# Semidefinite Programming

convex optimisation, from numbers to matrices

Martin Burke (with Claude)

Draft

**Abstract**

Most accounts of optimisation draw the important line in the wrong place. The line that matters is not between *linear* and *nonlinear* problems but between *convex* and *nonconvex* ones: convexity is what lets you trust an answer and, remarkably, lets the problem hand you a short proof that the answer is optimal. This note develops that idea and follows it to its richest practical conclusion, the *semidefinite program* (SDP). We climb a single ladder—optimise over numbers, then over vectors confined to a cone, then over *matrices* required to be positive semidefinite—and show that all three rungs are the same conic template wearing progressively roomier clothes. The positive semidefinite cone is the hero of the story: the most expressive convex set we can still optimise over efficiently. We explain what the constraint $X \succeq 0$ really means (it has four equivalent disguises), how a surprising range of problems collapses into an SDP through the Schur complement, why every SDP comes paired with a *dual* that certifies optimality, and how interior-point methods actually solve one using the $-\log\det$ barrier. We close with three worked examples—the nearest correlation matrix, Lyapunov stability, and the Goemans–Williamson relaxation of Max-Cut—and a forty-line solver you can read in one sitting.

# Contents

# 1 The watershed is convexity, not linearity

There is a tempting story about optimisation that goes like this: the easy problems are the *linear* ones—straight lines, flat planes, the stuff of school algebra—and everything gets hard the moment a curve appears. It is a tidy story, and it is wrong. The line that actually predicts whether you can solve a problem reliably does not run between linear and nonlinear at all. As the field likes to put it [1], the great watershed in optimisation is not linearity but *convexity*: on one side sit problems we can solve at scale and *trust* the answer; on the other sit problems where the best we can usually do is search, hope, and never quite know.

To say what that means, fix the object of study. An *optimisation problem* asks us to

$$\text{minimise } f(x) \quad \text{over } x \in C, \tag{1}$$

where $x$ is the *decision variable* (a vector of numbers we get to choose), $f$ is the *objective* (a single real number measuring how good a choice is— lower is better), and $C$ is the *feasible set* (the choices the rules allow). Maximising is just minimising $-f$, so one verb covers both. Everything in this note is some incarnation of (1); what changes from rung to rung is the *shape* of $f$ and $C$. The problem is *convex* when $C$ is a convex set and $f$ is a convex function—two notions we make precise in §2— and convexity, once present, hands you two gifts that nothing else does.

**The first gift: a local minimum is automatically the global one.** In a convex problem there is, in effect, a single basin. Walk downhill from anywhere and you arrive at the same lowest point; when an algorithm reports "I can no longer improve," you may believe it, because there is no better valley hiding over the next ridge. A *nonconvex* landscape offers no such guarantee. It is a range of hills and hollows, and an algorithm that stops at the bottom of *a* hollow has no way to know whether a deeper one lies just out of view—so where you happened to start decides which trap you fall into (Figure 1). This is the practical difference between an answer and *the* answer.

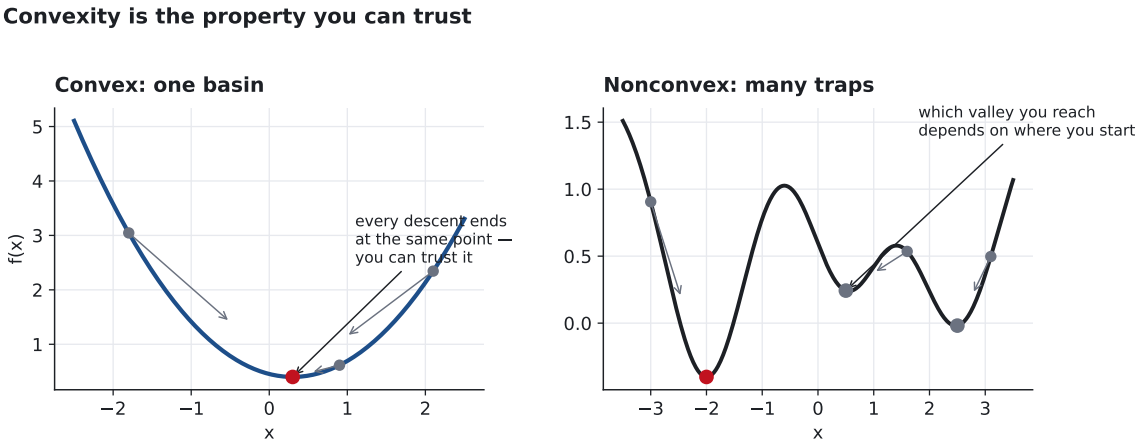


**Figure 1: Why convexity lets you trust the answer.** *Left*: a convex objective has a single basin, so every downhill path—whatever its starting point— ends at the same global minimum (red). *Right*: a nonconvex objective has many local minima; descent halts at whichever hollow you happened to start above, and there is no local signal telling you a deeper valley exists elsewhere. When a convex solver stops, the answer is provably best; when a nonconvex one stops, it is merely the best *nearby*.

**The second gift: a certificate.** This is the deeper and, at first, the more surprising one. A convex problem does not merely let you *find* the optimum—it lets you *prove* you have found it. Paired with every convex problem is a companion problem, its *dual*, whose feasible points supply guaranteed bounds on the original (the *primal*). Produce a primal solution and a matching dual one whose values agree, and that pair is a short, self-contained *certificate of optimality*: anyone

can check it without rerunning your solver or trusting your code. This is what separates "the algorithm converged" from "here is a proof." We make the claim precise in §7; for now, hold onto the promise that convexity buys not just a good answer but an *auditable* one.

So convexity—not linearity—marks the boundary of what we can solve reliably and at scale. The rest of this note climbs a ladder of ever-richer convex problems, each a strict superset of the last: first *linear programs*, where the objective and constraints are flat; then *second-order-cone programs*, which bend the constraints into a smooth cone; and finally the destination, *semidefinite programs*, where the decision variable is a whole matrix required to be positive semidefinite. Each rung keeps both gifts—trustworthy optima and checkable certificates—while buying a great deal more expressive power. Before we can build that ladder, though, we need to say precisely what "convex" means.

## 2 Convex sets and convex functions

Convexity is one idea wearing two outfits—one for sets, one for functions—and the whole subject hinges on being fluent in both. We take them in turn and then, crucially, on the rules that let us *build* convex objects out of simpler ones.

**Convex sets.** A set $C \subseteq \mathbb{R}^n$ is *convex* if it passes the *chord test*: take any two points in the set, draw the straight line segment between them, and the entire segment stays inside. Formally, for every $x, y \in C$ and every blend $\theta \in [0, 1]$,

$$\theta x + (1 - \theta)\, y \in C. \tag{2}$$

The expression $\theta x + (1-\theta)y$ is just a point sliding along the segment from $y$ (at $\theta = 0$) to $x$ (at $\theta = 1$); requiring it to belong to $C$ for *all* blends is exactly the statement that $C$ has no dents, notches, or holes. A solid ball is convex; so is a *half-space* (everything on one side of a flat plane, $\{x : a^\mathsf{T} x \le b\}$); so is the *intersection* of any number of half-spaces—and that last fact is quietly important, because a polygon, a polytope, and indeed the feasible set of any linear program is precisely such an intersection. A crescent moon, by contrast, fails the test: a chord between its two horns slips out through the bite (Figure 2).

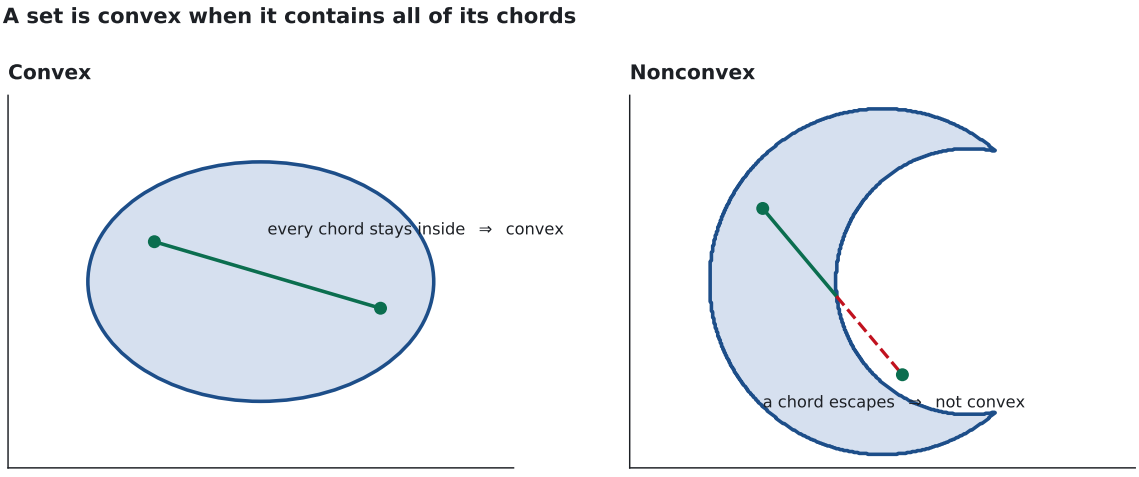


**Figure 2: The chord test for sets.** *Left*: a filled ellipse is convex— pick any two points and the whole segment joining them stays inside. *Right*: a crescent is not convex—a chord between two of its points escapes the set through the concavity. Convexity is simply the absence of any such escape.

**Convex functions.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if its graph bends upward like a bowl, never downward like a dome. The chord test transfers almost verbatim: pick two points on

the graph, draw the straight chord between them, and the chord lies *on or above* the graph everywhere in between. In symbols, for all $x, y$ and $\theta \in [0,1]$,

$$f(\theta x + (1-\theta)y) \;\le\; \theta f(x) + (1-\theta) f(y). \tag{3}$$

The left side is the function's value at the blended point; the right side is the height of the chord there; "function below chord" is the whole of convexity for functions. (A bowl satisfies it; a hill, where the chord cuts *below* the summit, does not.)

There is a second, equivalent definition that turns out to be the conceptual bridge of the whole subject. The *epigraph* of $f$ is the region of the plane (or space) lying on or above its graph,

$$\operatorname{epi} f \;=\; \{(x,t) : t \ge f(x)\}, \tag{4}$$

and the punchline is this: *$f$ is a convex function exactly when* $\operatorname{epi} f$ *is a convex set* (Figure 3). This is not a coincidence but a translation dictionary. It means every statement about convex functions can be recast as a statement about convex sets and vice versa—and it is why a subject that begins by talking about minimising functions ends up, in §3, talking about *cones*, which are sets.

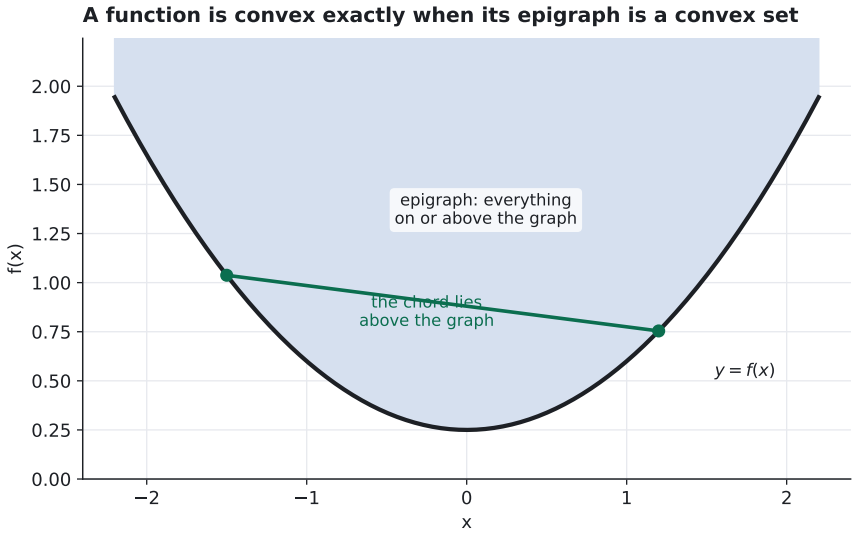


**Figure 3: A function is convex iff its epigraph is a convex set.** The shaded region is the epigraph—everything on or above the graph. Because the curve bows upward, any chord (the straight line between two points on the graph) lies above the curve, equivalently inside the shaded set. The two pictures—chord-above-graph and epigraph-is-convex—are the same fact seen from two sides.

**The calculus of convexity.** Definitions tell you how to *check* convexity; what makes the subject usable is knowing how to *build* it. A handful of operations are *convexity-preserving*: feed them convex ingredients and they return something convex, no fresh proof required. The essential four:

- *Nonnegative weighted sums.* If $f_1, \dots, f_k$ are convex and the weights $w_i \ge 0$, then $\sum_i w_i f_i$ is convex. (Adding bowls, and scaling a bowl by a nonnegative number, keeps it a bowl.)
- *Pointwise maximum and supremum.* If each $f_\alpha$ is convex, so is their upper envelope $f(x) = \sup_\alpha f_\alpha(x)$—take, at every $x$, the highest of the curves. (Geometrically the epigraph of the max is the *intersection* of the individual epigraphs, and intersections of convex sets are convex.) Flag this one: it is the reason the *largest eigenvalue* of a symmetric matrix will turn out to be a convex function of that matrix in §6, which is the hinge on which much of semidefinite modelling hangs.

- *Composition with an affine map.* If $f$ is convex then so is $g(x) = f(Ax + b)$ for any matrix $A$ and vector $b$. Stretching, rotating, and shifting the input cannot un-bowl a bowl.
- *Intersection of convex sets.* The intersection of any family of convex sets is convex—which is exactly why piling up constraints (each carving out a convex feasible region) leaves the overall feasible set convex.

These rules are the working grammar of convex modelling. In practice one almost never verifies (3) from scratch; one recognises a problem as *assembled* from convex atoms by convexity-preserving glue.

**Convexity is compositional.** Recognising that a problem is convex is rarely a matter of grinding through the chord inequality. It is a matter of seeing that the objective and constraints were *built*—out of norms, sums, maxima, and affine maps—using only operations that cannot destroy convexity. Master the handful of preserving operations and you can read convexity off the page.

## 3 From one inequality to a cone

The simplest constraint in all of optimisation is $x \geq 0$: a number must be nonnegative. Stack $n$ of them and you get the vector constraint "$x \geq 0$ componentwise," the backbone of every linear program. Now ask a slightly mischievous question: what *is* that constraint, geometrically? It says $x$ must live in the set of vectors with all coordinates nonnegative—the *nonnegative orthant*, the corner of space you reach by walking only in positive directions. The deep idea of this note is that "$x$ lives in the nonnegative orthant" is one instance of a far more general and more powerful notion: "$x$ lives in a *cone*," and choosing a roomier cone buys a roomier class of problems while changing almost nothing about the machinery.

**Cones and the order they induce.** A set $K$ is a *cone* if it is closed under nonnegative scaling: whenever $x \in K$ and $\lambda \geq 0$, the scaled vector $\lambda x \in K$ too. (Geometrically, a cone is a set you can see entirely by standing at the origin and shooting rays—if a point is in $K$, the whole ray from the origin through it is in $K$.) For a cone to define a sensible notion of order we ask for a little more. A *proper cone* is a cone that is

- *convex* (it passes the chord test of §2);
- *closed* (it contains its boundary—no missing edges);
- *pointed* (it contains no line—if both $x$ and $-x$ lie in $K$ then $x = 0$, so the cone does not double back on itself); and
- *solid* (it has nonempty interior—it is genuinely full-dimensional, not a flat sliver).

A proper cone $K$ does something lovely: it defines a *partial order*. We write $x \succeq_K 0$ to mean simply "$x \in K$," and read it as "$x$ is nonnegative in the sense of $K$." For the nonnegative orthant this recovers ordinary componentwise $\geq$; for richer cones it will mean something genuinely new—and yet every rule of manipulation we are used to (add two nonnegative things and the sum is nonnegative, scale by a positive number, and so on) survives, because those rules are exactly what "proper cone" was built to guarantee.

**One template, three families.** With a proper cone in hand we can write down a single problem template that contains the whole ladder of this note. The *conic program* is

$$\min_{x} \; c^{\mathsf{T}} x \quad \text{s.t.} \quad Ax = b, \;\; x \in K, \tag{5}$$

where we minimise a *linear* objective $c^{\mathsf{T}}x$ over the *affine* slice $\{x : Ax = b\}$ (the equality constraints), intersected with the proper cone $K$. What is remarkable is how little of (5) we have to touch to climb the ladder: the objective stays linear, the equalities stay linear, and *the only thing that changes from one family of problems to the next is the choice of cone* $K$. Three choices give the three families that matter.

- **The nonnegative orthant** $K = \mathbb{R}^n_+ = \{x : x_i \geq 0\}$ gives the *linear program* (LP): linear objective, linear equalities, and each coordinate pinned nonnegative. This is the classical workhorse of operations research.

- **The second-order cone**—also called the *Lorentz cone* or, more evocatively, the *ice-cream cone*—

$$\mathcal{Q}^{n+1} = \{(u, t) \in \mathbb{R}^n \times \mathbb{R} : \|u\|_2 \leq t\} \tag{6}$$

  gives the *second-order-cone program* (SOCP). The constraint $\|u\|_2 \leq t$ says the vector part $u$ must fit inside a circular cone whose half-angle is 45° and whose tip is the origin (Figure 4)—a smooth, curved generalisation of "nonnegative" that lets us model quadratic and norm constraints exactly.

- **The positive semidefinite cone** $\mathbb{S}^n_+ = \{X \in \mathbb{S}^n : X \succeq 0\}$, the set of symmetric matrices with no negative eigenvalues, gives the *semidefinite program* (SDP)—the subject of the entire rest of this note. Here the decision variable is not a vector but a *matrix*, and "nonnegative" has become "positive semidefinite."

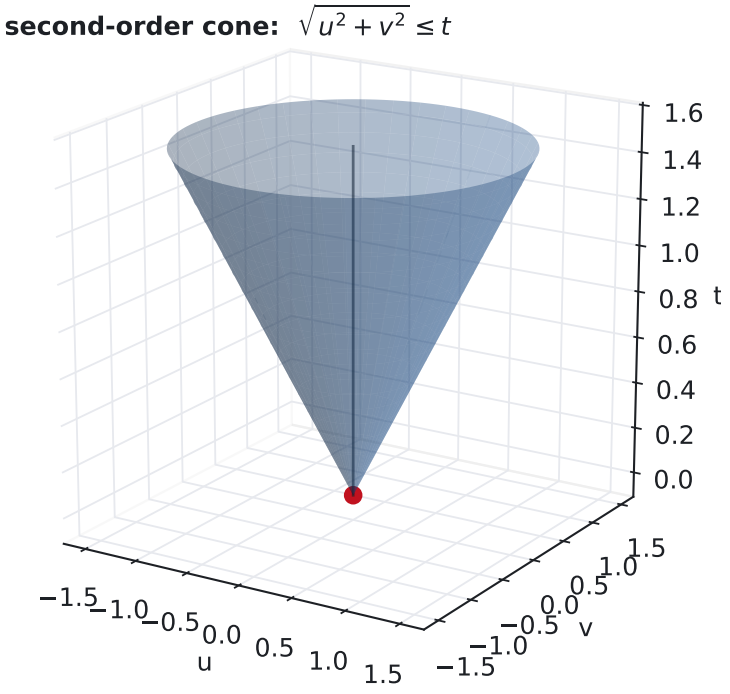


**Figure 4: The second-order (ice-cream) cone.** The set $\{(u, v, t) : \sqrt{u^2 + v^2} \leq t\}$ in three dimensions: every horizontal slice at height $t$ is a disc of radius $t$, so the boundary opens upward at a constant 45°. The constraint "$(u, t)$ lies in this cone" is a smooth, curved cousin of "$x \geq 0$"—and choosing it in the conic template (5) turns a linear program into a second-order-cone program.

**Why the nesting matters.** These three are not independent inventions; they nest, LP $\subseteq$ SOCP $\subseteq$ SDP, each a special case of the next (Figure 5). The componentwise constraint $x \geq 0$

is the second-order cone in one dimension, so every LP is an SOCP; and—as we will see in §6—a second-order-cone constraint can be rewritten exactly as a constraint that a certain symmetric matrix be positive semidefinite, so every SOCP is an SDP. The semidefinite cone sits at the top because it is the most expressive of the three: the roomiest cone we can still optimise over efficiently.

That single unifying template is the reason the climb is worth making. Because LP, SOCP and SDP are all the same problem (5) with a different cone plugged in, they share *one* theory and *one* toolkit. A single duality theory (§7) supplies the optimality certificates promised in §1 for all three at once; a single family of algorithms—the interior-point methods of §8—solves all three by the same device of sliding through the cone's interior. Learn the template once and the whole ladder comes with it.

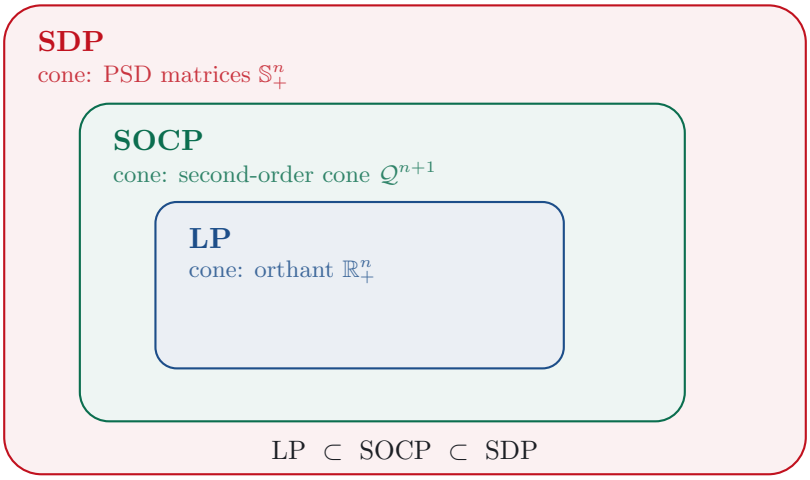


**Figure 5: One template, a nested ladder of cones.** Linear, second-order-cone and semidefinite programs are the same conic program (5) differing only in the cone $K$: the nonnegative orthant, the second-order cone, and the positive semidefinite cone. Each class is a strict special case of the next (LP $\subset$ SOCP $\subset$ SDP), so one duality theory and one interior-point solver serve the whole ladder.

## 4 The positive semidefinite cone

We now meet the hero of the story. Everything from here on lives in $\mathbb{S}^n$, the space of real symmetric $n \times n$ matrices — a vector space like any other (you can add matrices and scale them), but one we equip with the *trace inner product*

$$\langle A,\, B\rangle \;=\; \operatorname{tr}(A^{\mathsf{T}}B) \;=\; \sum_{i,j} A_{ij}B_{ij}. \tag{7}$$

Read the right-hand side and the meaning is plain: lay the two matrices on top of each other, multiply entrywise, and add everything up. It is the ordinary dot product you already know, just applied to matrices flattened into long vectors. This single bracket is the workhorse of the whole subject — objectives and constraints in an SDP are built from it — so it is worth keeping the entrywise picture in mind even when the $\operatorname{tr}(A^{\mathsf{T}}B)$ form is more convenient on paper. (Appendix A collects the linear-algebra facts we lean on.)

A symmetric matrix $X \in \mathbb{S}^n$ is *positive semidefinite*, written $X \succeq 0$, when it passes a certain nonnegativity test. The remarkable thing — and the reason the constraint is so versatile — is that the test can be stated in four completely different-looking ways that turn out to be *exactly the same condition.*

### One condition, four costumes

$X \succeq 0$ *is one condition wearing four costumes.* Each is the natural one for a different job, and fluency in switching between them is most of what it takes to think clearly about SDP.

(i) **The energy test.** The quadratic form is never negative:

$$v^{\mathsf{T}} X v \;\geq\; 0 \qquad \text{for every } v \in \mathbb{R}^n. \tag{8}$$

This is the picture to hold in your head: feed $X$ any direction $v$, and the number $v^{\mathsf{T}} X v$ — a stored energy, a variance, a curvature — comes back nonnegative. *Convenient for:* proving things, because it reduces a statement about a matrix to a statement about ordinary numbers.

(ii) **The spectral test.** Every eigenvalue of $X$ is nonnegative, $\lambda_i(X) \geq 0$ for all $i$. By the spectral theorem (Appendix A) a symmetric matrix has an orthonormal basis of eigenvectors and only real eigenvalues, so this is well defined; writing $v$ in that eigenbasis turns (8) into a sum $\sum_i \lambda_i (v^{\mathsf{T}} q_i)^2$, which makes the equivalence with (i) immediate. *Convenient for:* geometry and computation — the eigenvalues are what a solver actually watches.

(iii) **The factorisation test.** $X$ is a *Gram matrix*: $X = B^{\mathsf{T}} B$ for some matrix $B$ (whose columns we may call $b_1, \ldots, b_n$), so that $X_{ij} = b_i^{\mathsf{T}} b_j$ is literally a table of inner products. Every covariance or correlation matrix is of exactly this form. This is the "$X$ is a squared thing" picture: just as a real number is $\geq 0$ precisely when it is some $r^2$, a matrix is PSD precisely when it is some $B^{\mathsf{T}} B$. *Convenient for:* modelling, because so many quantities in statistics, geometry, and physics arrive already as Gram matrices.

(iv) **The determinant test.** *All* principal minors of $X$ are $\geq 0$. (A principal minor is the determinant of the submatrix you get by keeping the same set of rows and columns.) A word of caution that trips up many readers: the convenient shortcut — check only the *leading* principal minors, the top-left $1\times 1$, $2\times 2, \ldots,$ $n \times n$ blocks — is Sylvester's criterion, and it tests *positive definiteness* $X \succ 0$ (all $> 0$), *not* semidefiniteness. For $X \succeq 0$ the leading minors are not enough: $\operatorname{diag}(0, -1)$ has leading minors 0 and 0 yet is not PSD. You must check *all* principal minors. *Convenient for:* tiny hand computations, $2 \times 2$ and $3 \times 3$.

### The set of PSD matrices is a proper cone

Collect all the matrices that pass the test into one set,

$$\mathbb{S}^n_+ \;=\; \{\, X \in \mathbb{S}^n : X \succeq 0 \,\}. \tag{9}$$

This set is a *proper cone* in the precise sense of §2 — convex, a cone, closed, pointed, and solid — and that is exactly the licence the conic template (5) needs. Each property is a one-liner once you pick the right costume:

- **Convex.** Take $X, Y \succeq 0$ and a midpoint (more generally $\theta X + (1-\theta)Y$ with $\theta \in [0,1]$). Costume (i) settles it in one line: for any $v$,

$$v^{\mathsf{T}}\big(\theta X + (1-\theta)Y\big)v \;=\; \theta \underbrace{v^{\mathsf{T}} X v}_{\geq 0} + (1-\theta) \underbrace{v^{\mathsf{T}} Y v}_{\geq 0} \;\geq\; 0,$$

so the blend is PSD. A nonnegative combination of nonnegative numbers is nonnegative — that is the whole proof.

- **A cone.** If $X \succeq 0$ and $\alpha \geq 0$ then $v^\mathsf{T}(\alpha X)v = \alpha\, v^\mathsf{T}Xv \geq 0$, so $\alpha X \succeq 0$: scaling up or down (but not flipping sign) stays inside.
- **Closed.** The constraints $v^\mathsf{T}Xv \geq 0$ are non-strict inequalities that survive taking limits, so a convergent sequence of PSD matrices has a PSD limit. The boundary belongs to the set.
- **Pointed.** The cone contains no line: if both $X \succeq 0$ and $-X \succeq 0$ then every eigenvalue is simultaneously $\geq 0$ and $\leq 0$, forcing $X = 0$. The cone has a genuine tip at the origin.
- **Solid.** It has nonempty interior — and that interior is exactly the *positive definite* matrices, $X \succ 0$ (all eigenvalues strictly positive). Nudge a $\succ 0$ matrix in any direction and its eigenvalues, which were strictly positive, stay positive; so it has room around it.

One more fact we will use heavily in §7: the PSD cone is *self-dual*. Its dual cone — the set of matrices making a nonnegative inner product (7) with every member of $\mathbb{S}^n_+$ — is $\mathbb{S}^n_+$ itself. The orthant $\mathbb{R}^n_+$ shares this self-duality, and it is no coincidence: it is what makes the duality theory of §7 come out as cleanly for SDP as it does for LP. We defer the short proof to that section.

### The geometry: a curved cone, not a polyhedron

Here is where the PSD cone parts ways with the orthant. The orthant $\mathbb{R}^n_+$ is *polyhedral*: its boundary is made of flat faces (the coordinate hyperplanes $x_i = 0$) meeting at sharp corners. The PSD cone is different. Its boundary is the set of *singular* matrices,

$$\partial\mathbb{S}^n_+ \;=\; \{\, X \succeq 0 : \det X = 0 \,\},$$

the matrices where the smallest eigenvalue has just touched zero and the rank has dropped. As you move along this boundary, the rank changes smoothly and the surface *curves*. The PSD cone is decidedly not polyhedral — and that curvature is precisely the extra modelling power SDP buys over LP. Figure 6 shows the smallest interesting case.

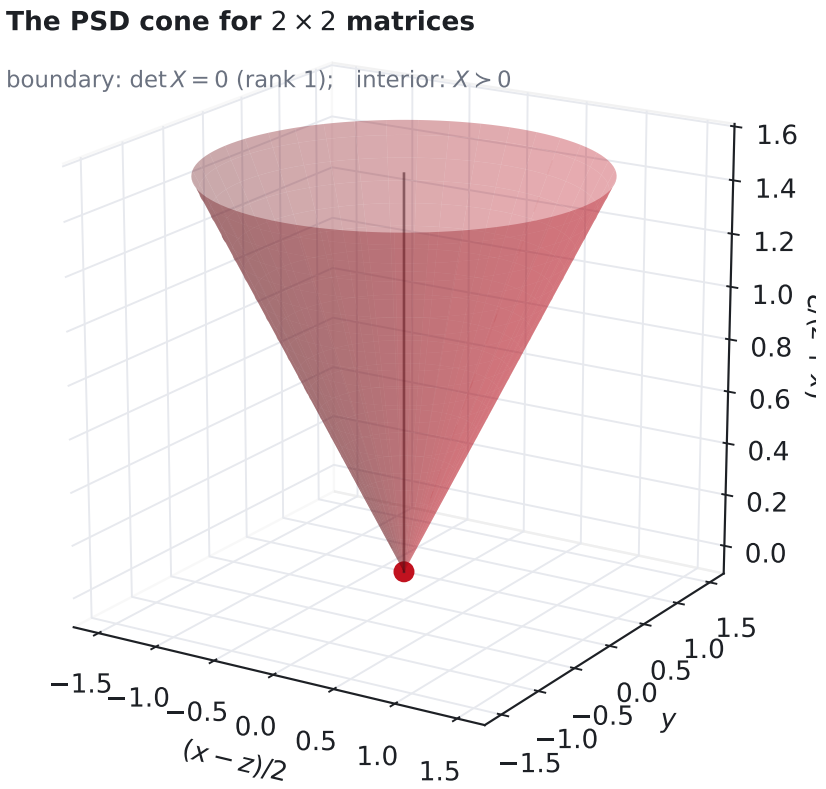


**Figure 6: The PSD cone for $2 \times 2$ matrices is a (rotated) ice-cream cone.** A symmetric $2 \times 2$ matrix $X = \left[\begin{smallmatrix} x & y \\ y & z \end{smallmatrix}\right]$ is PSD exactly when $x \geq 0,\ z \geq 0$, and $xz \geq y^2$. Under the linear change of coordinates $u = (x - z)/2,\ v = y,\ w = (x + z)/2$ this becomes $\sqrt{u^2 + v^2} \leq w$ — the second-order ("ice-cream") cone of §2, drawn here on the same $(u, v, w)$ axes. The curved boundary is $\det X = 0$ (the rank-1 matrices); the interior is $X \succ 0$.

The picture also lets us nail down where SDP sits on the ladder. Consider the generic $2 \times 2$ symmetric matrix and write out the test from costume (i)/(iv):

$$X = \begin{bmatrix} x & y \\ y & z \end{bmatrix} \succeq 0 \quad \Longleftrightarrow \quad x \geq 0, \quad z \geq 0, \quad xz \geq y^2. \tag{10}$$

Now apply the linear change of coordinates $u = (x - z)/2,\ v = y,\ w = (x + z)/2$. A small computation gives $xz = w^2 - u^2$, so the binding constraint $xz \geq y^2$ becomes $w^2 - u^2 \geq v^2$, i.e.

$$u^2 + v^2 \ \leq \ w^2, \qquad w \geq 0,$$

which is exactly the second-order (Lorentz, or "ice-cream") cone $\sqrt{u^2 + v^2} \leq w$ from §2. So already at $n = 2$, *the PSD cone is a rotated second-order cone*: the very same ice-cream shape, viewed in matrix coordinates. That is the bridge made concrete. For $n \geq 3$ the PSD cone is strictly richer — no longer expressible as a second-order cone — which is why the ladder

$$\text{LP} \ \subseteq \ \text{SOCP} \ \subseteq \ \text{SDP}$$

is a genuine hierarchy and not three names for one thing. The PSD cone is the roomiest of the three, and the most expressive convex set we can still optimise over efficiently.

**$X \succeq 0$ is the matrix version of "a number is $\geq 0$."** A real number passes $r \geq 0$ exactly when it is some square $r = s^2$; a symmetric matrix passes $X \succeq 0$ exactly when it is some "square" $X = B^{\mathsf{T}}B$. So $X \succeq 0$ is the statement that $X$ is a *valid covariance matrix*, a *sum of squares*, a *stored energy* — the matrix generalisation of nonnegativity, and the single ingredient that makes semidefinite programming work.

# 5 Semidefinite programs

A *semidefinite program* is what you get by dropping the PSD cone into the conic template (5). It comes in two standard dresses — a *primal* (or "standard") form and an *inequality* (or LMI) form — and the two are interchangeable. We give both, because each is the natural language for a different audience: the primal form is what a solver ingests, the LMI form is what a modeller writes down.

### Primal (standard) form

$$\min_{X \in \mathbb{S}^n} \ \langle C,\, X\rangle \quad \text{s.t.} \quad \langle A_i,\, X\rangle = b_i, \quad i = 1, \dots, m, \quad X \succeq 0. \tag{11}$$

Read piece by piece, this is gentle. The objective $\langle C, X\rangle = \operatorname{tr}(CX)$ is *linear* in the entries of $X$ (the matrix $C \in \mathbb{S}^n$ holds the cost of each entry). The $m$ equality constraints $\langle A_i, X\rangle = b_i$ are likewise linear — each fixed matrix $A_i \in \mathbb{S}^n$ and scalar $b_i$ pins down one weighted sum of entries. The only constraint that looks nonlinear is the last one, $X \succeq 0$; but as §4 showed, that is a *convex* cone constraint, not a nonlinearity to be afraid of. Compare (11) with the conic template (5): it *is* that template, with the variable promoted from a vector to a matrix and the cone $K$ taken to be $\mathbb{S}^n_+$. An LP is the same template with $K = \mathbb{R}^n_+$; an SDP simply swaps in the roomier cone.

### Inequality (LMI) form

$$\min_{x \in \mathbb{R}^k} \ c^{\mathsf{T}} x \quad \text{s.t.} \quad A_0 + x_1 A_1 + \cdots + x_k A_k \ \succeq 0, \tag{12}$$

where $c \in \mathbb{R}^k$ and the $A_0, \dots, A_k \in \mathbb{S}^n$ are fixed symmetric matrices. The variable is now an ordinary vector $x$, and the constraint is a *linear matrix inequality* (LMI): a matrix that depends *affinely* on $x$ — build it by starting at $A_0$ and adding $x_j A_j$ for each coordinate — is required to be PSD. The map $x \mapsto A_0 + \sum_j x_j A_j$ is the matrix-valued analogue of an affine function $x \mapsto A_0 + Ax$, and "$\succeq 0$" is the matrix-valued analogue of "$\geq 0$." LMIs are the modeller's natural language: control engineers, statisticians, and combinatorial optimisers tend to write their constraints *as* LMIs and only afterwards translate to standard form.

The two forms convert into each other, informally as follows. To go from LMI (12) to primal (11), treat the slack matrix $S = A_0 + \sum_j x_j A_j$ as the PSD variable and add equality constraints tying its entries to the affine combination of the $x_j$. To go the other way, take a matrix whose entries are free real parameters subject to the linear equations $\langle A_i, X\rangle = b_i$, solve those equations to express the feasible $X$ affinely in a smaller free vector $x$, and you are left with the single requirement $X(x) \succeq 0$ — an LMI. Either direction is bookkeeping; nothing is lost.

### Spectrahedra: feasible sets with curved faces

The feasible set of an LMI — the set of $x$ for which $A_0 + \sum_j x_j A_j \succeq 0$ — is called a *spectrahedron*. Geometrically it is a *slice of the PSD cone by an affine subspace*: take the curved cone $\mathbb{S}^n_+$ and intersect it with the flat plane traced out as $x$ ranges over $\mathbb{R}^k$. Because the cone is convex, every spectrahedron is convex; because the cone is curved, a spectrahedron inherits *curved faces*. This is the visible signature of SDP, and Figure 7 sets it beside an LP's polyhedron.

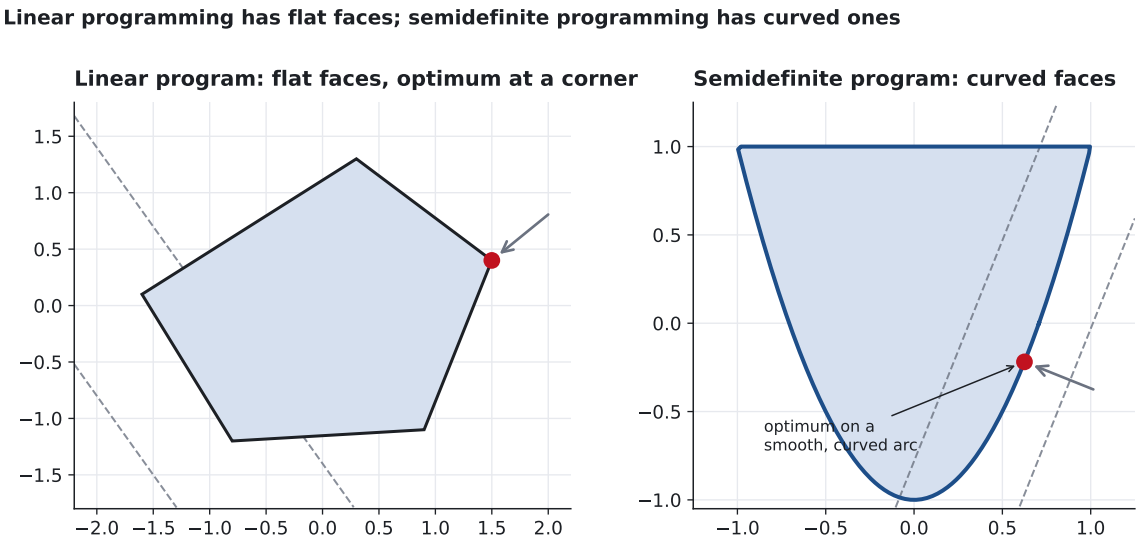


**Figure 7: Linear programming has flat faces; semidefinite programming has curved ones.** *Left:* an LP feasible set is a polyhedron — flat faces, sharp corners — and a linear objective is always optimised at a *vertex*. *Right:* a spectrahedron (the slice of the PSD cone cut by the LMI) has a smooth, curved boundary; sliding the same linear objective down, the optimum (near $(0.625, -0.219)$) comes to rest on a *smooth arc* of the boundary rather than at a corner. On that arc the boundary matrix is singular, so the optimal matrix is typically low-rank.

Two observations from this picture pay off later.

(a) **LP is literally an SDP.** Several LMIs stack into one by stuffing them onto a block diagonal: for symmetric blocks $M_1, \dots, M_r$,

$$\operatorname{diag}(M_1, \dots, M_r) \succeq 0 \quad \Longleftrightarrow \quad M_1 \succeq 0,\ \dots,\ M_r \succeq 0,$$

since a block-diagonal matrix is PSD exactly when each block is (its eigenvalues are the eigenvalues of the blocks). Specialise every block to a $1 \times 1$ scalar and the constraint becomes $\operatorname{diag}(d_1, \dots, d_r) \succeq 0 \iff d_i \geq 0$ for all $i$ — a list of ordinary inequalities. A *diagonal LMI is just an LP.* This is the rung-by-rung containment of the ladder made algebraic: LP sits inside SDP as the special case of diagonal matrices.

(b) **Optima are often low-rank.** On the curved part of the boundary the optimal $X$ has a zero eigenvalue, hence reduced rank. This is not an accident of the drawing; it is a structural feature that returns as a theorem in duality (§7) and as the reason SDP *relaxations* of hard combinatorial problems are so often nearly tight (§9).

## A tiny worked SDP

To watch the machinery turn, here is a fully worked $2 \times 2$ example in LMI form. Minimise $x$ subject to the single LMI

$$A(x) \;=\; \begin{bmatrix} x & 1 \\ 1 & 1 \end{bmatrix} \;\succeq 0.$$

Costume (iv) on a $2 \times 2$ matrix says $A(x) \succeq 0$ iff the diagonal entries are $\geq 0$ and the determinant is $\geq 0$: here $x \geq 0$, $1 \geq 0$ (automatic), and $\det A(x) = x \cdot 1 - 1 \cdot 1 = x - 1 \geq 0$. The binding requirement is $x \geq 1$, so the feasible set is the ray $[1, \infty)$ and the minimiser is

$$x^\star = 1, \qquad A(x^\star) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

which is singular ($\det = 0$) and rank 1 — the optimum sits exactly on the curved boundary of the cone, eigenvalue zero and all, just as Figure 7 predicted. A one-line example, but every feature of the general theory is already visible in it.

# 6 The modelling power: Schur complements and eigenvalue tricks

Why does SDP turn up in finance, control, statistics, combinatorics, and polynomial optimisation alike? Not because all those problems are "about matrices" — many are not, on their face. It is because a small kit of *tricks* converts constraints that look thoroughly non-SDP — ratios, inverses, squared norms, largest eigenvalues — into LMIs. Learn the kit and you start to see SDPs everywhere. Two tools do most of the work.

### The workhorse: the Schur complement

Suppose a symmetric matrix is partitioned into blocks $\left[\begin{smallmatrix} A & B \\ B^{\mathsf{T}} & C \end{smallmatrix}\right]$, with $A \succ 0$. The *Schur complement* of $A$ in this matrix is $C - B^{\mathsf{T}}A^{-1}B$, and the key identity is

$$\begin{bmatrix} A & B \\ B^{\mathsf{T}} & C \end{bmatrix} \succeq 0 \quad \Longleftrightarrow \quad C - B^{\mathsf{T}}A^{-1}B \ \succeq 0. \tag{13}$$

(The mechanism is block elimination: conjugating the left block matrix by the invertible $\left[\begin{smallmatrix} I & 0 \\ -B^{\mathsf{T}}A^{-1} & I \end{smallmatrix}\right]$ clears the off-diagonal blocks and leaves $\operatorname{diag}(A,\, C - B^{\mathsf{T}}A^{-1}B)$, which is PSD iff each block is, and $A \succ 0$ already.)

What does (13) *buy*? It *linearises* a constraint that hides an inverse or a quadratic. On the right of (13) sits a genuinely nonlinear object — $C - B^{\mathsf{T}}A^{-1}B$ involves an inverse and a product of the variables. On the left sits a *bigger* matrix whose every entry depends only *affinely* on the data. The Schur complement lets you trade the awkward nonlinear condition for an LMI in a larger matrix: you hide the nonlinearity inside extra dimensions, where it becomes linear. Figure 8 draws this as a single "lego" move.

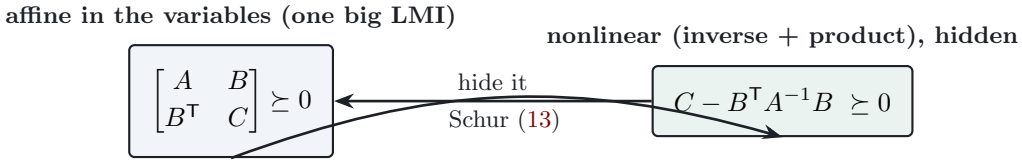


**Figure 8: The Schur complement as a lego move.** A nonlinear condition involving an inverse and a product of the variables (right, with $A \succ 0$) is *equivalent* to a single LMI in a larger block matrix whose entries are merely affine in the variables (left). You do not remove the nonlinearity; you pack it into extra dimensions where it reads as linear — which is exactly what a solver wants.

**A clean embedding.** Take the convex constraint

$$t \ \geq \ \|Bx - d\|_2^2,$$

which bounds a squared residual — the heart of least squares, ridge regression, and much of estimation. As written it is quadratic in $x$. Set $r = Bx - d$ (affine in $x$) and apply (13) with the choices $A = I$, $B \leftarrow r$, and $C = t$:

$$\begin{bmatrix} I & Bx - d \\ (Bx - d)^{\mathsf{T}} & t \end{bmatrix} \succeq 0 \quad \Longleftrightarrow \quad t - (Bx - d)^{\mathsf{T}}(Bx - d) \ \geq \ 0 \quad \Longleftrightarrow \quad t \geq \|Bx - d\|_2^2. \tag{14}$$

The left-hand matrix is affine in the variables $(x, t)$ — a bona fide LMI — so the quadratic constraint is now an SDP constraint. The same move with $t \geq x^{\mathsf{T}}x$ (take $B \leftarrow x$, $A = I$, $C = t$) shows the unit ball and every convex quadratic constraint sitting inside SDP. This is, concretely, *how* SOCP and convex-quadratic constraints embed into SDP — the ladder of §4 realised by a single $2 \times 2$ block trick.

### Eigenvalue optimisation

The second tool turns *spectral* objectives into LMIs. Recall from §2 that a pointwise maximum of linear functions is convex. The largest eigenvalue of a symmetric matrix has exactly that form — it is the Rayleigh quotient maximised over directions,

$$\lambda_{\max}(X) \;=\; \max_{\|v\|=1} v^{\mathsf{T}} X v, \tag{15}$$

a maximum of functions $v^{\mathsf{T}}Xv$ each of which is *linear* in $X$. So $\lambda_{\max}$, an upper envelope of linear pieces, is a convex function of $X$ — no calculation required, just the principle from §2. Figure 9 is this fact in one picture.

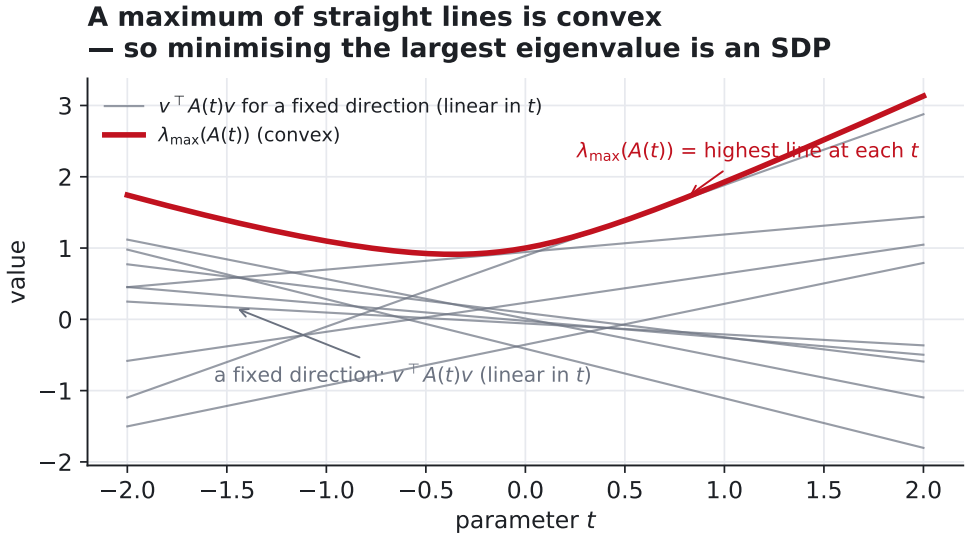


**Figure 9: The largest eigenvalue is a convex upper envelope.** Fix a direction $v$ and let a matrix $A(t)$ depend affinely on a parameter $t$; then $v^{\mathsf{T}}A(t)v$ is a *straight line* in $t$ (one thin grey line per direction). The largest eigenvalue $\lambda_{\max}(A(t))$ is, at each $t$, the *highest* of all these lines — their upper envelope (red). A maximum of straight lines is convex, which is why "minimise the largest eigenvalue" is a convex problem, in fact an SDP.

Convexity is one thing; an *SDP* is another. The bridge is a single equivalence. For any symmetric $Y$,

$$\lambda_{\max}(Y) \leq t \quad\Longleftrightarrow\quad \text{every eigenvalue of } Y \text{ is } \leq t \quad\Longleftrightarrow\quad tI - Y \;\succeq 0,$$

because the eigenvalues of $tI - Y$ are $t - \lambda_i(Y)$, all nonnegative exactly when the largest $\lambda_i(Y)$ does not exceed $t$. So if $A(x)$ is any matrix affine in a decision vector $x$, the problem "make the largest eigenvalue of $A(x)$ as small as possible" is the semidefinite program

$$\min_{x,\,t} \; t \quad \text{s.t.} \quad tI - A(x) \;\succeq 0. \tag{16}$$

The constraint is an LMI (affine in $(x, t)$), and the optimal $t$ is precisely $\min_x \lambda_{\max}(A(x))$. The same idea, with a sign flip, *maximises* the smallest eigenvalue ($A(x) - tI \succeq 0$, push $t$ up);

stacking both controls the *spread* $\lambda_{\max} - \lambda_{\min}$ and so tames condition-number-like objectives; and applying it to $\left[\begin{smallmatrix} tI & M \\ M^{\mathsf{T}} & tI \end{smallmatrix}\right]$ captures the operator (spectral) norm of a non-symmetric $M$, with a nuclear-norm analogue close behind. Each is the same template: bound a spectral quantity, read it off as an LMI.

### Why SDP is a Swiss-army knife

Put the two tools together and the breadth of SDP stops being mysterious. The Schur complement absorbs inverses, ratios, and quadratics; the eigenvalue/LMI construction absorbs spectral objectives and matrix norms. Between them they reach least squares, control (the Lyapunov inequalities of §9), experiment design, robust optimisation, polynomial and combinatorial relaxations — a span no single other model class covers. That combination, Schur complements plus eigenvalue modelling on top of the one curved cone of §4, is what makes the semidefinite program the Swiss-army knife of convex optimisation.

## 7 Duality, or how an SDP proves it is right

In §1 we promised that an SDP does not merely return an answer but hands you a *certificate* — a short, independently checkable proof that the answer is correct. That promise is the payoff of convexity, and the machinery that delivers it is *duality*. Here is the whole idea in one sentence before any symbols: *any way of relaxing a minimisation gives a lower bound on the true minimum, and the best such bound is itself an optimisation problem — the* dual.

Why does that matter? A lower bound is exactly what a minimiser cannot argue with. If you are trying to make $\langle C,\, X\rangle$ as small as possible and somebody proves it can never go below the value $\beta$, then any feasible $X$ achieving $\langle C,\, X\rangle = \beta$ is provably optimal — no further search required. Duality is the systematic way to manufacture the largest, tightest such $\beta$.

**The dual problem.** Recall the primal SDP (11): minimise $\langle C,\, X\rangle$ over $X \succeq 0$ subject to the $m$ linear equalities $\langle A_i,\, X\rangle = b_i$. Its dual — derived in full in Appendix B — is

$$\max_{y\in\mathbb{R}^m,\; S\in\mathbb{S}^n} \; b^{\mathsf{T}} y \quad \text{s.t.} \quad \sum_{i=1}^{m} y_i A_i + S = C, \quad S \succeq 0. \tag{17}$$

The vector $y \in \mathbb{R}^m$ carries one multiplier per equality constraint; the matrix $S \in \mathbb{S}^n$ is the *dual slack*. The equality constraint is just a definition of $S$, so (17) says equivalently

$$\text{maximise } b^{\mathsf{T}} y \quad \text{subject to} \quad C - \sum_{i=1}^{m} y_i A_i \;\succeq 0\; 0.$$

Notice the shape: the primal optimises over a *matrix* constrained to a cone with *scalar* equalities, the dual optimises over a *vector* with a single *matrix* cone constraint. They are mirror images.

**Weak duality — one line, and it is the whole game.** Take *any* primal-feasible $X$ and *any* dual-feasible $(y, S)$. Then

$$\langle C,\, X\rangle - b^{\mathsf{T}} y \;=\; \langle X,\, S\rangle \;\geq\; 0. \tag{18}$$

The equality is pure bookkeeping: substitute $C = \sum_i y_i A_i + S$ and use $\langle A_i,\, X\rangle = b_i$,

$$\langle C,\, X\rangle = \Big\langle \textstyle\sum_i y_i A_i + S,\, X \Big\rangle = \sum_i y_i \langle A_i,\, X\rangle + \langle S,\, X\rangle = b^{\mathsf{T}} y + \langle X,\, S\rangle.$$

The inequality is the one fact we have been saving up: $X \succeq 0$ and $S \succeq 0$ force $\langle X, S\rangle = \operatorname{tr}(XS) \geq 0$, because the PSD cone is *self-dual* (§4) — the inner product of two PSD matrices is never negative.

That single line is the certificate. Every dual-feasible $(y, S)$ produces a number $b^{\mathsf{T}}y$ that is *provably* a lower bound on the primal optimum, and the nonnegative quantity

$$\langle X, S\rangle \;=\; \langle C, X\rangle - b^{\mathsf{T}}y$$

— the *duality gap* — measures exactly how far the pair $(X, (y, S))$ is from optimal. Shrink the gap to zero and you have squeezed the true answer between an upper bound (the primal value) and a matching lower bound (the dual value).

**What you actually check.** This is the practical heart of the section. To be *convinced* that a reported optimum is correct you do not need to trust the solver, re-run it, or understand its internals. You need only verify three cheap facts about the numbers it returns:

(a) $X$ is *primal-feasible*: $X \succeq 0$ and $\langle A_i, X\rangle = b_i$;

(b) $(y, S)$ is *dual-feasible*: $S = C - \sum_i y_i A_i \succeq 0$;

(c) the two objective values *coincide*: $\langle C, X\rangle = b^{\mathsf{T}}y$.

Each is a matter of a few matrix multiplications and one positive-semidefiniteness test (a Cholesky factorisation). Pass all three and weak duality (18) guarantees optimality — there is no room left for a better solution to hide. Figure 10 shows the two streams of bounds closing on the answer from above and below.

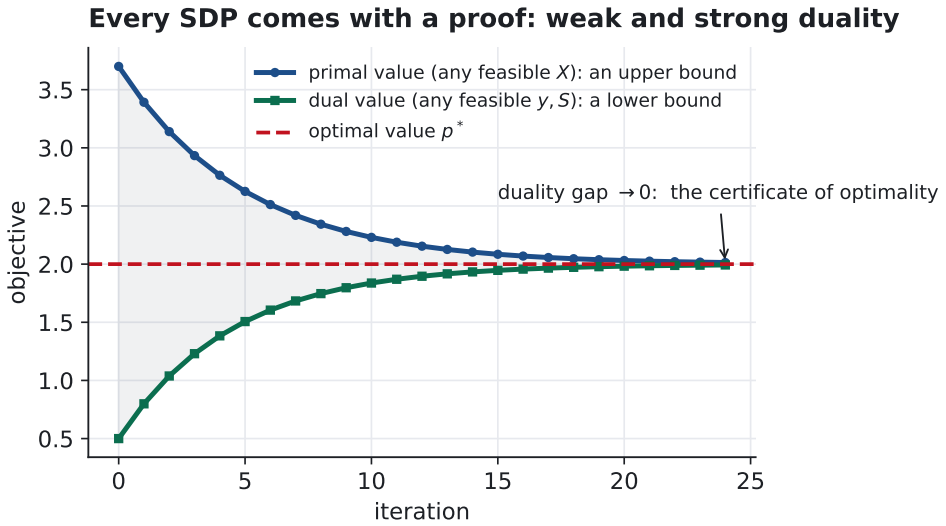


**Figure 10: The certificate, as two converging bounds.** Primal objective values (blue) descend as upper bounds; dual objective values (green) ascend as lower bounds; they meet at the red dashed optimal value. The shaded region is the duality gap $\langle X, S\rangle$ — when it closes, the answer is certified.

**Strong duality, and a real caveat.** Weak duality is free and unconditional. The deeper question is whether the best lower bound actually *reaches* the optimum — whether the gap can be driven all the way to zero. It can, provided the problem is not pathologically tight against its cone. The clean sufficient condition is *Slater's condition*: there exists a strictly feasible point — some $X \succ 0$ (not merely $\succeq 0$) meeting the equalities, or symmetrically some $S \succ 0$ in the dual. Under Slater's condition strong duality holds: the gap is zero *and* both optima are attained.

Here SDP differs from linear programming, and it is worth one sentence of candour. In an LP the gap is always zero at finite optima; in an SDP, when Slater fails, you can encounter

a strictly positive duality gap, or an optimal value that is approached but never attained [2]. These are edge cases — almost every SDP that arises in modelling is strictly feasible — but a careful reader should know the guarantee is conditional.

**Complementary slackness and why optima are low-rank.** When the gap does close, $\langle X, S\rangle = \operatorname{tr}(XS) = 0$ with both $X \succeq 0$ and $S \succeq 0$. For PSD matrices this forces not just the trace but the *product* to vanish: $XS = 0$. (Write $X = \sum_k \lambda_k u_k u_k^{\mathsf{T}}$ with $\lambda_k > 0$; then $0 = \operatorname{tr}(XS) = \sum_k \lambda_k\, u_k^{\mathsf{T}} S u_k$, and each term is nonnegative, so $Su_k = 0$ for every $k$ — the range of $X$ lies in the kernel of $S$.) The ranges of $X$ and $S$ are therefore orthogonal, which gives the rank bound

$$\operatorname{rank} X + \operatorname{rank} S \;\le\; n.$$

This is the structural reason, promised in §5, that SDP optima are so often *low-rank*: the dual slack $S$ at the optimum typically has high rank, and it crowds the primal solution $X$ into the small complementary subspace. Many SDP relaxations succeed precisely because this forces a rank-one $X$ — a genuine, not merely relaxed, solution.

**A certificate is cheaper than trust.** Duality turns "the solver says so" into "here is a proof you can check by hand." A dual-feasible $(y, S)$ is a short witness whose only claim — $S \succeq 0$ — is verified by one Cholesky factorisation, and whose value $b^{\mathsf{T}} y$ no feasible $X$ can undercut. You verify the answer without redoing the work, the same way checking a multiplication is easier than performing it.

## 8 How the solver actually does it

Knowing that a certificate *exists* is one thing; producing the optimal $X$ and its matching $(y, S)$ is another. Modern SDP solvers do it with *interior-point methods*, and the idea is far less mysterious than its reputation. Everything follows from one trick for staying inside the PSD cone.

**The barrier.** The constraint $X \succeq 0$ is awkward because it has a boundary — the singular matrices, where some eigenvalue hits zero. Newton's method loves smooth unconstrained problems and hates hard boundaries. So we replace the hard wall with a soft one: add to the objective a penalty that is gentle deep inside the cone but blows up as $X$ approaches the boundary. The canonical choice is the *log-determinant barrier*

$$\phi(X) \;=\; -\log\det X. \tag{19}$$

As any eigenvalue of $X$ tends to 0, $\det X \to 0^+$ and $\phi(X) \to +\infty$ — the penalty fences the iterate away from the singular boundary. It is the matrix analogue of $-\sum_i \log x_i$ for the nonnegative orthant, and it is *self-concordant*, the technical property that makes Newton's method provably efficient on it [4]. Its gradient is $\nabla\phi(X) = -X^{-1}$, which itself explodes as any eigenvalue $\to 0$.

**The central path.** Now trade the hard constraint for the barrier and sweep a knob. For each $\mu > 0$ define $X(\mu)$ as the minimiser of

$$\langle C, X\rangle + \mu\,\phi(X) \qquad \text{subject to the linear equalities } \langle A_i, X\rangle = b_i.$$

For large $\mu$ the barrier dominates and $X(\mu)$ sits near the *analytic centre* of the feasible set, comfortably interior. As $\mu \downarrow 0$ the penalty fades and the true objective takes over, so $X(\mu)$ slides toward the optimum. The locus $\{X(\mu) : \mu > 0\}$ is a smooth curve — the *central path* — running from the centre to the answer (Figure 11). The solver's job is simply to follow this path down to small $\mu$.

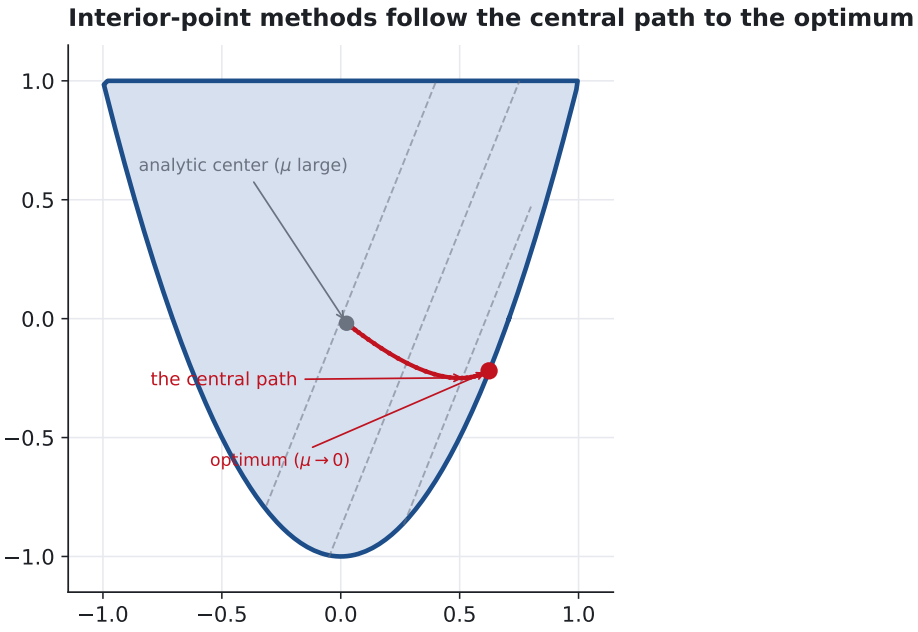


**Figure 11: The central path.** Inside the teardrop spectrahedron, the red central path runs from the analytic centre (large $\mu$) to the optimum on the boundary ($\mu \to 0$). Dashed lines are objective level sets; the path stays strictly interior until the very end.

**Primal–dual methods.** The fastest solvers do not follow the primal path alone; they track *both* $X$ and $(y, S)$ at once. The optimality (KKT) conditions for the central path read: primal feasibility, dual feasibility, and a *relaxed complementarity*

$$XS = \mu I,$$

which softens the optimality condition $XS = 0$ from §7 by the amount $\mu$. A *primal–dual interior-point method* applies Newton's method to this system of equations, taking a step in $(X, y, S)$, then decreases $\mu$ and repeats. As $\mu \to 0$ the relaxed condition $XS = \mu I$ tightens to $XS = 0$ — the solver lands on a primal $X$ and a dual $(y, S)$ that simultaneously satisfy the three certificate checks of §7. The certificate is a *by-product* of the algorithm, not an afterthought.

**Costs and limits, concretely.** The remarkable fact, the one that makes interior-point methods practical, is that the path is short. Following it to accuracy $\epsilon$ takes only

$$O\big(\sqrt{n}\,\log(1/\epsilon)\big)$$

Newton iterations in theory [4], and in practice a few tens of iterations regardless of problem size. The cost lives *inside* each iteration: forming and solving a dense $m \times m$ linear system (the *Schur complement* system) for the search direction, at roughly

$$O\big(m^2 n^2 + n^3\big)$$

work per iteration. So dense SDPs with matrix size $n$ and constraint count $m$ in the low thousands are comfortable on a laptop. Beyond that the dense Schur system becomes the bottleneck — its $O(m^2)$ storage and $O(m^3)$ factorisation do not scale — and you should reach for *first-order methods* instead: operator-splitting / ADMM schemes [10] whose iterations are cheap, dominated by a single eigenvalue decomposition to *project* onto the PSD cone. The trade-off is plain: interior-point methods give high accuracy at modest scale in few iterations; first-order methods scale to far larger problems but converge slowly and deliver lower accuracy. Pick by which corner you are in.

Algorithm 1 sketches the simplest path-follower — a primal barrier method with the knob written as $t = 1/\mu$, so $t$ *increases* as we descend the path. (Appendix C turns it into running code.)

**input:** strictly feasible $X \succ 0$, parameter $t > 0$, factor $\mu > 1$, tolerance $\epsilon$
**repeat**
  compute $X(t)$ minimising $t\,\langle C,\, X\rangle + \phi(X)$ over the equalities, ▷ *inner: damped Newton*
  starting from current $X$, by damped Newton steps;
  at each step keep $X \succ 0$ via a backtracking line search; ▷ *stay interior*
  set $X \leftarrow X(t)$;
  **if** $n/t < \epsilon$ **then return** $X$; ▷ *gap $n/t$ certified small*
  $t \leftarrow \mu\, t$; ▷ *tighten the barrier*
**until** converged

**Algorithm 1:** Barrier (central-path) method for an SDP

The stopping test deserves a word: at the central point $X(t)$ the duality gap is exactly $n/t$, so $n/t < \epsilon$ is a genuine, certified bound on suboptimality — the same certificate idea, now driving the termination logic.

**Push to the wall, but never touch it.** The barrier lets a smooth, unconstrained method respect a hard cone constraint: it feels an outward force that diverges at the boundary, so it can press arbitrarily close to the optimal face without ever leaving the cone. Cranking $t$ up weakens the force step by step, and the iterate walks down the central path to the corner where the answer lives.

# 9 Worked examples in the wild

Theory earns its keep when you watch it do work. The three examples below are not variations on one theme; they show three genuinely *different* ways an SDP earns a place in a practitioner's toolkit. The first is a *projection*: we have a matrix that ought to be positive semidefinite but is not, and we want the nearest one that is—a direct shove onto the PSD cone of §4. The second is a *feasibility* question: we do not want to minimise anything, only to find a single matrix that exists if and only if a physical system is stable—a certificate, written as a linear matrix inequality. The third is a *relaxation*: we take a hard discrete problem, blow each $\pm 1$ decision up into a unit vector, and let the SDP hand back not just a good candidate answer but a *bound* on how good the true answer could ever be. Projection, certification, relaxation—keep those three words in mind as we go.

## 9.1 The nearest correlation matrix

A correlation matrix is not just any symmetric matrix with ones on the diagonal. It is a *Gram matrix* of standardised random quantities—costume (iii) of the PSD cone from §4—and so it *must* be positive semidefinite. The reason is not a convention but a law: for any weights $w \in \mathbb{R}^n$, the variance of the combined position $\sum_i w_i Z_i$ is $w^{\mathsf{T}} X w$, and a variance cannot be negative. A symmetric unit-diagonal matrix with $w^{\mathsf{T}} X w < 0$ for some $w$ is therefore claiming that some portfolio has negative variance—impossible. Such a matrix is not a correlation matrix at all; it is an impostor.

**A non-PSD "correlation" matrix is a contradiction.** Each negative eigenvalue is a direction $w$ in which the data say variance is below zero. The matrix is internally inconsistent, and any computation that trusts it—a Cholesky factor, a simulation, a risk number—inherits the contradiction.

The trouble is that, in practice, correlation matrices are rarely measured as a single coherent object. They are *assembled*: pairwise correlations stitched together from different time windows, different data vendors, or hand-edited stress scenarios ("what if these two assets suddenly moved together?"). The result is symmetric with unit diagonal but, almost always, slightly *indefinite*—it has a few small negative eigenvalues. We need the nearest genuine correlation matrix. Formally, given the assembled $G \in \mathbb{S}^n$, solve

$$\min_{X \in \mathbb{S}^n} \|X - G\|_F \quad \text{subject to} \quad X \succeq 0, \quad X_{ii} = 1 \quad (i = 1, \ldots, n), \tag{20}$$

where $\|\cdot\|_F$ is the Frobenius norm of Appendix A. This is a projection: it finds the closest point of the feasible set—the intersection of the PSD cone with the affine set of unit-diagonal matrices, the *elliptope* we will meet again below—to the given $G$.

The key building block is the projection onto the PSD cone *alone*, ignoring the diagonal. It has a beautifully simple answer: *clip the negative eigenvalues to zero*. Eigendecompose the symmetric matrix as $G = \sum_i \lambda_i \, q_i q_i^{\mathsf{T}}$ (spectral theorem, Appendix A) and set

$$\Pi_{\succeq 0}(G) \;=\; \sum_{i:\, \lambda_i > 0} \lambda_i \, q_i q_i^{\mathsf{T}} \;=\; \sum_i \max(\lambda_i, 0)\, q_i q_i^{\mathsf{T}}. \tag{21}$$

Why is this the nearest PSD matrix? Because the Frobenius norm is invariant under the orthogonal change of basis $Q$, so in the eigenbasis the problem decouples into $n$ independent scalar problems: find the nearest *nonnegative* number to each $\lambda_i$. The nearest nonnegative number to a negative one is 0, and to a positive one is itself—which is exactly (21). Two lines, and we have projected onto a curved, infinite-dimensional-looking cone.

Clipping fixes definiteness but disturbs the diagonal: the clipped matrix no longer has exact ones down its diagonal. Restoring the diagonal, in turn, disturbs definiteness. Higham's insight [7] is that we can simply *alternate*—project onto the PSD cone, then onto the unit-diagonal set (reset the diagonal to one), then back—and, with a small correction term (Dykstra's), the iterates converge to the true solution of (20). It is the geometric picture of bouncing between two convex sets until you land in their intersection, and it is the workhorse behind "fix my correlation matrix" routines in risk systems everywhere.

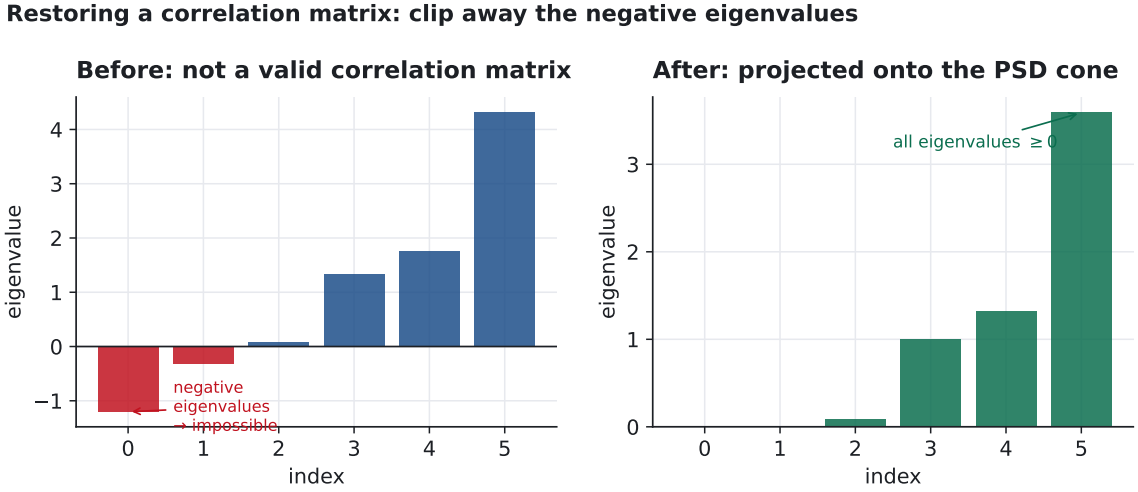


**Figure 12: Repairing an impostor by clipping eigenvalues.** *Left:* the eigenvalues of an assembled "correlation" matrix; several are negative, so the matrix is indefinite and cannot be a real correlation matrix—it implies a combination with negative variance. *Right:* after projecting onto the PSD cone via (21), the negatives are clipped to zero and every eigenvalue is $\geq 0$. This single projection is the engine inside Higham's alternating scheme for (20).

This example is, almost literally, "projection onto the PSD cone" from §4 made into a daily tool. The curved boundary of the cone is exactly the locus the clipped matrix lands on: with a zero eigenvalue, the repaired matrix sits on the cone's edge, low-rank and stretched as tight as the data allow.

### 9.2 Lyapunov stability of a linear system

Now a question with no objective at all. A continuous-time linear system

$$\dot{x} \;=\; Ax, \qquad x(t) \in \mathbb{R}^n,\; A \in \mathbb{R}^{n\times n},$$

is *stable* if every trajectory decays to the origin: $x(t) \to 0$ as $t \to \infty$, from any starting point. There are infinitely many trajectories, one per initial condition, so checking them one by one is hopeless. How do we certify stability of the *whole* flow with a finite, convex computation?

The classical answer is Lyapunov's, and it is an SDP in disguise. The system is stable if and only if there exists a matrix $P \succ 0$ satisfying the *Lyapunov inequality*

$$A^{\mathsf{T}}P + PA \;\prec\; 0 \qquad (\text{equivalently } A^{\mathsf{T}}P + PA \preceq 0 \text{ with } P \succ 0). \tag{22}$$

Read (22) carefully: the unknown is the *matrix* $P$, and both $P \succ 0$ and $A^{\mathsf{T}}P + PA \prec 0$ are linear matrix inequalities in its entries. This is a pure *feasibility* SDP—a semidefinite program with no cost function, asking only whether the spectrahedron of §5 carved out by these two LMIs is nonempty. The solver either returns a feasible $P$ (a certificate of stability) or a dual certificate that none exists.

The intuition is energy. Define $V(x) = x^{\mathsf{T}}Px$. Because $P \succ 0$, this is a *bowl*: strictly positive everywhere except at the origin, where it bottoms out. Differentiate along a trajectory:

$$\dot{V} \;=\; \dot{x}^{\mathsf{T}}Px + x^{\mathsf{T}}P\dot{x} \;=\; x^{\mathsf{T}}\big(A^{\mathsf{T}}P + PA\big)\,x \;<\; 0.$$

The condition $A^{\mathsf{T}}P + PA \prec 0$ says precisely that the energy is *always* falling, in every state $x \neq 0$. A trajectory can only ever run *downhill* in the bowl, so it must spiral into the bottom—the origin. The level sets $V(x) = c$ are nested ellipsoids, and stability means the flow always crosses them inward (Figure 13).

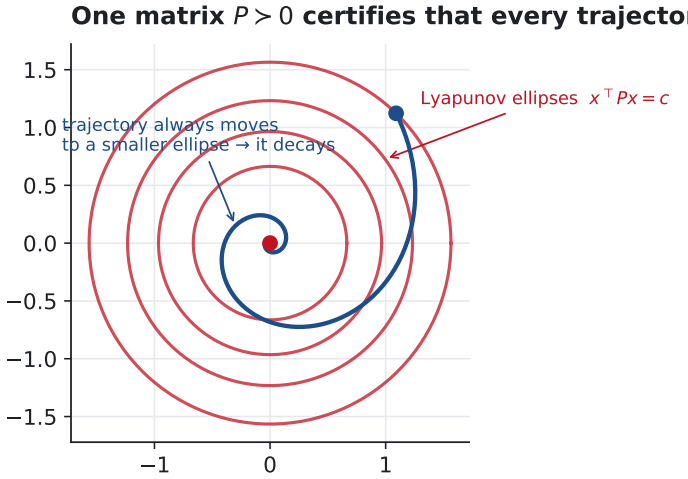


**Figure 13: A Lyapunov bowl the dynamics cannot climb.** The nested ellipses are level sets $x^{\mathsf{T}}Px = c$ of the energy $V$. A trajectory of $\dot{x} = Ax$ crosses each one strictly inward, because $\dot{V} = x^{\mathsf{T}}(A^{\mathsf{T}}P + PA)x < 0$ everywhere, and spirals into the origin. Finding the matrix $P$ that makes such a bowl exist is the feasibility SDP (22).

The payoff is worth dwelling on. A statement about an *infinite* family—all trajectories of the system—has collapsed into a *finite, convex* feasibility check on a single matrix. And the template scales up gracefully. If $A$ is not known exactly but lives in a family (an uncertain or time-varying plant), we ask for one $P$ that works for the *whole* family—robust stability, still an LMI feasibility problem. If we are allowed to choose a feedback law, the search for $P$ and the controller becomes a (suitably reparametrised) SDP too. This is why linear matrix inequalities are a cornerstone of modern control [1].

### 9.3 Max-Cut and the Goemans–Williamson relaxation

The headline use of SDP: as the tightest *tractable* relaxation of a problem that is genuinely hard. *Max-Cut* asks us to split the vertices of a weighted graph into two groups so as to maximise the total weight of the edges that cross between them. Assign each vertex a sign $y_i \in \{-1, +1\}$ for its side. An edge $(i, j)$ is cut exactly when $y_i \neq y_j$, i.e. when $y_i y_j = -1$, so the quantity $\frac{1}{2}(1 - y_i y_j)$ is 1 for a cut edge and 0 otherwise. The problem is

$$\max_{y \in \{-1,+1\}^n} \ \tfrac{1}{2} \sum_{i<j} w_{ij} \, (1 - y_i y_j).$$

This is NP-hard: the $\{-1, +1\}$ constraint is the whole difficulty, and brute force costs $2^n$.

Here is the relaxing move. Replace each scalar sign $y_i \in \{-1, +1\}$ by a *unit vector* $v_i \in \mathbb{R}^n$ on the sphere, and replace the product $y_i y_j$ by the inner product $v_i^{\mathsf{T}} v_j$. Two old signs that agreed ($y_i y_j = +1$) become two parallel vectors; two that disagreed become antiparallel; but now the vectors are free to point anywhere in between. Collect the inner products into a matrix $X_{ij} = v_i^{\mathsf{T}} v_j$. By costume (iii) again, $X$ is a Gram matrix, hence $X \succeq 0$, and since each $v_i$ is a unit vector, $X_{ii} = 1$. We have arrived back at the *elliptope*, and the relaxed problem is the SDP

$$\max_{X \in \mathbb{S}^n} \ \tfrac{1}{2} \sum_{i<j} w_{ij} \, (1 - X_{ij}) \qquad \text{subject to} \qquad X \succeq 0, \quad X_{ii} = 1. \tag{23}$$

The connection to §5 is exact and instructive. If we *forced* $X$ to have rank one, $X = yy^{\mathsf{T}}$ with $y_i = \pm 1$, then $X_{ij} = y_i y_j$ and (23) would be the original Max-Cut, untouched. The relaxation is precisely the act of *dropping the rank-one constraint* and letting $X$ climb to higher rank. Since we have enlarged the feasible set, the SDP optimum is an *upper bound* on the true maximum cut—we can only do better with more freedom.

That leaves the question of turning a high-rank $X$ back into a genuine $\pm 1$ assignment. The Goemans–Williamson *rounding* is as elegant as the relaxation. Solve the SDP; factor the optimal $X = V^{\mathsf{T}} V$ to recover the unit vectors $v_i$ (the columns of $V$); pick a *random hyperplane* through the origin—equivalently, a random direction $r$ on the sphere—and set $y_i = \operatorname{sign}(r^{\mathsf{T}} v_i)$, sorting each vertex by which side of the hyperplane its vector falls on. Vectors that the SDP pushed far apart are likely to be separated; vectors it kept close are likely to stay together.

The miracle is the guarantee. Goemans and Williamson proved that the expected weight of this random cut is at least

$$\alpha_{\mathrm{GW}} \ = \ \min_{0 \le \theta \le \pi} \frac{2}{\pi} \, \frac{\theta}{1 - \cos\theta} \ \approx \ 0.878$$

times the SDP optimum, hence at least 0.878 times the true Max-Cut [6]—the celebrated 0.878 approximation, the first of its kind and still a landmark. The probability that a random hyperplane separates $v_i$ and $v_j$ is just their angle over $\pi$, and comparing that angle-based separation to the relaxed objective term by term yields the constant.

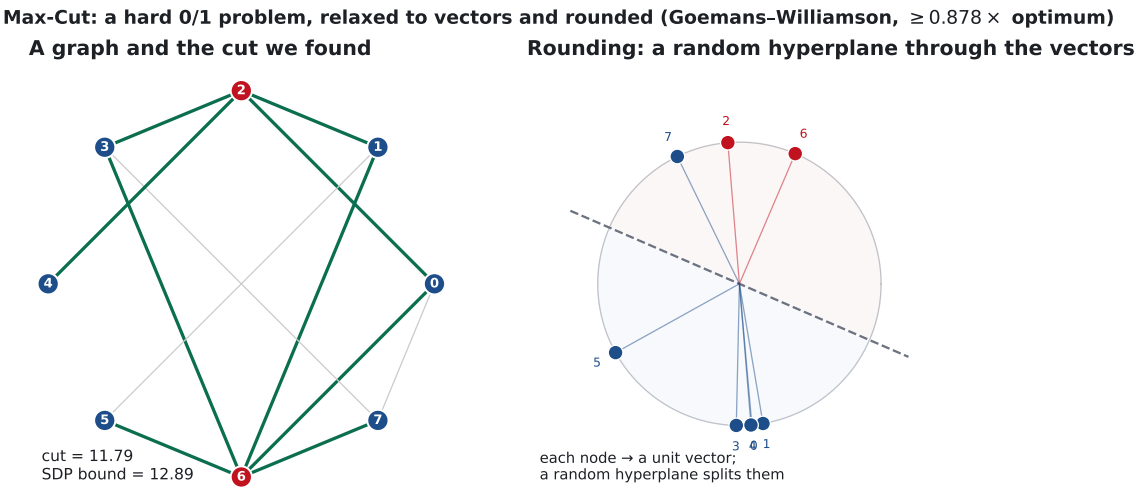


**Figure 14: From graph, to vectors, to a cut.** *Left:* an eight-node weighted graph; nodes are coloured by the side they land on and the cut edges crossing between the two groups are highlighted. *Right:* the SDP solution as unit vectors $v_i$ on the circle; a random hyperplane through the origin splits them into the two sides. On this instance the SDP gives an upper bound near 12.9 while the rounded cut reaches about 11.8—comfortably inside the 0.878 guarantee.

**A relaxation gives you a candidate *and* a yardstick.** The rounded cut is a concrete feasible answer; the SDP optimum is a provable upper bound on *any* answer. Their ratio tells you, for the very instance in front of you, how much room is left—a luxury most heuristics for hard problems simply cannot offer.

## 10 What semidefinite programming cannot (yet) do

A candid closing is more useful than a triumphant one. SDP is powerful, but it has edges, and knowing them is part of using it well.

**The scaling wall.** Interior-point methods (§8) solve SDPs to high accuracy in remarkably few iterations, but each iteration forms and factors a dense linear system whose size grows with the matrix dimension and the number of constraints. In practice, dense SDPs start to strain memory and time once the matrix variable or the constraint count climbs past the low thousands. The PSD cone is the most expressive convex set we can optimise over efficiently—but *efficiently* has a ceiling. *First-order* methods (operator splitting, the alternating-direction style of §8) push the size further by avoiding the big factorisation, at the cost of slower, lower-accuracy convergence. The right tool depends on whether you need six digits or three.

**Not everything is natively an SDP—but the reach is vast.** Some convex problems do not wear PSD clothing naturally, and forcing them into an SDP is wasteful. Yet the territory SDP can *reach* is astonishing. The *sum-of-squares* viewpoint and the *Lasserre hierarchy* [8, 9] attack *polynomial* optimisation—a huge, thoroughly nonconvex class—with a tower of SDPs of growing size. The seed idea is disarmingly simple: a polynomial that can be written as a sum of squares of other polynomials is obviously nonnegative everywhere, and the question "is this polynomial a sum of squares?" is itself an SDP (it asks for a PSD matrix of coefficients in a monomial basis). Climbing the hierarchy—allowing higher-degree certificates—tightens the approximation, often to exactness, but the SDP grows quickly with each rung. It is real power at a real price.

**Choose the right rung of the ladder.** The conic ladder of this note runs LP $\subset$ SOCP $\subset$ SDP, each rung strictly more expressive and strictly more expensive than the last. The discipline is to climb *only as far as you must.* If your constraints are linear, use a linear program—nothing is faster. If you have Euclidean norms or convex quadratics, a second-order cone program fits them snugly. Reach for an SDP when you genuinely have *eigenvalue* objectives, *matrix-inequality* constraints, or a *relaxation* of discrete structure of the kind we have just seen. Modelling a plain LP as an SDP is not clever; it is a tax you pay for nothing.

**Outlook.** None of these are walls so much as frontiers. Solver scaling improves year on year, the sum-of-squares machinery keeps finding new application domains, and the modelling vocabulary—"where does this fit on the ladder?"—is a durable skill that outlasts any one solver. If you can recognise PSD structure when it appears, you can borrow four decades of beautiful theory and fast software the moment you need them.

# A Linear algebra refresher

This appendix collects the handful of linear-algebra facts the body leans on. It is a refresher, not a course—just enough to keep every symbol grounded.

**Symmetric matrices.** We write $\mathbb{S}^n$ for the set of real symmetric $n \times n$ matrices, those with $X = X^\mathsf{T}$. They are the natural home for semidefinite programming for two reasons. First, the definiteness of a matrix is really a statement about its symmetric part: the quadratic form $v^\mathsf{T} X v$ that the PSD condition constrains depends only on $\frac{1}{2}(X + X^\mathsf{T})$, so we lose nothing by restricting to symmetric $X$. Second, $\mathbb{S}^n$ is itself a vector space—of dimension $\binom{n+1}{2}$—and the PSD cone lives inside it as a convex cone. Everything happens in this one tidy space.

**The spectral theorem.** The central fact. Every symmetric $X \in \mathbb{S}^n$ can be written

$$X \;=\; Q\,\Lambda\,Q^\mathsf{T} \;=\; \sum_{i=1}^{n} \lambda_i\, q_i q_i^\mathsf{T}, \tag{24}$$

where $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ holds the *eigenvalues* (all *real*) and the columns $q_i$ of $Q$ are *orthonormal* eigenvectors, so $Q^\mathsf{T} Q = I$. Geometrically, a symmetric matrix is just a set of orthogonal axes (the $q_i$) with a real stretch factor $\lambda_i$ along each. This is what the constraint "$X \succeq 0$" really controls: $X \succeq 0$ holds if and only if *every* eigenvalue $\lambda_i \geq 0$, and $X \succ 0$ if and only if every $\lambda_i > 0$. Definiteness is a fact about the spectrum, nothing more—which is why eigenvalue clipping (21) is exactly projection onto the cone.

**The trace inner product.** To do geometry on $\mathbb{S}^n$ we need an inner product, and the natural one is the *trace inner product*

$$\langle A,\, B\rangle \;=\; \operatorname{tr}(A^\mathsf{T} B) \;=\; \sum_{i,j} A_{ij} B_{ij}. \tag{25}$$

It is nothing exotic: it treats each matrix as a long vector of its entries and takes the ordinary dot product. For *symmetric* matrices, $A^\mathsf{T} = A$, so $\langle A,\, B\rangle = \operatorname{tr}(AB)$—a small convenience the body uses freely. The induced norm is the *Frobenius norm*

$$\|X\|_F \;=\; \sqrt{\langle X,\, X\rangle} \;=\; \Big(\textstyle\sum_{i,j} X_{ij}^2\Big)^{1/2},$$

the straight-line (Euclidean) distance in matrix space. It is the notion of "nearest" minimised in the nearest-correlation-matrix problem (20), and because the trace is invariant under orthogonal conjugation, $\|X\|_F^2 = \sum_i \lambda_i^2$—the Frobenius norm is just the Euclidean length of the spectrum, which is exactly why the projection (21) decouples eigenvalue by eigenvalue.

**Quadratic forms and the Rayleigh quotient.** The quadratic form $v^\mathsf{T} X v$ is the single number that ties matrices to definiteness. Substituting the spectral decomposition (24) and writing $c = Q^\mathsf{T} v$ for the coordinates of $v$ in the eigenbasis gives

$$v^\mathsf{T} X v \;=\; \sum_{i=1}^{n} \lambda_i\, c_i^2.$$

A quadratic form is just a $\lambda$-weighted sum of squares in the eigenbasis. On the unit sphere $\|v\| = 1$ (so $\sum_i c_i^2 = 1$), this is a weighted average of the eigenvalues, and a weighted average lies between the smallest and largest ingredient:

$$\lambda_{\min}(X) \;=\; \min_{\|v\|=1} v^\mathsf{T} X v, \qquad \lambda_{\max}(X) \;=\; \max_{\|v\|=1} v^\mathsf{T} X v. \tag{26}$$

The ratio $v^{\mathsf{T}}Xv/v^{\mathsf{T}}v$ is the *Rayleigh quotient*, and (26) says $\lambda_{\max}$ and $\lambda_{\min}$ are simply its extreme values—the most and least the matrix can stretch a unit vector. This is the bridge to the modelling tricks of §6: bounding $\lambda_{\max}(X) \le t$ is the same as demanding $v^{\mathsf{T}}Xv \le t$ for *every* unit $v$, i.e. $tI - X \succeq 0$—an eigenvalue bound rewritten as a linear matrix inequality. With the spectral theorem, the trace inner product, and the Rayleigh quotient in hand, every symbol in the body has a home.

# B Deriving the semidefinite dual

This appendix derives the dual (17) from the primal (11) by the standard Lagrangian route, then re-reads weak duality from that vantage.

**The Lagrangian.** Start from the primal: minimise $\langle C,\, X\rangle$ over $X \succeq 0$ subject to $\langle A_i,\, X\rangle = b_i$ for $i = 1, \dots, m$. We keep the cone constraint $X \succeq 0$ as a hard restriction on the domain and attach a real multiplier $y_i$ to each *equality*. The *Lagrangian* is

$$L(X, y) = \langle C,\, X\rangle - \sum_{i=1}^{m} y_i\big(\langle A_i,\, X\rangle - b_i\big) = \langle C,\, X\rangle - \sum_{i=1}^{m} y_i\langle A_i,\, X\rangle + b^{\mathsf{T}}y. \tag{27}$$

The *dual function* is its infimum over the primal domain $X \succeq 0$:

$$g(y) \;=\; \inf_{X \succeq 0}\; L(X, y).$$

Because $L$ is linear in $X$, collect the terms multiplying $X$. Using linearity of the inner product, $\langle C,\, X\rangle - \sum_i y_i\langle A_i,\, X\rangle = \big\langle C - \sum_i y_i A_i,\, X\big\rangle$. Define the *dual slack*

$$S \;:=\; C - \sum_{i=1}^{m} y_i A_i \;\in\; \mathbb{S}^n,$$

so that

$$g(y) \;=\; b^{\mathsf{T}}y \;+\; \inf_{X \succeq 0}\, \langle S,\, X\rangle. \tag{28}$$

**Evaluating the infimum over the cone.** Everything now rests on $\inf_{X\succeq 0}\langle S,\, X\rangle$. Two cases, and they hinge on the self-duality of the PSD cone (§4).

- If $S \succeq 0$, then $\langle S,\, X\rangle = \operatorname{tr}(SX) \ge 0$ for every $X \succeq 0$, and the value 0 is attained at $X = 0$. Hence the infimum is 0.
- If $S \not\succeq 0$, then $S$ has a negative eigenvalue: $Sv = \lambda v$ with $\lambda < 0$ for some unit vector $v$. Take $X = \tau\, vv^{\mathsf{T}} \succeq 0$ for $\tau > 0$; then $\langle S,\, X\rangle = \tau\, v^{\mathsf{T}}Sv = \tau\lambda \to -\infty$ as $\tau \to \infty$. The infimum is $-\infty$.

Combining with (28),

$$g(y) \;=\; \begin{cases} b^{\mathsf{T}}y, & S = C - \sum_i y_i A_i \;\succeq\; 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

A finite lower bound requires $S \succeq 0$, so the best (largest) bound is obtained by maximising $g$ over the region where it is finite:

$$\max_{y \in \mathbb{R}^m}\; b^{\mathsf{T}}y \quad \text{s.t.} \quad C - \sum_{i=1}^{m} y_i A_i \;\succeq\; 0,$$

which, naming the slack $S$ explicitly, is precisely the dual (17).

**Weak duality, re-read.** The Lagrangian makes weak duality almost a tautology. For any primal-feasible $X$ (so $X \succeq 0$, $\langle A_i, X\rangle = b_i$) the equality terms in (27) vanish, giving $L(X, y) = \langle C, X\rangle$. For any dual-feasible $(y, S)$ we have $g(y) = b^{\mathsf{T}} y$ and, by definition of the infimum, $g(y) \le L(X, y)$. Chaining,

$$b^{\mathsf{T}} y \;=\; g(y) \;\le\; L(X, y) \;=\; \langle C,\, X\rangle,$$

so $\langle C, X\rangle - b^{\mathsf{T}} y \ge 0$. Spelling out the difference with $C = \sum_i y_i A_i + S$ recovers the identity (18):

$$\langle C,\, X\rangle - b^{\mathsf{T}} y = \Big\langle \textstyle\sum_i y_i A_i + S,\, X \Big\rangle - b^{\mathsf{T}} y = \sum_i y_i b_i + \langle S,\, X\rangle - b^{\mathsf{T}} y = \langle X,\, S\rangle \ge 0.$$

**Strong duality.** Weak duality is unconditional. Equality of the two optima — zero gap, both attained — follows from *Slater's condition*: if the primal has a strictly feasible point $X \succ 0$ satisfying the equalities (or, symmetrically, the dual has $S \succ 0$), then strong duality holds. This is the matrix instance of the general convex-duality theorem; the strict interior point is exactly what rules out the degenerate, boundary-tight cases warned about in §7 [1, 2].

## C A semidefinite solver in forty lines

To make §8 concrete, here is a tiny but *real* solver — a primal barrier method for the LMI-form SDP (12),

$$\min_{x \in \mathbb{R}^m} \; c^{\mathsf{T}} x \quad \text{s.t.} \quad A(x) := A_0 + \sum_{i=1}^{m} x_i A_i \;\succ 0\; 0.$$

Following Algorithm 1, we minimise $t\, c^{\mathsf{T}} x + \phi(x)$ with $\phi(x) = -\log\det A(x)$ for an increasing sequence of $t$, each inner solve by damped Newton. The two ingredients Newton needs are the gradient and Hessian of the barrier. With $A = A(x)$ and $G = A^{-1}$, differentiating $-\log\det A(x)$ (using $\partial_x \log\det A = \operatorname{tr}(A^{-1}\partial_x A)$ and $\partial_x A = A_i$) gives

$$\partial_i\, \phi(x) = -\operatorname{tr}\big(G\, A_i\big), \tag{29}$$

$$\partial^2_{ij}\, \phi(x) = \operatorname{tr}\big(G\, A_i\, G\, A_j\big). \tag{30}$$

Newton then minimises the gradient $g = t\, c + \nabla\phi$ using the Hessian $H_{ij} = \operatorname{tr}(G A_i G A_j)$: the step is $\Delta x = -H^{-1} g$, backtracked to keep $A(x) \succ 0$. After each inner solve we increase $t$ by a factor $\mu$ and stop when the certified gap $k/t < \epsilon$, where $k$ is the matrix size.

```
import numpy as np


def sdp_barrier(c, A_list, x0, t=1.0, mu=10.0, eps=1e-6):
    c = np.asarray(c, dtype=float)
    A_list = [np.asarray(A, dtype=float) for A in A_list]
    A0, Ai = A_list[0], A_list[1:]   # A0 + sum x_i Ai[i]
    n = A0.shape[0]                  # matrix size k
    m = len(Ai)                      # number of variables

    def amat(x):
        return A0 + sum(xi * Ai[i] for i, xi in enumerate(x))

    def is_pd(M):                    # Cholesky-based PD test
        try:
            np.linalg.cholesky(M)
```

```
            return True
        except np.linalg.LinAlgError:
            return False

    x = np.asarray(x0, dtype=float).copy()
    while n / t > eps:                # outer: gap n/t -> 0
        for _ in range(50):           # inner: damped Newton
            A = amat(x)
            G = np.linalg.inv(A)
            grad = t * c + np.array(
                [-np.trace(G @ Ai[i]) for i in range(m)])
            H = np.array([[np.trace(G @ Ai[i] @ G @ Ai[j])
                           for j in range(m)] for i in range(m)])
            step = -np.linalg.solve(H, grad)
            if np.sqrt(-grad @ step) < 1e-9:
                break                 # Newton decrement small
            s = 1.0                   # backtrack to stay PD
            while not is_pd(amat(x + s * step)):
                s *= 0.5
            f0 = t * c @ x - np.linalg.slogdet(A)[1]
            while (t * c @ (x + s * step)
                   - np.linalg.slogdet(amat(x + s * step))[1]
                   > f0 + 0.25 * s * grad @ step):
                s *= 0.5              # sufficient-decrease test
            x = x + s * step
        t *= mu                       # tighten the barrier
    return x
```

**Reading the code against §8.** The outer `while` is the central-path loop: each pass multiplies $t$ by $\mu$, weakening the barrier and stepping down the path, and it terminates exactly on the certified gap $n/t < \epsilon$. The inner `for` is damped Newton on $t\,c^{\mathsf{T}}x + \phi(x)$, using the closed-form gradient (29) and Hessian (30). The two backtracking loops are the line search of Algorithm 1: the first halves the step until $A(x)$ is positive definite (the `is_pd` Cholesky test — note we test feasibility this way, not by a sign of det, since a negative-definite even-dimensional matrix has *positive* determinant), the second enforces sufficient decrease so the iterate makes real progress.

**What it is, and is not.** This is a *primal-only*, *dense*, teaching solver: it forms full $n \times n$ matrices and an $m \times m$ Hessian explicitly, so it is happy on small problems and will choke on large or sparse ones. Production solvers differ in two ways — they are *primal–dual* (tracking $X$ and $(y, S)$ together, as in §8, for faster and more numerically robust convergence) and they *exploit sparsity* in the $A_i$ to avoid the dense Schur system. Even from this primal iterate a dual certificate is recoverable: at convergence the central point yields $S = C - \sum_i y_i A_i$ with $y$ read off the barrier gradient, giving the $(y, S)$ that certifies the answer by the three checks of §7. The point here is not speed but transparency — every line maps to a sentence in the theory.

## References


[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[2] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[3] M. J. Todd. Semidefinite optimization. *Acta Numerica*, 10:515–560, 2001.

[4] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, 1994.

[5] H. Wolkowicz, R. Saigal, and L. Vandenberghe (eds.). *Handbook of Semidefinite Programming*. Kluwer, 2000.

[6] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[7] N. J. Higham. Computing the nearest correlation matrix—a problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, 2002.

[8] J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[9] P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2):293–320, 2003.

[10] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.