Chapter 2
Semidefinite Programming

Let us start with the concept of *linear programming*. A *linear program* is the problem of maximizing (or minimizing) a linear function in n variables subject to linear equality and inequality constraints. In *equational form*, a linear program can be written as

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{array}$$

Here $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a vector of n variables,[1] $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ is the objective function vector, $\mathbf{b} = (b_1, b_2, \ldots, b_m)$ is the right-hand side, and $A \in \mathbb{R}^{m \times n}$ is the constraint matrix. The bold digit $\mathbf{0}$ stands for the zero vector of the appropriate dimension. Vector inequalities like $\mathbf{x} \geq \mathbf{0}$ are to be understood componentwise.

In other words, among all $\mathbf{x} \in \mathbb{R}^n$ that satisfy the matrix equation $A\mathbf{x} = \mathbf{b}$ and the vector inequality $\mathbf{x} \geq \mathbf{0}$ (such $\mathbf{x}$ are called *feasible solutions*), we are looking for an $\mathbf{x}^*$ with the highest value $\mathbf{c}^T\mathbf{x}^*$.

2.1 From Linear to Semidefinite Programming

To get a semidefinite program, we replace the vector space $\mathbb{R}^n$ underlying $\mathbf{x}$ by another real vector space, namely the vector space

$$\mathrm{SYM}_n = \{X \in \mathbb{R}^{n \times n} : x_{ij} = x_{ji}, 1 \leq i < j \leq n\}$$

of symmetric $n \times n$ matrices, and we replace the matrix A by a linear mapping $A\colon \mathrm{SYM}_n \to \mathbb{R}^m$.

[1] Vectors are column vectors, but in writing them explicitly, we use the n-tuple notation.

B. Gärtner and J. Matousek, *Approximation Algorithms and Semidefinite Programming*, DOI 10.1007/978-3-642-22015-9_2,

The standard scalar product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y}$ over $\mathbb{R}^n$ gets replaced by the standard scalar product

$$X \bullet Y := \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} y_{ij}$$

over SYM_n. Alternatively, we can also write $X \bullet Y = \mathrm{Tr}(X^T Y)$, where for a square matrix M, $\mathrm{Tr}(M)$ (the *trace* of M) is the sum of the diagonal entries of M.

Finally, we replace the constraint $\mathbf{x} \geq \mathbf{0}$ by the constraint

$$X \succeq 0.$$

Here $X \succeq 0$ stands for "the matrix X is positive semidefinite."

Next, we will explain all of this in more detail.

2.2 Positive Semidefinite Matrices

First we recall that a *positive semidefinite matrix* is a real matrix M that is *symmetric* (i.e., $M^T = M$, and in particular, M is a square matrix) and has all eigenvalues *nonnegative*. (The condition of symmetry is all too easy to forget. Let us also recall from Linear Algebra that a symmetric real matrix has only real eigenvalues, and so the nonnegativity condition makes sense.)

Here are several equivalent characterizations.

2.2.1 Fact. *Let $M \in \mathrm{SYM}_n$. The following statements are equivalent.*

(i) *M is positive semidefinite, i.e., all the eigenvalues of M are nonnegative.*
(ii) *$\mathbf{x}^T M \mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$.*
(iii) *There exists a matrix $U \in \mathbb{R}^{n \times n}$ such that $M = U^T U$.*

This can easily be proved using diagonalization, which is a basic tool for dealing with symmetric matrices.

Using the condition (ii), we can see that a semidefinite program as introduced earlier can be regarded as a "linear program with infinitely many constraints." Indeed, the constraint $X \succeq 0$ for the unknown matrix X can be replaced with the constraints $\mathbf{a}^T X \mathbf{a} \geq 0$, $\mathbf{a} \in \mathbb{R}^n$. That is, we have infinitely many linear constraints, one for every vector $\mathbf{a} \in \mathbb{R}^n$.

2.2.2 Definition. *PSD_n is the set of all positive semidefinite $n \times n$ matrices.*

A matrix M is called *positive definite* if $\mathbf{x}^T M\mathbf{x} > 0$ for all $\mathbf{x} \neq \mathbf{0}$. It can be checked that the positive definite matrices form the interior of the set $\mathrm{PSD}_n \subseteq \mathrm{SYM}_n$.

2.3 Cholesky Factorization

In semidefinite programming we often need to compute, for a given positive semidefinite matrix M, a matrix U as in Fact 2.2.1(iii), i.e., such that $M = U^T U$. This is called the computation of a *Cholesky factorization*. (The definition also requires U to be upper triangular, but we don't need this.)

We present a simple explicit method, the *outer product Cholesky Factorization* [GvL96, Sect. 4.2.8], which uses $O(n^3)$ arithmetic operations for an $n \times n$ matrix M.

If $M = (\alpha) \in \mathbb{R}^{1\times 1}$, we set $U = (\sqrt{\alpha})$, where $\alpha \geq 0$ by the nonnegativity of the eigenvalues. Otherwise, since M is symmetric, we can write it as

$$M = \begin{pmatrix} \alpha & \mathbf{q}^T \\ \mathbf{q} & N \end{pmatrix}.$$

We also have $\alpha = \mathbf{e}_1^T M\mathbf{e}_1 \geq 0$ by Fact 2.2.1(ii). Here $\mathbf{e}_i$ denotes the i-th unit vector of the appropriate dimension.

There are two cases to consider. If $\alpha > 0$, we compute

$$M = \begin{pmatrix} \sqrt{\alpha} & \mathbf{0}^T \\ \frac{1}{\sqrt{\alpha}}\mathbf{q} & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & N - \frac{1}{\alpha}\mathbf{q}\mathbf{q}^T \end{pmatrix} \begin{pmatrix} \sqrt{\alpha} & \frac{1}{\sqrt{\alpha}}\mathbf{q}^T \\ \mathbf{0} & I_{n-1} \end{pmatrix}. \tag{2.1}$$

The matrix $N - \frac{1}{\alpha}\mathbf{q}\mathbf{q}^T$ is again positive semidefinite (Exercise 2.2), and we can recursively compute a Cholesky factorization

$$N - \frac{1}{\alpha}\mathbf{q}\mathbf{q}^T = V^T V.$$

Elementary calculations yield that

$$U = \begin{pmatrix} \sqrt{\alpha} & \frac{1}{\sqrt{\alpha}}\mathbf{q}^T \\ \mathbf{0} & V \end{pmatrix}$$

satisfies $M = U^T U$, and so we have found a Cholesky factorization of M.

In the other case ($\alpha = 0$), we also have $\mathbf{q} = \mathbf{0}$ (Exercise 2.2). The matrix N is positive semidefinite (apply Fact 2.2.1(ii) with $\mathbf{x} = (0, x_2, \ldots, x_n)$), so we can recursively compute a matrix V satisfying $N = V^T V$. Setting

$$U = \begin{pmatrix} 0 & \mathbf{0}^T \\ \mathbf{0} & V \end{pmatrix}$$

then gives $M = U^T U$, and we are done with the outer product Cholesky factorization.

Exercise 2.3 asks you to show that the above method can be modified to check whether a given matrix M is positive semidefinite.

We note that the outer product Cholesky factorization is a polynomial-time algorithm only in the real RAM model. We can transform it into a polynomial-time Turing machine, but at the cost of giving up the exact factorization. After all, a Turing machine cannot even exactly factor the 1×1 matrix (2), since $\sqrt{2}$ is an irrational number that cannot be written down with finitely many bits.

The error analysis of Higham [Hig91] implies the following: when we run a modified version of the above algorithm (the modification is to base the factorization (2.1) not on $\alpha = m_{11}$ but rather on the largest diagonal entry m_{jj}), and when we round all intermediate results to $O(n)$ bits (the constant chosen appropriately), then we will obtain a matrix U such that the relative error $\|U^T U - M\|_F / \|M\|_F$ is bounded by 2^{-n}. (Here $\|M\|_F = \left(\sum_{i,j=1}^n m_{ij}^2\right)^{1/2}$ is the *Frobenius norm*.) This accuracy is sufficient for most purposes, and in particular, for the Goemans–Williamson MaxCut algorithm of the previous chapter.

2.4 Semidefinite Programs

2.4.1 Definition. *A semidefinite program in equational form is the following kind of optimization problem:*

$$\begin{array}{ll} \textit{Maximize} & \sum_{i,j=1}^n c_{ij} x_{ij} \\ \textit{subject to} & \sum_{i,j=1}^n a_{ijk} x_{ij} = b_k, \quad k = 1, \ldots, m, \\ & X \succeq 0, \end{array} \tag{2.2}$$

where the x_{ij}, $1 \le i, j \le n$, are n^2 variables satisfying the symmetry conditions $x_{ji} = x_{ij}$ for all i, j, the c_{ij}, a_{ijk} and b_k are real coefficients, and

$$X = (x_{ij})_{i,j=1}^n \in \mathrm{SYM}_n.$$

In a more compact form, the semidefinite program in this definition can be written as

$$\begin{array}{ll} \text{Maximize} & C \bullet X \\ \text{subject to} & A_1 \bullet X = b_1 \\ & A_2 \bullet X = b_2 \\ & \quad \vdots \\ & A_m \bullet X = b_m \\ & X \succeq 0, \end{array} \tag{2.3}$$

where

$$C = (c_{ij})_{i,j=1}^{n}$$

is the matrix expressing the objective function,[2] and

$$A_k = (a_{ijk})_{i,j=1}^{n}, \qquad k = 1, 2, \ldots, m.$$

(We recall the notation $C \bullet X = \sum_{i,j=1}^{n} c_{ij} x_{ij}$ introduced earlier.)

We can write the system of m linear constraints $A_1 \bullet X = b_1, \ldots, A_m \bullet X = b_m$ even more compactly as

$$A(X) = \mathbf{b},$$

where $\mathbf{b} = (b_1, \ldots, b_m)$ and $A\colon \mathrm{SYM}_n \mapsto \mathbb{R}^m$ is a linear mapping. This notation will be useful especially for general considerations about semidefinite programs.

Following the linear programming case, we call the semidefinite program (2.3) *feasible* if there is some *feasible solution*, i.e., a matrix $\tilde{X} \in \mathrm{SYM}_n$ with $A(\tilde{X}) = \mathbf{b}$, $\tilde{X} \succeq 0$. The *value* of a feasible semidefinite program is defined as

$$\sup\{C \bullet X : A(X) = \mathbf{b}, X \succeq 0\}, \tag{2.4}$$

which includes the possibility that the value is ∞. In this case, the program is called *unbounded*; otherwise, we speak of a *bounded* semidefinite program.

An *optimal solution* is a feasible solution X^* such that $C \bullet X^* \geq C \bullet X$ for all feasible solutions X. Consequently, if there is an optimal solution, the value of the semidefinite program is finite, and it is attained, meaning that the supremum in (2.4) is a maximum.

Warning: If a semidefinite program has finite value, generally we *cannot* conclude that the value is attained! We illustrates this with an example below. For applications, this presents no problem: All known efficient algorithms for solving semidefinite programs return only *approximately optimal* solutions, and these are the ones that we rely on in applications.

Here is the example. With $X \in \mathrm{SYM}_2$, let us consider the problem

$$\begin{array}{lrcl} \text{Maximize} & -x_{11} & & \\ \text{subject to} & x_{12} & = & 1 \\ & X & \succeq & 0. \end{array}$$

The feasible solutions of this semidefinite program are all positive semidefinite matrices X of the form

$$X = \begin{pmatrix} x_{11} & 1 \\ 1 & x_{22} \end{pmatrix}.$$

[2] Since X is symmetric, we may also assume that C is symmetric, without loss of generality; similarly for the matrices A_k.

It is easy to see that such a matrix is positive semidefinite if and only if $x_{11}, x_{22} \geq 0$ and $x_{11}x_{22} \geq 1$. Equivalently, if $x_{11} > 0$ and $x_{22} \geq 1/x_{11}$. This implies that the value of the program is 0, but there is no solution that attains this value.

2.5 Non-standard Form

Semidefinite programs do not always look exactly as in (2.3). Besides the constraints given by linear equations, as in (2.3), there may also be inequality constraints, and one may also need extra real variables that are not entries of the positive semidefinite matrix X. Let us indicate how such more general semidefinite programs can be converted to the standard form (2.3).

First, introducing extra nonnegative real variables $x_1, x_2, \ldots, x_k$ not appearing in X can be handled by incorporating them into the matrix. Namely, we replace X with the matrix $X' \in \mathrm{SYM}_{n+k}$, of the form

$$X' = \begin{pmatrix} X & 0 & 0 & \cdots & 0 \\ 0 & x_1 & 0 & \cdots & 0 \\ 0 & 0 & x_2 & \cdots & 0 \\ & & & \ddots & \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \cdots & x_k \end{pmatrix}.$$

We note that the zero entries really mean adding equality constraints to the standard form (2.3). We have $X' \succeq 0$ if and only if $X \succeq 0$ and $x_1, x_2, \ldots, x_k \geq 0$.

To get rid of inequalities, we can add nonnegative slack variables, just as in linear programming. Thus, an inequality constraint $x_{23} + 5x_{15} \leq 22$ is replaced with the equality constraint $x_{23} + 5x_{15} + y = 22$, where y is an extra nonnegative real variable that does not occur anywhere else. Finally, an unrestricted real variable x_i (allowed to attain both positive and negative values) is replaced by the difference $x_i' - x_i''$, where x_i' and x_i'' are two new *nonnegative* real variables.

By these steps, a non-standard semidefinite program assumes the form of a standard program (2.3) over SYM_{n+k} for some k.

2.6 The Complexity of Solving Semidefinite Programs

In Chap. 1 we claimed that under suitable conditions, satisfied in the Goemans–Williamson MaxCut algorithm and many other applications, a semidefinite program can be solved in polynomial time up to any desired accuracy ε. Here we want to make this claim precise.

In order to claim that a semidefinite program is (approximately) solvable in polynomial time, we need to assume that it is "well-behaved" in some sense. Namely, we need that the feasible solutions cannot be too large: we will assume that together with the input semidefinite program, we also obtain an integer R bounding the Frobenius norm of all feasible matrices X.

We will be able to claim polynomial-time approximate solvability only in the case where R has polynomially many digits. As we will see later, one can construct examples of semidefinite programs where this fails and one needs exponentially many bits in order to write down any feasible solution.

What the ellipsoid method can do. The strongest known *theoretical* result on solvability of semidefinite programs follows from the *ellipsoid method* (a standard reference is Grötschel et al. [GLS88]). The ellipsoid method is a general algorithm for maximizing (or minimizing) a given linear function over a given *full-dimensional* convex set C.[3]

In our case, we would like to apply the ellipsoid method to the set $C \subseteq \mathrm{SYM}_n$ of all feasible solutions of the considered semidefinite program.

This set C is convex but not full-dimensional, due to the linear equality constraints in the semidefinite program. But since the affine solution space L of the set of linear equalities can be computed in polynomial time through Gaussian elimination, we may restrict C to this space and then we have a full-dimensional convex set. Technically, this can either be done through an explicit coordinate transformation, or dealt with implicitly (we will do the latter).

The ellipsoid method further requires that C should be enclosed in a ball of radius R and it should be given by a polynomial-time *weak separation oracle* [GLS88, Sect. 2.1]. In our case, this means that for a given symmetric matrix X that satisfies all the equality constraints, we can either certify that it is "almost" feasible (i.e., has small distance to the set PSD_n), or find a hyperplane that almost separates X from C. Polynomial time is w.r.t. the encoding length of X, the bound R, and the amount of "almost."

It turns out that a polynomial-time weak separation oracle is provided by the Cholesky factorization algorithm (see Sect. 2.3 and Exercise 2.3). The only twist is that we need to perform the decomposition "within" L, i.e., for a suitably transformed matrix X' of lower dimension.

Indeed, if the approximate Cholesky factorization goes through, X' is an almost positive semidefinite matrix, since it is close (in absolute terms) to a positive semidefinite matrix $U^T U$. The outer product Cholesky factorization guarantees a small *relative* error, but this can be turned into a small absolute error by computing with $O(\log R)$ more bits.

Similarly, if the approximate Cholesky factorization fails at some point, we can reconstruct a vector $\mathbf{v}$ (by solving a system of linear equations) such that $\mathbf{v}^T X' \mathbf{v}$ is negative or at least very close to zero; this gives us an almost separating hyperplane.

[3] A set C is convex if for all $\mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0,1]$, we also have $(1-\lambda)\mathbf{x} + \lambda\mathbf{y} \in C$.

To state the result, we consider a semidefinite program (P) in the form

$$\begin{array}{ll} \text{Maximize} & C \bullet X \\ \text{subject to} & A_1 \bullet X = b_1 \\ & A_2 \bullet X = b_2 \\ & \quad \vdots \\ & A_m \bullet X = b_m \\ & X \succeq 0. \end{array}$$

Let $L := \{X \in \mathrm{SYM}_n : A_i \bullet X = b_i, i = 1, 2, \ldots, m\}$ be the affine subspace of matrices satisfying all the equality constraints. Let us say that a matrix $X \in \mathrm{SYM}_n$ is an *ε-deep feasible solution* of (P) if all matrices $Y \in L$ of (Frobenius) distance at most ε from X are feasible solutions of (P).

Now we can state a precise result about the solvability of semidefinite programs, which follows from general results about the ellipsoid method [GLS88, Theorem 3.2.1. and Corollary 4.2.7].

2.6.1 Theorem. *Let us assume that the semidefinite program (P) has rational coefficients, let R be an explicitly given bound on the maximum Frobenius norm $\|X\|_F$ of all feasible solutions of (P), and let $\varepsilon > 0$ be a rational number.*
Let us put $v_{\mathrm{deep}} := \sup\{C \bullet X : X$ an ε-deep feasible solution of (P)$\}$. There is an algorithm, with runtime polynomial in the (binary) encoding sizes of the input numbers and in $\log(R/\varepsilon)$, that produces one of the following two outputs.

(a) *A matrix $X^* \in L$ (i.e., satisfying all equality constraints) such that $\|X^* - X\|_F \leq \varepsilon$ for some feasible solution X, and with $C \bullet X^* \geq v_{\mathrm{deep}} - \varepsilon$.*
(b) *A certificate that (P) has no ε-deep feasible solutions. This certificate has the form of an ellipsoid $E \subset L$ that, on the one hand, is guaranteed to contain all feasible solutions, and on the other hand, has volume so small that it cannot contain an ε-ball.*

One has to be careful here: This theorem does not yet imply the informal claim made in Chap. 1. It does so if R is not too large. Unfortunately, R may have to be very large in general, namely doubly-exponential in n, the matrix size; see the pathological example below. In such a case, the bound of Theorem 2.6.1 is exponential!

What saves us in the applications is that R is usually small. In the MAX-CUT application, for example, all entries of a feasible solution X are inner products of unit vectors. Hence the entries are in $[-1, 1]$, and thus $\|X\|_F \leq n$.

A glance at other algorithms. First we want to point out that the ellipsoid method is the *only known* method that provably yields polynomial runtime

in the Turing machine model, at least under suitable and fairly general conditions such as a good bound R.

On the other hand, the practical performance of the ellipsoid method is poor, and completely different algorithms have made semidefinite programming into an extremely powerful computational tool in practice.

Perhaps the most significant and most widely used class of algorithms are *interior-point methods*, which we will outline in Chap. 6. On the theoretical side, they are capable of providing polynomial-time bounds in the RAM model, but there is no control over the sizes of the intermediate numbers that come up in the computations, as far as we could find in the (huge) literature. Moreover, describing these methods in full detail is beyond the scope of this book.

In order to provide a simple and complete algorithm for semidefinite programming, we will present and analyze *Hazan's algorithm* in Chap. 5. This is a recent alternative method for approximately solving semidefinite programs, with a polynomial bound on the running time in the real RAM model. It comes with output guarantees similar to the ones in Theorem 2.6.1 above, and it is efficient in practice. However, the running time bound is polynomial only in $1/\varepsilon$ and not in $\log(1/\varepsilon)$.

A semidefinite program where all feasible solutions are huge. To get such a pathological example, let us consider a semidefinite program with the following constraints:

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 2 & x_1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & x_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x_2 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & x_2 & x_3 & \cdots & 0 & 0 \\ & & & \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & x_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & x_{n-1} & x_n \end{pmatrix} \succeq 0.$$

This is in fact a constraint of the form $X \succeq 0$, along with various equalities involving entries of X. Due to the block structure, we have $X \succeq 0$ if and only if

$$\begin{pmatrix} 1 & x_{i-1} \\ x_{i-1} & x_i \end{pmatrix} \succeq 0, \quad i = 1, \ldots, n,$$

where $x_0 := 2$. But this implies

$$\det \begin{pmatrix} 1 & x_{i-1} \\ x_{i-1} & x_i \end{pmatrix} = x_i - x_{i-1}^2 \geq 0, \quad i = 1, \ldots, n,$$

equivalently $x_i \geq x_{i-1}^2$, $i = 1, \ldots, n$. It follows that

$$x_n \geq 2^{2^n}$$

for every feasible solution, which is doubly-exponential in n. Hence, the encoding size of x_n (when written as say a rational number) is exponential in n and also in the number of variables.

Exercises

2.1 Prove or disprove the following claim: For all $A, B \in \mathrm{SYM}_n$, we also have $AB \in \mathrm{SYM}_n$.

2.2 Fill in the missing details of the outer product Cholesky factorization.

(i) If the matrix

$$M = \begin{pmatrix} \alpha & \mathbf{q}^T \\ \mathbf{q} & N \end{pmatrix}$$

is positive semidefinite with $\alpha > 0$, then the matrix

$$N - \frac{1}{\alpha}\mathbf{q}\mathbf{q}^T$$

is also positive semidefinite.

(ii) If the matrix

$$M = \begin{pmatrix} 0 & \mathbf{q}^T \\ \mathbf{q} & N \end{pmatrix}$$

is positive semidefinite, then also $\mathbf{q} = \mathbf{0}$.

2.3 Show that the outer product Cholesky factorization can also be used to test whether a matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite.

2.4 A *rank-constrained* semidefinite program is a problem of the form

$$\begin{array}{ll} \text{Maximize} & C \bullet X \\ \text{subject to} & A(X) = \mathbf{b} \\ & X \succeq 0 \\ & \mathrm{rank}(X) \leq k, \end{array}$$

where k is a fixed integer. Show that the problem of solving a rank-constrained semidefinite program is NP-hard for $k = 1$.

2.5 A matrix $M \in \mathbb{R}^{n \times n}$ is called a *Euclidean distance matrix* if there exist n points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^n$, such that M is the matrix of pairwise squared Euclidean distances, i.e.,

$$m_{ij} = \|\mathbf{p}_i - \mathbf{p}_j\|^2, \quad 1 \le i, j \le m.$$

Prove that a matrix M is a Euclidean distance matrix if and only if M is symmetric, $m_{ii} = 0$ for all i, and

$$\mathbf{x}^T M \mathbf{x} \le 0 \text{ for all } \mathbf{x} \text{ with } \sum_{i=1}^{n} x_i = 0.$$

2.6 Let $G = (\{1, \ldots, n\}, E)$ be a graph with two edge weight functions $\alpha_e \le \beta_e$, $e \in E$. We want to know whether there exist points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^n$, such that

$$\alpha_{\{i,j\}} \le \|\mathbf{p}_i - \mathbf{p}_j\|^2 \le \beta_{\{i,j\}}, \text{ for all } \{i, j\} \in E.$$

Show that this decision problem can be formulated as a semidefinite program!

2.7 (Sums of squares and minimization I)

(a) Let $p(x) \in \mathbb{R}[x]$ be a univariate polynomial of degree d with real coefficients. We would like to decide whether $p(x)$ is a *sum of squares*, i.e., if it can be written as $p(x) = q_1(x)^2 + \cdots + q_m(x)^2$ for some $q_1(x), \ldots, q_m(x) \in \mathbb{R}[x]$. Formulate this problem as the feasibility of a semidefinite program.

(b) Let us call a polynomial $p(x) \in \mathbb{R}[x]$ *nonnegative* if $p(x) \ge 0$ for all $x \in \mathbb{R}$. Obviously, a sum of squares is nonnegative. Prove that the converse holds as well: Every nonnegative univariate polynomial is a sum of squares. (Hint: First factor into quadratic polynomials.)

(c) Let $p(x) \in \mathbb{R}[x]$ be a given polynomial. Express its global minimum $\min\{p(t) : t \in \mathbb{R}\}$ as the optimum of a suitable semidefinite program (use (b) and a suitable extension of (a)).

2.8 (Sums of squares and minimization II)

(a) Now let $p(x_1, \ldots, x_n)$ be a polynomial in n variables of degree d with real coefficients, and as in Exercise 2.7, we ask whether it can be expressed as a sum of squares (of n-variate real polynomials). Formulate this problem as the feasibility of a semidefinite program. How many variables and constraints are there in this SDP?

(b) Verify that the *Motzkin polynomial* $p(x, y) = 1 + x^2y^2(x^2 + y^2 - 3)$ is nonnegative for all pairs $(x, y) \in \mathbb{R}^2$, but it is not a sum of squares.

Even though part (b) shows that for multivariate polynomials, nonnegativity is not equivalent to being a sum of squares, the multivariate version of the method from Exercise 2.7 constitutes a powerful tool in practice, which can find a global minimum in many cases (see, e.g., [Par06] or [Las10]).

http://www.springer.com/978-3-642-22014-2

Approximation Algorithms and Semidefinite Programming

Gärtner, B.; Matousek, J.

2012, XI, 251 p., Hardcover
ISBN: 978-3-642-22014-2