CS 435, 2018
Lecture 2, Date: 1 March 2018
Instructor: Nisheeth Vishnoi

Convex Programming and Efficiency

In this lecture, we formalize convex programming problem, discuss what it means to solve it efficiently and present various ways in which a convex set or a function can be specified.

Contents

1 Convex Programs

In this course we are interested in optimization problems of the form

$$\inf_{x \in K} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $K \subseteq \mathbb{R}^n$. When f is a *convex function* and K is a *convex set* we call such a problem a *convex program*, and formulating problems using convex programs and developing methods to solve them is the main focus of the course. We say that a convex program is *unconstrained* when $K = \mathbb{R}^n$, i.e., when we are optimizing over all inputs, and we call it *constrained* when the set K is non-trivial. Further, when f is $\mathcal{C}^1$ function (i.e., f is differentiable, with a continuous derivative) we call (1) a *smooth* convex program and *non-smooth* otherwise. Many of the functions we consider have higher-order smoothness properties and rarely we encounter functions that are not smooth in their domains.

Towards developing algorithms to solve problems of this kind (1), we need to specify a model of computation, explain how we can input a convex set or a function, and what does it mean for an algorithm to be efficient.

2 The Computational Model

As our model of an computation we use the standard notion of a Turing machine (TM) that has a single one-directional infinite tape and works with a finite-sized alphabet and transition function. The *head* of a TM that implements its transitions function encodes the *algorithm*. A computation on a TM is initialized with a binary input $x \in \{0,1\}^\star$ on its tape and terminates with an output $y \in \{0,1\}^\star$ written on it. We measure the running time of such a TM as the *maximum* number of steps performed on an input of a certain length n. Here are two important extensions of this standard model which will show up in our discussions quite frequently:

- **Randomized TMs.** In this model, the TM has access to an additional tape which is an infinite stream of independent and uniformly distributed random bits. We then often allow such machines to be sometimes incorrect (in what they output as a solution), but we require that the probability of providing a correct answer is, say, at least $\frac{2}{3}$.
- **TMs with Oracles.** For some applications it is convenient to consider Turing machines having access to a black-box primitive (called an *oracle*) for answering certain questions or computing certain functions. Formally, such a Turing machine has an additional tape on which it can write a particular input and query the oracle, which then writes the answer to such a query on the same tape. Importantly, we always require that the size of the output of such an oracle is polynomially bounded as a function of the input query length.

As it is standard in computational complexity our notion of efficiency is polynomial time computability, i.e., we would like to design Turing machines (algorithms) which solve a particular problem and have (randomized) polynomial running time. We will see soon that for certain convex programs this goal is possible to achieve, while for others it might be hard or even impossible. The reader is referred to the book [1] for more on computational complexity and time complexity classes such as **P** and its randomized variants such as **RP** and **BPP**.

In subsequent sections we analyze several natural questions one might ask regarding convex sets and convex functions and formalize further what kind of an input shall we provide and what type of a solution are we really looking for.

3 Membership Problem for Convex Sets

Perhaps the simplest computational problem one can consider, which is relevant to convex optimization is:

$$\text{For a point } x \in \mathbb{R}^n \text{ and } K \subseteq \mathbb{R}^n : \quad \text{is } x \in K?$$

We now examine a couple of specific instances of this question and understand how to specify the input (x, K) to the algorithm. While for x, we can imagine writing it down in binary (if it is rational), K is an infinite set and we need to work with various finite representations of it. In this process we introduce several important concepts which will be useful later on.

Example: Halfspace. Let

$$K = \{y \in \mathbb{R}^n : \langle a, y \rangle \leq b\}$$

where $a \in \mathbb{R}^n$ is a vector and $b \in \mathbb{R}$ is a number. K then represents a halfspace in the Euclidean space $\mathbb{R}^n$. For a given $x \in \mathbb{R}^n$, checking whether $x \in K$ is a trivial task and just boils down to verifying whether the inequality $\langle a, x \rangle \leq b$ is satisfied or not. Computing $\langle a, x \rangle$ requires $O(n)$ efficient arithmetic operations and hence is an easy task.

Let us now try to formalize what should an input for an algorithm checking whether $x \in K$ should be. We need to provide x, a and b to this algorithm. For this we need to represent these objects exactly in finite space, hence we assume that these are rational numbers, i.e., $x \in \mathbb{Q}^n$, $a \in \mathbb{Q}^n$ and $b \in \mathbb{Q}$. If we want to provide a rational number $y \in \mathbb{Q}$ on input we represent it as an irreducible fraction $y = \frac{y_1}{y_2}$ for $y_1, y_2 \in \mathbb{Z}$ and give these numbers in binary. The bit complexity $L(z)$ of an integer $z \in \mathbb{Z}$ is defined to be the number of bits required to store its binary representation, thus

$$L(z) := 1 + \lceil \log(|z| + 1) \rceil$$

and we overload this notation to rational numbers by defining

$$L(y) = L\left(\frac{y_1}{y_2}\right) := L(y_1) + L(y_2).$$

Further, if $x \in \mathbb{Q}^n$ is a vector of rationals we define

$$L(x) := L(x_1) + L(x_2) + \cdots + L(x_n).$$

And finally, for inputs being collections of rational vectors, numbers, matrices, etc., we set

$$L(x, a, b) = L(x) + L(a) + L(b),$$

i.e., the total bit complexity of all numbers involved.

Given this definition, one can now verify that checking whether $x \in K$, for a halfspace K can be done in polynomial time with respect to the input length, i.e., the bit complexity $L(x, a, b)$. It is important to note that the arithmetic operations involved in checking this are multiplication, addition, and comparison and all of them can be implemented efficiently on a TM.

Example: Ellipsoid. Let now $K \subseteq \mathbb{R}^n$ be an ellipsoid, i.e., a set of the form

$$K := \{y \in \mathbb{R}^n : y^\top A y \leq 1\}$$

for a positive definite matrix $A \in \mathbb{Q}^{n \times n}$. Similarly as before, the task of checking whether $x \in K$, for an $x \in \mathbb{Q}^n$ can be performed efficiently: indeed computing $x^\top A x$ requires $O(n^2)$ arithmetic operations (multiplications and additions) and thus can be performed in polynomial time with respect to the bit complexity $L(x, A)$ of the input.

Example: Intersections, Polyhedra. Note that being able to answer membership questions about two sets $K_1, K_2 \subseteq \mathbb{R}^n$ allows us to answer questions about their intersection $K_1 \cap K_2$ as well. For instance, consider a polyhedron, i.e., a set of the form

$$K := \{y \in \mathbb{R}^n : \langle a_i, y \rangle \leq b_i \text{ for } i = 1, 2, \ldots, m\},$$

where $a_i \in \mathbb{Q}^n$ and $b_i \in \mathbb{Q}$ for $i = 1, 2, \ldots, m$. Such a K is an intersection of m halfspaces, which are easy to deal with computationally. Answering whether $x \in K$ reduces to m such questions and hence is easy as well – can be answered in polynomial time with respect to the bit complexity $L(x, a_1, b_1, a_2, b_2, \ldots, a_m, b_m)$.

Example: ℓ_1 Ball. Consider the case when

$$K = \{y \in \mathbb{R}^n : \|y\|_1 \leq r\}$$

for $r \in \mathbb{Q}$. Interestingly, K falls into the category of polyhedra, it can be written as:

$$K = \{y \in \mathbb{R}^n : \langle y, s \rangle \leq r, \text{ for all } s \in \{-1, 1\}^n\},$$

and one can argue that none of these 2^n (**exponentially** many!) different linear inequalities inequalities is redundant. Thus by using the method from the previous example, our algorithm clearly would not be efficient. However, in this case one can just check whether

$$\sum_{i=1}^{n} |x_i| \leq r,$$

this requires only $O(n)$ arithmetic operations.

Example: PSD cone. Consider now $K \subseteq \mathbb{R}^{n\times n}$ to be the set of all symmetric PSD matrices. Is it easy to check whether $X \in K$? A simple exercise is to show that K is convex. Recall that for a symmetric matrix $X \in \mathbb{Q}^{n\times n}$

$$X \in K \;\Leftrightarrow\; \forall y \in \mathbb{R}^n \;\; y^\top X y \geq 0.$$

Thus really K is defined as an intersection of infinitely many halfspaces – one for every vector $y \in \mathbb{R}^n$. Clearly, it is not possible to go over all such y and check them one by one, thus a different approach for checking if $X \in K$ is required. Recall that if by $\lambda_{\min}(X)$ we denote the smallest eigenvalue of X (note that because of symmetry, all its eigenvalues are real) then it holds that

$$X \in K \;\Leftrightarrow\; \lambda_{\min}(X) \geq 0.$$

Since eigenvalues of X are simply roots of its characteristic polynomial, one can just try to compute all of them and check if they are nonnegative. However, computing roots of a polynomial is not a trivial matter. On the one hand there are efficient procedures to compute eigenvalues of matrices. On the other hand – they are all approximate, since eigenvalues are typically irrational numbers and thus cannot be represented exactly in binary. More precisely, there are algorithms to compute a sequence of numbers $\lambda'_1, \lambda'_2, \ldots, \lambda'_n$ such that $|\lambda_i - \lambda'_i| < \varepsilon$ where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are eigenvalues of X and $\varepsilon \in (0, 1)$ is the specified precision. Such algorithms can be made run in time polynomial in the bit complexity $L(X)$ and $\log \frac{1}{\varepsilon}$.

Consequently, we can check whether $X \in K$ "approximately". This kind of an approximation is sufficient in many cases. However, in this case, one can avoid being approximate by using so called Sturm chains for the characteristic polynomial of X and compute (exactly) the number of roots of this polynomian in the interval $(-\infty, 0)$.

3.1 Separation Oracles

When considering a question of whether a point x is in a set K or not, in the case when the answer is negative one might want to give a "certificate" for that, i.e., a proof that x is outside of K. It turns out that if K is a convex set, such a certificate always exists and is given by a hyperplane separating x from K (see [6]) More precisely, the following theorem holds:

Theorem 1. *Let $K \subseteq \mathbb{R}^n$ be a closed convex set and $x \in \mathbb{R}^n \setminus K$ then there exists a hyperplane separating K from x, i.e., there exist a vector $a \in \mathbb{R}^n$ and a number $b \in \mathbb{R}$ such that $\langle a, y\rangle < b$ for all $y \in K$ and $\langle a, x\rangle > b$.*

In this theorem the hyperplane separating K from x is $\{y \in \mathbb{R}^n : \langle a, y\rangle = b\}$, as K is on one side of the hyperplane while x is on the other side. A proof of Theorem 1 is not difficult and can be found for instance in [6]. The converse of Theorem 1 is also true: every set which can be separated (by a hyperplane) from every point in its complement is convex.

Theorem 2. *Let $K \subseteq \mathbb{R}^n$ be a closed set. If for every point $x \in \mathbb{R}^n \setminus K$ there exists a hyperplane separating x from K then K is a convex set.*

Thus in other words a convex set can be represented as a collection of hyperplanes separating it from points outside. This motivates the following definition

Definition 3 (Separation Oracle)**.** *A separation oracle for a convex set $K \subseteq \mathbb{R}^n$ is a primitive which:*

- *given $x \in K$, answers YES,*
- *given $x \notin K$, answers NO and outputs $a \in \mathbb{Q}^n$, $b \in \mathbb{Q}$ such that the hyperplane $\{y \in \mathbb{R}^n : \langle a, y\rangle = b\}$ separates x from K.*

Importantly, we always require that the output of a separation oracle, i.e., a and b has polynomial bit complexity $L(a, b)$ as a function of $L(x)$. Separation oracles provide a convenient way of "accessing" a given convex set K, as by Theorems 1 and 2 they provide a complete description of K.

Let us now give some examples and make a couple of remarks regarding separation oracles.

Example: Spanning Tree Polytope. Consider the *spanning tree polytope*: take any undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$ and assume that the edges are labeled by numbers, i.e., $E := [m] = \{1, 2, \ldots, m\}$. Next, let $\mathcal{T} \subseteq 2^{[m]}$ be the set of all spanning trees in G and define the *spanning tree polytope* $P \subseteq \mathbb{R}^m$ as follows:

$$P := \text{conv}\{1_T : T \in \mathcal{T}\},$$

where conv is the convex hull of a set of vectors[1] and $1_T \in \mathbb{R}^m$ is an indicator vector of a set $T \subseteq [m]$[2]. Sets of this form show up and have numerous applications in combinatorial optimization (see [5]). One can describe P using an exponential number of inequalities, however, clearly such a representation is not very practical when one works with graphs of hundreds, thousands or even millions of vertices. Surprisingly, one can prove that there exist polynomial time separation oracles for such polytopes – see for instance [5].

Note that the algorithm to check if a matrix X is PSD can also be turned into an approximate separation oracle for the set of PSD matrices. Compute an approximation λ to the smallest eigenvalue $\lambda_{\min}(X)$ along wth a v such that $Xv = \lambda v$. If $\lambda \geq 0$, output YES. If $\lambda < 0$ output NO along with $vv^\top$. In this case, we know that

$$\langle X, vv^\top \rangle := \text{Tr}(Xvv^\top) = v^\top Xv < 0.$$

In both cases we are correct approximately here the quality of the approximation depends on the error in our estimate for $\lambda_{\min}(X)$.

[1] For a set of vectors $X \subseteq \mathbb{R}^n$ its convex hull $\text{conv}(X) \subseteq \mathbb{R}^n$ is defined as the set of all convex combinations $\sum_{j=1}^r \alpha_j x_j$ for $x_1, x_2, \ldots, x_r \in X$ and $\alpha_1, \alpha_2, \ldots, \alpha_r \geq 0$ such that $\sum_{j=1}^r \alpha_j = 1$.

[2] The indicator vector $1_T \in \{0, 1\}^m$ of a set $T \subseteq [m]$ is defined as $1_T(i) = 1$ for all $i \in T$ and $1_T(i) = 0$ otherwise.

Separation vs Optimization. It is known that constructing efficient (polynomial-time) separation oracles for a given family of convex sets is equivalent to constructing algorithms to optimize *linear* functions over convex sets in this family – see Chapter 2 of [2] for a precise statement of this result and connections between other computational problems involving convex sets. When applying such theorems one has be careful though, as several technicalities show up related to bit complexity, the way how these sets are parametrized, etc. Reading this chapter is highly-recommended.

Composing Separation Oracles. Let K_1, K_2 be two convex sets and that we have access to separation oracles for them. Then we can construct a separation oracle for their intersection $K := K_1 \cap K_2$. Indeed, given a point x we can just check whether x belongs to both of K_1, K_2 and if not, say $x \notin K_1$ then we output the separating hyperplane output by the oracle for K_1.

By a different composition, we can also construct a separating oracle for $K_1 \times K_2 \subseteq \mathbb{R}^{2n}$.

4 Solution Concepts for Optimization Problems

Knowing how to represent convex sets in a computationally convenient way we go back to optimizing functions and to a crucial question: what does it mean to solve a convex program of the form (1)?

One way of approaching this problem might be to formulate it as a decision problem of the following form:

$$\text{Given } c \in \mathbb{Q}, \quad \text{is } \inf_{x \in K} f(x) = c?$$

While in many important cases (e.g. Linear Programming) we can solve this problem, even for the simplest of nonlinear convex programs this may not be possible.

Example: Irrational Solutions. Consider a simple, univariate convex program of the form:

$$\min_{x \geq 1} \frac{2}{x} + x.$$

By a derivative calculation one can easily see that the optimal solution is $x^\star = \sqrt{2}$. Thus, we obtain an irrational optimal solution even though the problem was stated as minimizing a rational function with rational entries. In such a case the optimal solution cannot be represented exactly using a finite binary expansion thus we are forced to consider approximate solutions.

For this reason we often cannot hope to obtain the optimal value

$$y^\star := \inf_{x \in K} f(x)$$

and we can revise the question of asking for an approximation, or in other words: a small interval $[c, c+\varepsilon]$ containing $y^\star$.

$$\text{Given } \varepsilon > 0, \ \text{ compute } c \in \mathbb{Q}, \text{ such that } c \leq \inf_{x \in K} f(x) \leq c + \varepsilon. \tag{2}$$

Since it requires roughly $\log \frac{1}{\varepsilon}$ bits to store ε, we expect an **efficient algorithm** to run in time **poly-logarithmic** in $\frac{1}{\varepsilon}$. We elaborate a little more about this issue in the subsequent section.

Optimal Value vs Optimal Point. Perhaps the most natural way to approach the problem (2) is to try finding a point $x \in K$ such that $y^\star \leq f(x) \leq y^\star + \varepsilon$. Then we can simply take $c := f(x) - \varepsilon$. Note that providing a point $x \in K$ which is a significantly stronger notion of a solution (than just outputting an appropriate c) and provides much more information. The algorithms we consider in this course will typically output both the optimal point as well as the value, but there are several examples of methods which naturally compute the optimal value only, and recovering the optimal input for them is rather non-trivial, if not impossible.

Example: Linear Programming. In linear programming we deal with a linear objective

$$f(x) = \langle c, x \rangle$$

and a polytope[3]

$$K = \{x \in \mathbb{R}^n : Ax \leq b\},$$

where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. It is not hard to show that whenever the problem $\inf_{x \in K} f(x)$ is bounded (i.e., the solution is not $-\infty$), and K has a **vertex**[4], then its optimal value is achieved at some vertex of K. Thus, in order to solve a linear program, it is enough to find the minimum value over all its vertices.

It turns out that every vertex $\tilde{x}$ of K can be represented as a solution to $A'x = b$, for A' being some square submatrix of A. In particular, this means that there is a *rational* optimal solution, whenever the input data is rational. Moreover, if the bit complexity of (A, b) is L then the bit complexity of the optimal solution is polynomial in L (and n) and hence a priori, there is no direct obstacle (from the viewpoint of bit complexity) for obtaining efficient algorithms for exactly solving linear programs.

Distance to Optimum vs Approximation in Value. Consider a convex program which has a unique optimal solution $x^\star \in K$. One can then ask what is the relation between the following two statements about an $x \in K$:

- being approximately optimal in value: $f(x) \leq f(x^\star) + \varepsilon$,

[3]Formally K is a polyhedron – polytopes are defined to be bounded and full-dimensional polyhedra. In linear programming one can assume without loss of generality that K is a polytope.

[4]A vertex of a polytope K is a point $x \in K$ which does not belong to $\mathrm{conv}(K \setminus \{x\})$.

- being close to optimum: $\|x - x^\star\| \leq \varepsilon$.

It is not hard to see that in general (when f is an arbitrary convex function) neither of these conditions implies the other, even approximately. However, as we will see soon, under conditions such as *Lipschitzness* or *strong convexity* we might expect some such implications to hold.

4.1 Representing Functions

Finally, we come to the question of how is the objective function f given to us when trying to solve convex programs of the form (1)? Below we discuss several possibilities and refer to [3] for a more formal and thorough treatment.

Compact Descriptions of Functions. There are several important families of functions which we often focus on, instead of considering the general, abstract case. These include

- Linear and affine functions: $f(x) = \langle c, x \rangle + b$ for a vector $c \in \mathbb{Q}^n$ and $b \in \mathbb{Q}$,
- Quadratic functions: $f(x) = x^\top A x + b^\top x + c$ for a positive semidefinite matrix $A \in \mathbb{Q}^n$, a vector $b \in \mathbb{Q}^n$ and a number $c \in \mathbb{Q}$.
- Linear matrix functions: $f(X) = \mathrm{Tr}(XA)$ where $A \in \mathbb{R}^{n \times n}$ is a fixed symmetric matrix and $X \in \mathbb{R}^{n \times n}$ is a symmetric matrix variable.

All these families of functions can be described compactly, for instance linear functions by providing c and b and quadratic functions by providing A, b, c.

Black-box models. Consider again the example of an undirected graph $G = (V, E)$ along with its set of spanning trees $\mathcal{T}$. Then one can define the following function $g : \mathbb{R}^m \to \mathbb{R}$, for any point θ in the spanning tree polytope P

$$f(y) := \log \left(\sum_{T \in \mathcal{T}} e^{\langle y, 1_T \rangle} \right) - \langle \theta, y \rangle .$$

One can show (exercise) that this function is convex. Can we then minimize f? For this, we would like to (at least) have a method to compute $f(y)$ given $y \in \mathbb{Q}^n$. Whether such an efficient method exists is a separate question[5] which we are not going to discuss now, but let us assume for a moment we are given such a method as a black-box. Thus, we ask a question: can we efficiently solve an optimization problem when instead of a compact description of a function we are just given an oracle access to it?

Note that such an oracle is in fact a complete description of the function, i.e., it hides no information about f, however if we have no idea about what f is, such a description might be hard to work with. In fact, if we drop the convexity assumption and only assume that

[5] In fact one can evaluate g in polynomial time – it follows from a generalized version of Matrix-Tree Theorem.

f is continuous with a small Lipschitz constant, then one can show that no algorithm can efficiently optimize f – see [3] for a proof. Under the convexity assumption the situation improves, but still, typically more information about the local behavior of f around a given point is desired to optimize f efficiently. If the function f is sufficiently smooth one can ask for gradients, Hessians and possibly higher-order derivatives of f. We formalize this concept in the definition below.

Definition 4 (Function Oracles)**.** *For a function $f : \mathbb{R}^n \to R$ and $k \in \mathbb{N}_{\geq 0}$ a kth order oracle for f is a primitive which*

$$\textit{given } x \in \mathbb{R}^n \quad \textit{outputs } D^k f(x),$$

i.e., the kth order derivative of f.

In particular the 0-order oracle (also called an evaluation oracle) is simply a primitive which given x outputs $f(x)$, a 1st-order oracle given x outputs the gradient $\nabla f(x)$ and a 2nd-order oracle outputs the Hessian of f. In some cases, we do not need the Hessian per se but, for a vector v, $(\nabla^2 f(x))^{-1}v$.

We will soon see (when discussing the gradient descent method) that even a 1st order oracle alone is sometimes enough to approximately minimize a convex function (rather) efficiently.

Stochastic Models. Another interesting (and practically relevant) way of representing a function is to provide a randomized black-box procedure for computing their values, gradients or higher-order information. To explain this, consider an example of a quadratic loss function which often shows up in machine learning applications. Given $a_1, a_2, \ldots, a_m \in \mathbb{R}^n$ and $b_1, b_2, \ldots, b_m \in \mathbb{R}$ we define

$$l(x) := \frac{1}{m}\sum_{i=1}^{m} \|\langle a_i, x\rangle - b_i\|^2 .$$

Now suppose that $F(x)$ is a randomized procedure which given $x \in \mathbb{Q}^n$ outputs at random one of $\|\langle a_1, x\rangle - b_1\|^2, \|\langle a_2, x\rangle - b_2\|^2, \ldots, \|\langle a_m, x\rangle - b_m\|^2$, each with probability $\frac{1}{m}$. Then we have

$$\mathbb{E}\left[F(x)\right] = l(x),$$

and thus $F(x)$ provides an unbiased estimator for the value $l(x)$. One can design a similar unbiased estimator for the gradient $\nabla l(x)$. Note that such "randomized oracles" have this advantage over just computing the value or the gradient of l that they are much more efficient computationally – in fact, they run m times faster than their exact deterministic counterparts. Still, in many cases, one can perform minimization given just oracles of this type – for instance using a method called *stochastic gradient descent*.

5 The notion of "Polynomial Time" in Convex Optimization

For decision problems (where the answer is "YES" or "NO" and the input is a string of characters), one defines the class **P** to be all such problems which can be solved by polynomial time algorithms. The randomized counterparts to **P** are **RP**, **BPP** and problems belonging to these classes are also considered efficiently solvable.

For optimization problems we can define a concept analogous to a polynomial time algorithm, yet for this we need to take into account the running time as a function of the precision ε. We say that an algorithm runs in **polynomial time** for a class of optimization problems of the form $\inf_{x \in K} f(x)$ (for a specific family of sets K and functions f), if the time to find an ε-approximate solution $x \in K$ (i.e., such that $f(x) \leq y^\star + \varepsilon$) is polynomial in the bit complexity L of the input, the dimension n and $\log \frac{1}{\varepsilon}$. Similarly we can talk about polynomial time algorithms in the black-box model – then the part depending on L is not present, we require that the number of oracle calls (to the set K and to the function f) is polynomially bounded in the dimension and importantly $\log \frac{1}{\varepsilon}$.

One might ask why do we insist on the dependency on $\frac{1}{\varepsilon}$ to be poly-logarithmic. There are several important applications where a polynomial dependency on $\frac{1}{\varepsilon}$ would be not sufficient. For instance, consider a linear program $\min_{x \in K} \langle c, x \rangle$ with $K = \{x \in \mathbb{R}^n : Ax \leq b\}$ with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. For simplicity, assume there is exactly one optimal solution (a vertex) $v^\star \in K$ of value $y^\star = \langle c, v^\star \rangle$. Suppose we would like to find $v^\star$ in polynomial time, given an algorithm which provides only approximately optimal answers.

How small an $\varepsilon > 0$ do we need to take, to make sure that an approximate solution $\tilde{x}$ (with $\langle c, \tilde{x} \rangle \leq y^\star + \varepsilon$) can be uniquely "rounded" to the optimal vertex $v^\star$? A necessary condition for that is that there is no non-optimal vertex $v \in K$ with

$$y^\star < \langle c, v \rangle \leq y^\star + \varepsilon.$$

Thus we should ask: what is the minimum non-zero distance in value between two distinct vertices of K? By our discussion on linear programming in Section 4 it follows that every vertex is a solution to an $n \times n$ subsystem of $Ax = b$. Thus one can easily deduce that a minimal such gap is at least, roughly 2^{-L}. Consequently, our choice of ε should be no bigger than that, for the rounding algorithm to work properly[6]. Assuming our rounding algorithm is efficient, to get a polynomial time algorithm for linear programming, we need to make the dependence on ε in our approximate algorithm **polynomial in** $\log \frac{1}{\varepsilon}$ in order to be polynomial in L. Thus in this example: an approximate polynomial time algorithm for linear programming yields an exact polynomial time algorithm for linear programming.

Finally, we note that such an approximate algorithm with logarithmic dependence on $\frac{1}{\varepsilon}$ indeed exists (and we will learn a couple of such algorithms in this course).

[6]Such a rounding algorithm is quite non-trivial, and is based on the LLL lattice basis reduction, see for instance [4].

5.1 Is Convex Programming in P or not?

Given the notion of efficiency sketched in the previous sections one may ask: Can every convex program be solved in polynomial time? Even though the class of convex programs is very special and generally believed to have efficient algorithms, the short answer is

No! Not every convex program is polynomial-time solvable.

There are various reasons why this is the case. For instance, not all (even natural examples of) convex sets have polynomial-time separation oracles, which makes the task of optimizing over them impossible. Secondly, it is often hard, or even impossible (especially in black-box models) to design algorithms with logarithmic dependency on the error ε. And finally, being polynomial in the bit complexity L of the input is also often a serious problem, since for certain convex programs, the bit complexity of all close-to-optimal solutions is exponential in L and thus even the space required to output such a solution is already exponential.

Despite these obstacles, there are still numerous interesting algorithms which show that (certain subclasses) of convex programs can be solved efficiently (might not necessarily mean "in polynomial time"). We will see several examples of such in this and subsequent lectures.

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[2] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[3] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014.

[4] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[5] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

[6] Lieven Vandenberghe Stephen Boyd. *Convex optimization*. Cambridge University Press, 2004.