } \mathbf{v} \in \mathbb{R}^d \text{ and } \mathbf{x} \in \mathcal{X}. \tag{46}$$

In other words, second derivatives of f restricted to any one-dimensional line are all nonnegative. Notice that we already have an equivalent way of stating the above condition: by Lemma 8, it is just saying that $\nabla^2 f(\mathbf{x})$ is PSD for all $\mathbf{x} \in \mathcal{X}$. We are therefore able to throw our entire PCA toolbox from Section 4 at characterizing and manipulating high-dimensional convex functions.

5.2 Linear regression

Let $\mathcal{X} := \mathbb{R}^d$ for simplicity. To motivate our development in this section, the condition (46) is a bit hard to interpret, because we have not even given any geometric intuition on what the Hessian $\nabla^2 f(\mathbf{x})$ looks like. However, the condition is particularly simple when the Hessian $\nabla^2 f(\mathbf{x})$ is a constant. This is the case for quadratics of the form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{M} \mathbf{x} + \mathbf{v}^\top \mathbf{x}, \text{ for symmetric } \mathbf{M} \in \mathbb{R}^{d \times d}, \ \mathbf{v} \in \mathbb{R}^d. \tag{47}$$

The quadratic function (47) should be thought of as the high-dimensional generalization of $f(x) = ax^2 + bx + c$. We omitted the constant term $+c$ as it does not change where the minimizer of f lies, and we multiplied $\mathbf{M}$ (the proxy for the quadratic coefficient a) by a factor of $\frac{1}{2}$. This is for convenience in the following derivative calculations involving f.

We claim that the gradient of f, $\nabla f(\mathbf{x})$, is $\mathbf{Mx} + \mathbf{v}$. To see this, it is helpful to rewrite

$$f(\mathbf{x}) = \frac{1}{2} \sum_{(i,j)\in[d]\times[d]} \mathbf{M}_{ij}\mathbf{x}_i\mathbf{x}_j + \sum_{i\in[d]} \mathbf{v}_i\mathbf{x}_i,$$

which follows by directly expanding the definition (47). Then, the i^{th} coordinate of $\nabla f(\mathbf{x})$ is

$$\frac{\partial}{\partial \mathbf{x}_i} f(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}_i}\left(\frac{1}{2}\mathbf{M}_{ii}\mathbf{x}_i^2 + \sum_{\substack{j\in[d]\\ j\neq i}} \mathbf{M}_{ij}\mathbf{x}_i\mathbf{x}_j\right) + \mathbf{v}_i = \sum_{j\in[d]} \mathbf{M}_{ij}\mathbf{x}_j + \mathbf{v}_i = [\mathbf{Mx}+\mathbf{v}]_i,$$

as claimed, where we dropped all terms in $\mathbf{x}^\top\mathbf{Mx}$ that do not involve $\mathbf{x}_i$. We similarly have

$$\frac{\partial^2}{\partial \mathbf{x}_i \partial \mathbf{x}_j} f(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}_j}\left(\frac{\partial}{\partial \mathbf{x}_i} f(\mathbf{x})\right) = \frac{\partial}{\partial \mathbf{x}_j}\left(\sum_{k\in[d]} \mathbf{M}_{ik}\mathbf{x}_k + \mathbf{v}_i\right) = \mathbf{M}_{ij},$$

where we plugged in our earlier formula for $\frac{\partial}{\partial \mathbf{x}_i} f(\mathbf{x})$. We can hence concisely summarize these calculations as: $\nabla^2 f(\mathbf{x}) = \mathbf{M}$ for all $\mathbf{x} \in \mathbb{R}^d$, when f is a quadratic of the form (47).

Now, applying our discussion in Section 5.1, specifically Lemma 8 and (46), shows that a quadratic (47) is convex iff $\mathbf{M} = \nabla^2 f(\mathbf{x})$ is PSD. A famous example of convex quadratics is *linear regression*:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) := \|\mathbf{Ax} - \mathbf{b}\|_2^2 = \sum_{i\in[n]} (\mathbf{a}_i^\top \mathbf{x} - \mathbf{b}_i)^2, \tag{48}$$

where we let $\{\mathbf{a}_i\}_{i\in[n]} \subset \mathbb{R}^d$ be rows of $\mathbf{A}$. Intuitively, (48) models a linear relationship between $\mathbf{b}_i$ and $\mathbf{a}_i$, asking for a coefficient vector $\mathbf{x}$ such that $\mathbf{a}_i^\top\mathbf{x} \approx \mathbf{b}_i$ for all $i \in [n]$. This is the high-dimensional generalization of the “line of best fit” perspective from linear regression in $\mathbb{R}$.

An extreme example where (48) is useful is solving linear systems of equations, $\mathbf{Ax}^\star = \mathbf{b}$. In this case, the minimum value of (48) is clearly zero (by choosing $\mathbf{x} = \mathbf{x}^\star$), so computing the minimizer to (48) returns a solution $\mathbf{x}$ satisfying $\mathbf{Ax} = \mathbf{b}$. In the case where $\mathbf{A}$ has rank-d, $\mathbf{x} = \mathbf{x}^\star$ uniquely.

Why is (48) always convex? It is helpful to first put it in the form of (47):

$$\|\mathbf{Ax} - \mathbf{b}\|_2^2 = (\mathbf{Ax} - \mathbf{b})^\top (\mathbf{Ax} - \mathbf{b}) = \mathbf{x}^\top\mathbf{A}^\top\mathbf{Ax} - 2\mathbf{b}^\top\mathbf{Ax} + \mathbf{b}^\top\mathbf{b}. \tag{49}$$

Thus, up to a constant shift by $\mathbf{b}^\top\mathbf{b}$, linear regression is a quadratic (47) with $\mathbf{M} = 2\mathbf{A}^\top\mathbf{A}$ and $\mathbf{v} = -2\mathbf{A}^\top\mathbf{b}$. Its Hessian is thus the constant matrix $\nabla^2 f(\mathbf{x}) = 2\mathbf{A}^\top\mathbf{A}$. We already argued in (41) that $\mathbf{A}^\top\mathbf{A}$ is always PSD, and multiplication by 2 doubles all eigenvalues. We have therefore established $\nabla^2 f(\mathbf{x})$ is PSD for all $\mathbf{x} \in \mathbb{R}^d$, and hence linear regression is always convex.

Let us pause to contrast the linear regression problem (48) with a problem we tackled in Section 4.3: eigenvector computation. It is a classic fact in linear algebra that if $\mathbf{M}$ if PSD, then

$$\text{argmax}_{\substack{\mathbf{v}\in\mathbb{R}^d\\ \|\mathbf{v}\|_2=1}} \mathbf{v}^\top\mathbf{Mv}$$

evaluates to the top eigenvector of $\mathbf{M}$. Incredibly, this problem is asking for the *maximum* (rather than minimum) of a convex function, over the set $\|\mathbf{v}\|_2 = 1$, which is highly *nonconvex*. It is somewhat miraculous that the power method (Section 4.3) is able to approximately solve this problem, highlighting how highly-structured nonconvex optimization problems may still be tractable.

Gradient descent on quadratics in $\mathbb{R}^d$. Our developments are compatible with Section 3.4. Let $\mathbf{M} := 2\mathbf{A}^\top\mathbf{A}$ and $\mathbf{v} = -2\mathbf{A}^\top\mathbf{b}$, so that as discussed after (49), a problem equivalent to (48) is

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^\top\mathbf{Mx} + \mathbf{v}^\top\mathbf{x}, \tag{50}$$

for PSD $\mathbf{M}$. For notational convenience we let $\mathbf{U}$ and $\boldsymbol{\Lambda} = \mathbf{diag}(\boldsymbol{\lambda})$ be defined as in the eigendecomposition (40), and assume that $\boldsymbol{\lambda}$ is sorted with $\boldsymbol{\lambda}_1 \geq \boldsymbol{\lambda}_2 \geq \ldots \geq \boldsymbol{\lambda}_d > 0$.

For intuition, it is simplest to discuss the case when $d = 2$ and $\mathbf{M}$ is a diagonal matrix,

$$\mathbf{M} = \begin{pmatrix} \boldsymbol{\lambda}_1 & 0 \\ 0 & \boldsymbol{\lambda}_2 \end{pmatrix}.$$

In this case, (50) separates cleanly into two one-dimensional quadratics:

$$f(\mathbf{x}) = \left(\frac{\boldsymbol{\lambda}_1}{2}\mathbf{x}_1^2 + \mathbf{v}_1\mathbf{x}_1\right) + \left(\frac{\boldsymbol{\lambda}_2}{2}\mathbf{x}_2^2 + \mathbf{v}_2\mathbf{x}_2\right),\ \nabla f(\mathbf{x}) = \begin{pmatrix} \boldsymbol{\lambda}_1\mathbf{x}_1 + \mathbf{v}_1 \\ \boldsymbol{\lambda}_2\mathbf{x}_2 + \mathbf{v}_2 \end{pmatrix}.$$

We can now apply the one-dimensional theory in Section 3.2 to minimize f in each coordinate separately. The catch is that we must choose a single gradient descent step size η for all coordinates.

What step size η should we choose? We claim that setting $\eta = \frac{1}{\boldsymbol{\lambda}_1}$ is a good idea. This step size immediately sets $\mathbf{x}_1$ to the minimizer of $\frac{\boldsymbol{\lambda}_1}{2}\mathbf{x}_1^2 + \mathbf{v}_1\mathbf{x}_1$, a $\boldsymbol{\lambda}_1$-smooth function of $\mathbf{x}_1$, in a single step. Moreover, we know that taking $\eta = \frac{1}{\boldsymbol{\lambda}_1}$ in the second coordinate will cause $\mathbf{x}_2$ to never overshoot the optimal second coordinate, because $\eta \leq \frac{1}{\boldsymbol{\lambda}_2}$. How quickly does $\mathbf{x}_2$ converge with this choice of η? We know that had we taken a larger step size of $\frac{1}{\boldsymbol{\lambda}_2}$, we would have immediately converged to the optimal second coordinate. This means that in each step, with the smaller step size $\eta = \frac{1}{\boldsymbol{\lambda}_1}$, we make a $\frac{\boldsymbol{\lambda}_2}{\boldsymbol{\lambda}_1}$ fraction of the progress to the optimizer. We can check that for $\epsilon \in (0, 1)$,

$$\left(1 - \frac{\boldsymbol{\lambda}_2}{\boldsymbol{\lambda}_1}\right)^T \leq \epsilon, \text{ for } T \geq \frac{\boldsymbol{\lambda}_1}{\boldsymbol{\lambda}_2}\log\left(\frac{1}{\epsilon}\right).$$

Formalizing this argument shows gradient descent on our two-dimensional example gives an ϵ-approximate minimizer in $\approx \frac{\boldsymbol{\lambda}_1}{\boldsymbol{\lambda}_2}\log(\frac{1}{\epsilon})$ iterations, for an appropriate notion of approximation.

More generally, although we will not do so here, one can extend this argument to arbitrary linear regression problems, even when $d > 2$ and the associated PSD matrix $\mathbf{M} = 2\mathbf{A}^\top\mathbf{A}$ is not diagonal. The general idea is to still set $\eta = \frac{1}{\boldsymbol{\lambda}_1}$, where $\boldsymbol{\lambda}_1$ is the largest eigenvalue of $\mathbf{M}$. The "slowest-progressing" coordinate corresponds to the smallest eigenvalue, where we should have taken a step size of $\frac{1}{\boldsymbol{\lambda}_d}$; thus, we still make a $\frac{\boldsymbol{\lambda}_d}{\boldsymbol{\lambda}_1}$ fraction of progress in each coordinate. We can take care of the rotation matrix $\mathbf{U}$ by tracking all of our progress in the corresponding orthonormal basis.

Following this reasoning, one can prove that gradient descent on high-dimensional linear regression in $\mathbf{M} = 2\mathbf{A}^\top\mathbf{A}$ converges to an ϵ-approximate minimizer in $\approx \kappa(\mathbf{M})\log(\frac{1}{\epsilon})$ iterations, where

$$\kappa(\mathbf{M}) := \frac{\boldsymbol{\lambda}_1}{\boldsymbol{\lambda}_d}.$$

The quantity $\kappa(\mathbf{M})$ is often called the *condition number* of a PSD matrix. Visually, identifying PSD matrices with ellipsoids, $\kappa(\mathbf{M})$ is the multiplicative distortion between its largest and smallest axes; when $\mathbf{M}$ is a multiple of the identity, the associated ellipsoid is a sphere, and $\kappa(\mathbf{M}) = 1$.

Importantly, in many interesting cases, $\kappa(\mathbf{M})$ does not grow with the dimension d, providing motivation (as in Section 3.4) to use gradient descent for linear regression. We call such problems (with small $\kappa(\mathbf{M})$) "well-conditioned." In cases where the matrix $\mathbf{M}$ has highly-distorted eigenvalues, we call the problem "ill-conditioned," and alternative algorithms to gradient descent are preferred.

5.3 Linear programming algorithms

We bring these notes full circle by applying our gradient descent intuition to algorithms for linear programming (5). Solving an LP appears to be a very different task than the linear regression problem (48). For one thing, LPs are constrained optimization problems, whereas we only discussed how to solve (48) over $\mathbb{R}^d$. For another, the constraint set describing an LP is highly non-smooth, i.e., it has sharp corners, so it appears that gradient descent should not apply. Algorithms for LPs need to overcome these challenges, and hence they are somewhat complicated in general. Nonetheless, we have developed enough machinery and intuition to give an overview of these algorithms, and to appreciate how they work at a (very) high level.

There are three broad families of LP algorithms we will describe here, whose runtimes range from *pseudopolynomial* to *weakly polynomial* (see Section 4.3, Part V for a review). In particular, it is open how to solve an LP defined by constraints $\mathbf{A} \in \mathbb{R}^{n\times d}$ in *strongly polynomial* time, i.e., $\text{poly}(n, d)$ arithmetic operations. In fact, giving a strongly polynomial time LP algorithm was listed by Steve Smale as one of the eighteen most important unsolved mathematics problems [Sma98]. Fortunately, it has been known since breakthrough work of [Kha80] that LPs can be solved in weakly polynomial time. This suffices for highly-accurate approximate LP solutions, which has been enormously influential in both theory and practice.

Simplex. The simplex method, originally developed by George Dantzig, takes a very geometric approach to linear programming. It initializes an iterate $\mathbf{x}_0$ to an *extreme point* of the polytope, i.e., a vertex or "corner" where multiple faces intersect. Intuitively, the algorithm "walks" along edges of the polytope, maintaining the invariant that its iterates are always vertices of the polytope. Notably, it was listed as one of the 10 most influential algorithms of the 20^{th} century [DS00].

Slightly more formally, on a given iteration starting at a vertex $\mathbf{x}_t$, the algorithm then scans all edges meeting at $\mathbf{x}_t$, and chooses any one of them such that moving along that edge improves the objective $\mathbf{c}^\top \mathbf{x}_t$. If there exists such an edge, the simplex algorithm updates $\mathbf{x}_{t+1}$ to the other vertex of the polytope that bounds the chosen edge. If there are no such directions of improvement, then the algorithm terminates. The fact that termination of the simplex method implies that we have found the optimal vertex is essentially equivalent to Farkas' lemma, mentioned in Section 2.3.

The simplex method can be viewed as a descent method, where instead of gradients, we use edges of the polytope as descent directions. In this analogy, the step size η is then set to the "length" of the polytope edge in each step, rather than a fixed constant. Unfortunately, as is typical of iterative descent methods (see, e.g., the pseudopolynomial dependence on $\frac{1}{\epsilon}$ in Lemmas 4 and 6), the simplex method does not always yield high-accuracy solutions in polynomial time. This is for good reason: on a famous construction known as the *Klee-Minty cube* [KM72], the simplex method (with a worst-case edge selection rule) takes $\Omega(2^d)$ time, where d is the problem dimension.

Nonetheless, the simplex method is often very competitive on practical instances, especially in moderate dimension. Towards explaining this empirical success, [ST04] introduced a line of work called *smoothed analysis*, which showed that the simplex method has a polynomial runtime for almost all "mildly random" linear programs (i.e., with perturbed constraints). Conceptually, the [ST04] result suggests that hard instances for the simplex method are relatively scarce.

Interior-point. Interior-point methods (IPMs) are a powerful alternative to the simplex method which have the advantage of being both provably weakly-polynomial time, and the practitioner's method of choice on large LP instances. As evidence of their importance, IPMs are a rare example of an algorithm whose discovery made it onto the cover of the New York Times [Gle84].

Compared to the simplex method, IPMs take an opposite approach to constrained optimization. Rather than walk along the polytope boundary, IPMs maintain that their iterates stay sufficiently far from the polytope faces. More concretely, IPMs aim to maximize a sequence of objectives

$$f_t(\mathbf{x}) \approx \mathbf{c}^\top \mathbf{x} + t \cdot b(\mathbf{x}),$$

where $b(\mathbf{x})$ is a *barrier function* that explodes to $-\infty$ as $\mathbf{x}$ approaches the boundary of the polytope. Intuitively, b prevents $\mathbf{x}$ from leaving the constraint set by "reshaping space" near the polytope boundary so that $\mathbf{x}$ moves less. Observe that f_0 is just our original LP objective. The goal of an IPM is then to alternately set an iterate $\mathbf{x}$ to the minimizer of f_t, and to decrease the t parameter by an inverse-polynomial multiplicative factor. After a few alternating iterations, the t parameter is sufficiently small that the IPM can round its iterate to an optimal vertex of the polytope.

The solutions $\mathbf{x}_t := \text{argmax}_{\mathbf{x}\in\mathbb{R}^d \mid \mathbf{A}\mathbf{x}\le\mathbf{b}}\{f_t(\mathbf{x})\}$ are called the "central path" of the IPM, and intuitively trace out a curved trajectory from the center of the polytope to the optimal vertex, as t decreases. To recenter iterates to the central path, IPMs must choose a barrier function b that sufficiently stable to small changes in t. It turns out that for an appropriate barrier function b, after performing an appropriate reshaping of space near $\mathbf{x}_t$, the function f_t is essentially a well-conditioned quadratic. Thus, the methods of Section 5.2 can be used to minimize $f_{t(1+\delta)}$ for an inverse-polynomially small δ, by efficiently computing the next central path point $\mathbf{x}_{t(1+\delta)}$.

Cutting-plane. The final algorithm we will discuss is *cutting-plane methods* (CPMs). CPMs are my favorite algorithm, and learning about them is the reason that I became a continuous algorithms researcher. For this reason, I hope the reader will humor their inclusion in these notes.

The punchline is that CPMs perform binary search in $\mathbb{R}^d$. In fact, CPMs entirely generalize to optimize arbitrary convex functions, not just LPs. Recall from Section 3.1 that when $f : \mathbb{R} \to \mathbb{R}$ is a one-dimensional convex function, the sign of $f'(x)$ allowed us to perform binary search for the minimizer $x^\star$. In high dimensions, applying the inequality (45) with $\mathbf{y} \leftarrow \mathbf{x}^\star$ shows that

$$f(\mathbf{x}^\star) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{x}^\star - \mathbf{x}) \geq f(\mathbf{x}^\star) + \nabla f(\mathbf{x})^\top (\mathbf{x} - \mathbf{x}^\star)$$
$$\implies \nabla f(\mathbf{x})^\top \mathbf{x}^\star \geq \nabla f(\mathbf{x})^\top \mathbf{x}.$$

Thus, at a point $\mathbf{x}$, we can query its gradient $\mathbf{a} := \nabla f(\mathbf{x})$ to learn a halfspace $\mathbf{a}^\top \mathbf{x}^\star \geq b := \nabla f(\mathbf{x})^\top \mathbf{x}$ that the overall minimizer $\mathbf{x}^\star$ must lie in. Pictorially, this halfspace splits $\mathbb{R}^d$ in two by a plane, and we learn which side of the plane $\mathbf{x}^\star$ belongs to. The idea of cutting plane methods is to always maintain a set $\mathcal{K}$ that contains the minimizer $\mathbf{x}^\star$, and slowly bisect it with the aim of decreasing its volume. At the beginning of the algorithm, $\mathcal{K}$ can either be set to the original constraint set $\mathcal{X}$, or if we have extra domain knowledge, a large enough ball that contains $\mathbf{x}^\star$.

How should we choose the query point $\mathbf{x}$ to repeatedly decrease the volume of $\mathcal{K}$? A natural choice is the *center-of-gravity* of $\mathcal{K}$, i.e., the average of all points inside $\mathcal{K}$. There is an amazing fact called *Grünbaum's theorem* [Gru60] that says for any "cutting plane" slicing through the center-of-gravity of a convex set in $\mathbb{R}^d$, there is at least a $\frac{1}{e} \geq 30\%$ of the volume of $\mathcal{K}$ on both sides of the cutting plane. That is, we may have a 40%-60% split, or a 50%-50% split, but our cutting plane will never heavily favor one side (e.g., a 1%-99% split). Therefore, each CPM iteration decreases the volume of $\mathcal{K}$ by a constant fraction, upon which we can recurse until we have found $\mathbf{x}^\star$ to high accuracy.

There are many details that need to be filled in to fully execute this plan in polynomial time, including: how do we efficiently compute gradients, and how do we efficiently find the center-of-gravity of a convex set? These topics are far beyond the scope of this course, but for an introduction to the central ideas, see, e.g., the first set of lecture notes of [Tia24a].

Further reading

References

[BL07] James Bennett and Stan Lanning. The netflix prize. 2007.

[DS00] J. Dongarra and F. Sullivan. Guest Editors Introduction to the top 10 algorithms . *Computing in Science & Engineering*, 2(01):22–23, 2000.

[Eri24] Jeff Erickson. *Algorithms*. 2024.

[EY36] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[Gle84] James Gleick. Breakthrough in problem solving. *The New York Times*, 1984.

[Gru60] B. Grunbaum. Partitions of mass-distributions and convex bodies by hyperplanes. *Pacific Journal of Mathematics*, 10(4):1257–1261, 1960.

[Kha80] Leonid G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Computational Physics*, 20(1):53–72, 1980.

[KM72] Victor Klee and George J. Minty. How good is the simplex algorithm? In *Inequalities III (Proceedings of the Third Symposium on Inequalities)*, pages 159–175. New York-London: Academic Press, 1972.

[Mir60] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math. Oxford*, 11:50–59, 1960.

[MM15] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, pages 1396–1404, 2015.

[Nes83] Yurii Nesterov. A method for solving a convex programming problem with convergence rate $o(1/k^2)$. *Doklady AN SSSR*, 269:543–547, 1983.

[Sma98] Stephen Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20:7–15, 1998.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.

[Tia24a] Kevin Tian. Cs 395t: Continuous algorithms. https://kjtian.github.io/cs395t.html, 2024.

[Tia24b] Kevin Tian. Cs 395t: Continuous algorithms, part ii (gradient descent). https://kjtian.github.io/notes/CS%20395T%20(Spring%202024)/Part2_main.pdf, 2024.

[vN28] John von Neumann. Zur theorie der gesellschaftsspiele. *Math. Ann.*, 100:295–320, 1928.