Examples of convex optimization problems

Before we dig deeper into the mathematical and algorithmic details of convex optimization, we will start with a very brief tour of common categories of convex optimization problems, giving a few practical examples where each arises. This discussion is by no means exhaustive, but is merely intended to help you to have some concrete examples in the back of your mind where the techniques we will soon start developing can be applied.

Linear programming

Perhaps the simplest convex optimization problem to write down (although not necessarily the easiest to solve) is a **linear program** (LP). An LP minimizes a linear objective function subject to multiple linear constraints:

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \ \text{ subject to } \ \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} \leq b_m, \ \ m = 1, \ldots, M.$$

The general form above can include linear equality constraints $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} = b_i$ by enforcing both $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} \leq b_i$ and $(-\boldsymbol{a}_i)^{\mathrm{T}}\boldsymbol{x} \leq b_i$ — in our study later on, we will find it convenient to specifically distinguish between these two types of constraints. We can also write the M constraints compactly as $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, where $\boldsymbol{A}$ is the $M \times N$ matrix with the $\boldsymbol{a}_m^{\mathrm{T}}$ as rows.

Linear programs do not necessarily have to have a solution; it is possible that there is no $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, or that the program is unbounded in that there exists a series $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$, all obeying $\boldsymbol{A}\boldsymbol{x}_k \leq \boldsymbol{b}$, with $\lim \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_k \to -\infty$.

There is no formula for the solution of a general linear program. Fortunately, there exists very reliable and efficient software for solving them. The first LP solver was developed in the late 1940s (Dantzig's "simplex algorithm"), and now LP solvers are considered a mature technology. If the constraint matrix $\boldsymbol{A}$ is structured, then linear programs with millions of variables can be solved to high accuracy on a standard computer.

Linear programs are a very important class of optimization problems. However, if a single constraint (or the objective function) are nonlinear, then we move into the much broader class of **nonlinear programs**. While much of what we will discuss in this course is relevant to LPs, we will spend a greater fraction of the course discussing these more general nonlinear optimization problems.

Example: Chebyshev approximations

Suppose that we want to find the vector $\boldsymbol{x}$ so that $\boldsymbol{A}\boldsymbol{x}$ does not vary too much in its maximum deviation:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \quad \max_{m=1,\ldots,M} |y_m - \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x}| \;=\; \underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_\infty.$$

This is called the **Chebyshev approximation problem**.

We can solve this problem with linear programming. To do this, we introduce the auxiliary variable $u \in \mathbb{R}$ — it should be easy to see that the program above is equivalent to

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^N,\; u\in\mathbb{R}}{\text{minimize}} \;\; u \quad \text{subject to} \;\; & y_m - \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} \le u \\ & y_m - \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} \ge -u \\ & m = 1, \ldots, M. \end{aligned}$$

To put this in the standard linear programming form, take

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ u \end{bmatrix}, \quad \boldsymbol{c}' = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix}, \quad \boldsymbol{A}' = \begin{bmatrix} -\boldsymbol{A} & -1 \\ \boldsymbol{A} & -1 \end{bmatrix}, \quad \boldsymbol{b}' = \begin{bmatrix} -\boldsymbol{y} \\ \boldsymbol{y} \end{bmatrix},$$

and then solve

$$\underset{\boldsymbol{z} \in \mathbb{R}^{N+1}}{\text{minimize}} \ \boldsymbol{c}'^{\mathrm{T}} \boldsymbol{z} \ \text{ subject to } \ \boldsymbol{A}' \boldsymbol{z} \leq \boldsymbol{b}'.$$

One natural application of this arises in the context of **filter design**. The standard "filter synthesis" problem is to find an finite-impulse response (FIR) filter whose discrete-time Fourier transform (DTFT) is as close to some target $H^\star(\omega)$ as possible.

We can write this as an optimization problem as follows:

$$\underset{H}{\text{minimize}} \ \sup_{\omega \in [-\pi, \pi]} \left| H^\star(\omega) - H(\omega) \right|, \ \text{ subject to } H(\omega) \text{ being FIR}$$

When the deviation from the optimal response is measured using a uniform error, this is called "equiripple design", since the error in the solution will tend to have ripples a uniform distance away from the ideal.

If we restrict ourselves to the case where $H^\star(\omega)$ has linear phase (so the impulse response is symmetric around some time index)[1] we can recast this as a Chebyshev approximation problem.

Specifically, a symmetric filter with $2K+1$ taps (meaning that $h_n = 0$ for $|n| > K$) has a real DTFT that can be written as a superposition of a DC term plus K cosines:

$$H(\omega) = \sum_{k=0}^{K} \tilde{h}_k \cos(k\omega), \quad \tilde{h}_k = \begin{cases} h_0, & k = 0 \\ 2h_k, & 1 \leq k \leq K. \end{cases}$$

[1] The case with general phase can also be handled using convex optimization, but it is not naturally stated as a linear program.

So we are trying to solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^{K+1}}{\text{minimize}} \quad \sup_{\omega\in[-\pi,\pi]} \left| H^\star(\omega) - \sum_{k=0}^{K} x_k \cos(k\omega) \right|.$$

It is actually possible to solve this problem as stated – our very own Jim McClellan worked this out in the early 1970s with his advisor Tom Parks, developing the now ubiquitous Parks-McClellan filter design algorithm. The solution is not obvious, however, mostly due to the presence of supremum over ω.

Here, suppose we instead approximate the supremum on the inside by measuring it at M equally spaced points $\omega_1, \ldots, \omega_M$ between $-\pi$ and π. Then

$$\underset{\boldsymbol{x}}{\text{minimize}} \max_{\omega_m} \left| H^\star(\omega_m) - \sum_{k=0}^{K} x_k \cos(k\omega_m) \right| = \underset{\boldsymbol{x}}{\text{minimize}} \, \|\boldsymbol{y} - \boldsymbol{F}\boldsymbol{x}\|_\infty,$$

where $\boldsymbol{y} \in \mathbb{R}^M$ and the $M \times (K+1)$ matrix $\boldsymbol{F}$ are defined as

$$\boldsymbol{y} = \begin{bmatrix} H^\star(\omega_1) \\ H^\star(\omega_2) \\ \vdots \\ H^\star(\omega_M), \end{bmatrix} \quad \boldsymbol{F} = \begin{bmatrix} 1 & \cos(\omega_1) & \cos(2\omega_1) & \cdots & \cos(K\omega_1) \\ 1 & \cos(\omega_2) & \cos(2\omega_2) & \cdots & \cos(K\omega_2) \\ \vdots & & \ddots & & \\ 1 & \cos(\omega_M) & \cos(2\omega_M) & \cdots & \cos(K\omega_M) \end{bmatrix}$$

It should be noted that since the ω_m are equally spaced, the matrix $\boldsymbol{F}$ (and its adjoint) can be applied efficiently using a fast discrete cosine transform. This has a direct impact on the number of computations we need to solve the Chebyshev approximation problem above.

Least squares

A prototypical example of a *nonlinear* convex optimization problem is **least squares**. Specifically, given a $M \times N$ matrix $\boldsymbol{A}$ and a vector $\boldsymbol{y} \in \mathbb{R}^M$, the unconstrained least squares problem is given by

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2. \tag{1}$$

When $\boldsymbol{A}$ has full column rank (and so $M \geq N$), then there is a unique closed-form solution:

$$\widehat{\boldsymbol{x}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}.$$

We can also write this in terms of the SVD of $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$:

$$\widehat{\boldsymbol{x}} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}.$$

The mapping from the data vector $\boldsymbol{y}$ to the solution $\widehat{\boldsymbol{x}}$ is linear, and the corresponding $N \times M$ matrix $\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^{\mathrm{T}}$ is called the **pseudo-inverse**.

When $\boldsymbol{A}$ does not have full column rank, then the solution is non-unique. An interesting case is when $\boldsymbol{A}$ is underdetermined ($M < N$) with $\mathrm{rank}(\boldsymbol{A}) = M$ (full row rank). Then there are many $\boldsymbol{x}$ such that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ and so $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 = 0$. Of these, we might choose the one which has the smallest norm:

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{x}\|_2 \ \text{ subject to } \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}.$$

It turns out that the solution is again given by the pseudo-inverse. We can still write $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$, where $\boldsymbol{\Sigma}$ is $M \times M$, diagonal, and invertible, $\boldsymbol{U}$ is $M \times M$ and $\boldsymbol{V}$ is $N \times M$. Then $\widehat{\boldsymbol{x}} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^{\mathrm{T}}$ find the shortest vector (in the Euclidean sense) that obeys the M specified linear constraints.

Example: Regression

A fundamental problem in statistics is to estimate a function given point samples (that are possibly heavily corrupted). We observe pairs of points[2] (x_m, y_m) for $m = 1, \ldots, M$, and want to find a function $f(x)$ such that

$$f(x_m) \approx y_m, \quad m = 1, \ldots, M.$$

Of course, the problem is not well-posed yet, since there are any number of functions for which $f(x_m) = y_m$ exactly. We regularize the problem in two ways. The first is by specifying a class that $f(\cdot)$ belongs to. One way of doing this is by building f up out of a linear combination of basis functions $\phi_n(\cdot)$:

$$f(x) = \sum_{n=1}^{N} \alpha_n \phi_n(x).$$

We now fit a function by solving for the expansion coefficients $\boldsymbol{\alpha}$. There is a classical complexity versus robustness trade-off in choosing the number of basis functions N.

The quality of a proposed fit is measured by a loss function — this loss is typically (but not necessarily) specified pointwise at each of the samples, and then averaged over all the sample points:

$$\text{Loss}(\boldsymbol{\alpha}; \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{M} \sum_{m=1}^{M} \ell(\boldsymbol{\alpha}; x_m, y_m).$$

[2] We are just considering functions of a single variable here, but it is easy to see how the basic setup extends to functions of a vector.

One choice for $\ell(\cdot)$ is the *squared-loss*:

$$\ell(\boldsymbol{\alpha}; x_m, y_m) = \left(y_m - \sum_{n=1}^{N} \alpha_n \phi_n(x_m) \right)^2,$$

which is just the square between the difference of the observed value y_m and its prediction using the candidate $\boldsymbol{\alpha}$.

We can express everything more simply by putting it in matrix form. We create the $M \times N$ matrix $\boldsymbol{\Phi}$:

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_N(x_2) \\ \vdots & & \ddots & \\ \phi_1(x_M) & \phi_2(x_M) & \cdots & \phi_N(x_M) \end{bmatrix}$$

$\boldsymbol{\Phi}$ maps a set of expansion coefficients $\boldsymbol{\alpha} \in \mathbb{R}^N$ to a set of M predictions for the vector of observations $\boldsymbol{y} \in \mathbb{R}^M$. Finding the $\boldsymbol{\alpha}$ that minimizes the squared-loss is now reduced to the standard least squares problem:

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\alpha}\|_2^2$$

It is also possible to smooth the results and stay in the least squares framework. If $\boldsymbol{\Phi}$ is ill-conditioned, then the least squares solution might do dramatic things to $\boldsymbol{\alpha}$ to make it match $\boldsymbol{y}$ as closely as possible. To discourage this, we can penalize $\|\boldsymbol{\alpha}\|_2$:

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\alpha}\|_2^2 + \tau \|\boldsymbol{\alpha}\|_2^2,$$

where $\tau > 0$ is a parameter we can adjust. This can be converted back to standard least squares problem by concatenating ($\sqrt{\tau}$ times)

the identity to the bottom of $\boldsymbol{\Phi}$ and zeros to the bottom of $\boldsymbol{y}$. At any rate, the formula for the solution to this program is

$$\widehat{\boldsymbol{x}} = (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \tau\mathbf{I})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}.$$

This is called *ridge regression* in the statistics community (and *Tikhonov regularization* in the linear inverse problems community).

Another strategy in such cases is to choose a slightly different regularizer and penalize $\|\boldsymbol{\alpha}\|_1$:

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\alpha}\|_2^2 + \tau\|\boldsymbol{\alpha}\|_1.$$

This is most commonly known as the *LASSO* (for "least absolute shrinkage and selection operator"). This small change can have a dramatic impact in the properties of the resulting solution. In particular, it is an effective strategy for promoting *sparsity* in the solution $\widehat{\boldsymbol{\alpha}}$. This is useful in a variety of circumstances, and is something we will return to later in this course.

Note, however, that unlike ridge regression/Tikhonov regularization, the LASSO no longer has a closed form solution. Moreover, the term involving $\|\boldsymbol{\alpha}\|_1$ is not differentiable. Optimization problems like this are an important class of problems, and one that we will devote significant attention to later in this course.

Quadratic programming

Let us briefly return to our standard least squares problem in (1). It is easy to show that this is equivalent to the problem of

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}.$$

Suppose you now wanted to enforce some additional structure on $\boldsymbol{\alpha}$. For example, you might have reason to desire a solution with only non-negative values. In adding such a constraint, we arrive at an example of a **quadratic program** (QP).

A QP minimizes a quadratic functional subject to linear constraints:

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}, \ \text{ subject to } \ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}.$$

If $\boldsymbol{H}$ is symmetric positive semidefinite (i.e., symmetric with nonnegative eigenvalues), then the program is convex. If $\boldsymbol{H}$ has even a single negative eigenvalue, then solving the program above is NP-hard.

QPs are almost as ubiquitous as LPs; they have been used in finance since the 1950s (see the example below), and are found all over operations research, control systems, and machine learning. As with LPs, there are reliable solvers and can be considered a mature technology.

A **quadratically constrained quadratic program** (QCQP) allows (convex) quadratic inequality constraints:

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}, \ \text{ subject to } \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_m\boldsymbol{x} + \boldsymbol{c}_m^{\mathrm{T}}\boldsymbol{x} \leq b_m, \quad m = 1, \ldots, M.$$

This program is convex if all of the $\boldsymbol{H}_m$ are symmetric positive semidefinite; we are minimizing a convex quadratic functional over a region defined by an intersection of ellipsoids.

Example: Portfolio optimization

One of the classic examples in convex optimization is finding investment strategies that "optimally"[3] balance the risk versus the return. The following quadratic program formulation is due to Markowitz, who formulated it in the 1950s, then won a Nobel Prize for it in 1990.

We want to spread our money over N different assets; the fraction of our money we invest in asset n is denoted x_n. We have the immediate constraints that

$$\sum_{n=1}^{N} x_n = 1, \quad \text{and} \quad 0 \leq x_n \leq 1, \quad \text{for } n = 1, \ldots, N.$$

The expected return on these investments, which are usually calculated using some kind of historical average, is $\mu_1, \ldots, \mu_N$. The μ_n are specified as multipliers, so $\mu_n = 1.16$ means that asset n has a historical return of 16%. We specify some target expected return ρ, which means

$$\sum_{n=1}^{N} \mu_n x_n \geq \rho.$$

We want to solve for the $\boldsymbol{x}$ that achieves this level of return while minimizing our *risk*. Here, the definition of risk is simply the variance of our return — if the assets have covariance matrix $\boldsymbol{R}$, then the risk of a given portfolio allocation $\boldsymbol{x}$ is

$$\text{Risk}(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{R} \boldsymbol{x} = \sum_{m=1}^{M} \sum_{n=1}^{M} R_{mn} x_m x_n.$$

[3] I put "optimally" in quotes because, like everything in finance and the world, this technique finds the optimal answer for a specified model. The big question is then how good your model is ...

Our optimization program is then[4]

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \boldsymbol{x}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x} \geq \rho \\ & \mathbf{1}^{\mathrm{T}}\boldsymbol{x} = 1 \\ & \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}. \end{aligned}$$

This is an example of a QP with linear constraints. It is convex since the matrix $\boldsymbol{R}$ is a covariance matrix, and so by construction it is symmetric positive semidefinite.

Example: Support vector machines

Support vector machines (SVMs) are a classical approach for designing a classifier in machine learning, and involves solving an optimization problem that we will revisit again in more detail later on in the course. An SVM takes as input a dataset $\{\boldsymbol{x}_i, y_i\}$ with $\boldsymbol{x}_i \in \mathbb{R}^N$ and $y_i \in \{-1, +1\}$.

The goal of the SVM is to find a vector $\boldsymbol{w} \in \mathbb{R}^N$ and a scalar $b \in \mathbb{R}$ that define a separating hyperplane, i.e., a hyperplane that separates the sets $\{\boldsymbol{x}_i : y_i = +1\}$ and $\{\boldsymbol{x}_i : y_i = -1\}$. This can be posed as constraints on $\boldsymbol{w}$ and b of the form

$$(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b)y_i \geq 1.$$

Among all separating hyperplanes, it turns out that the one that minimizes $\|\boldsymbol{w}\|_2^2$ corresponds to the hyperplane that maximizes the

[4] Throughout these notes, we will use $\mathbf{1}$ for a vector of all ones, and $\mathbf{0}$ for a vector of all zeros.

margin between the two classes, where the margin corresponds to the distance from the hyperplane to the nearest $\boldsymbol{x}_i$.

Thus, the problem of finding the best (maximum margin) separating hyperplane reduces to

$$\underset{\boldsymbol{w},b}{\text{minimize}} \ \|\boldsymbol{w}\|_2^2 \ \text{ subject to } \ (\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b)y_i \geq 1 \text{ for all } i.$$

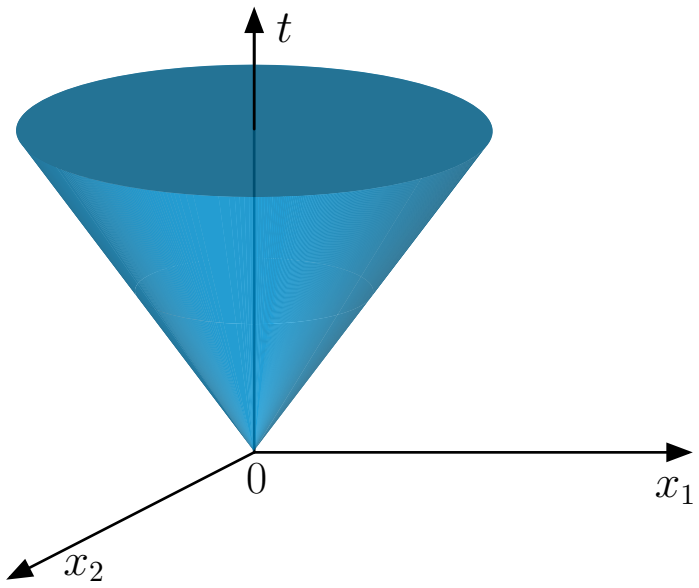
This is another example of a QP with linear constraints.

Second-order cone programs

A **second-order cone program** (SOCP) is an optimization problem where the constraint set forms what is called, perhaps unsurprisingly, a second-order cone. The canonical example of a second-order cone is the set:

$$\left\{(\boldsymbol{x}, t), \boldsymbol{x} \in \mathbb{R}^N, t \in \mathbb{R} : \|\boldsymbol{x}\|_2 \leq t\right\}.$$

This is a subset of $\mathbb{R}^{N+1}$. Here is an example in $\mathbb{R}^3$:



The standard form of a SOCP is

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} && \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ &\text{subject to} && \|\boldsymbol{A}_m\boldsymbol{x}+\boldsymbol{b}_m\|_2 \leq \boldsymbol{c}_m^{\mathrm{T}}\boldsymbol{x}+d_m, \quad m=1,\ldots,M. \end{aligned}$$

We have a linear objective function and constraints that require $(\boldsymbol{y}, t)$ to lie inside the second-order cone, where $\boldsymbol{y}$ and t are allowed to be any affine function of $\boldsymbol{x}$.

SOCPs turn out to be much more common than you might initially expect. First, it is not hard to show that an LP is also a SOCP. It turns out that QPs and (convex) QCQPs are also SOCPs, so we can think of SOCPs as a generalization of what we have already seen. However, the class of possible SOCPs also includes many optimization problems beyond what we have seen so far.

Example: Generalized geometric medians

Suppose that we have M points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_M \in \mathbb{R}^N$ and that we would like to find the "center" of this set of points. The *geometric median* is the point $\boldsymbol{x}$ that minimizes the sum (or equivalently, average) of the distances to the points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_M$. This can be posed as the optimization problem

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \sum_{m=1}^{M} \|\boldsymbol{x}-\boldsymbol{p}_m\|_2.$$

In the case where $N=1$, this is equivalent to the standard median. The special case of $M=3$ points in a dimension $N \geq 2$ was first considered by Pierre de Fermat, with Evangelista Torricelli providing a simple geometric solution in the 17^{th} century. In general, however, there is no closed-form solution to this problem.

It is relatively straightforward to show that this problem can be cast as a SOCP. Specifically, it should be clear that it is equivalent to:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{t}}{\text{minimize}} \quad & \sum_{m=1}^{M} t_m \\ \text{subject to} \quad & \|\boldsymbol{x} - \boldsymbol{p}_m\|_2 \leq t_m, \quad m = 1, \ldots, M. \end{aligned}$$

A slight variation on this problem is to try to minimize the maximum distance from $\boldsymbol{x}$ to the $\boldsymbol{p}_m$:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \max_{m \in \{1,\ldots,M\}} \|\boldsymbol{x} - \boldsymbol{p}_m\|_2.$$

This too has a simple formulation as a SOCP:

$$\begin{aligned} \underset{\boldsymbol{x},t}{\text{minimize}} \quad & t \\ \text{subject to} \quad & \|\boldsymbol{x} - \boldsymbol{p}_m\|_2 \leq t, \quad m = 1, \ldots, M. \end{aligned}$$

Semidefinite programs

So far we have typically been looking at problems where we are optimizing over vectors $\boldsymbol{x} \in \mathbb{R}^N$. In many important applications, our decision variables are more naturally represented as a matrix $\boldsymbol{X}$. In such problems, it is common to encounter the constraint that this matrix $\boldsymbol{X}$ must be positive semidefinite. When the objective function is linear and we have affine constraints, this is called a **semidefinite program** (SDP).

To state the standard form for an SDP, it is useful to introduce some notation. First, we will let $\mathbb{S}^N$ denote the set of $N \times N$ symmetric

matrices, and $\mathbb{S}_+^N$ the set of symmetric positive semidefinite matrices. Furthermore, we let

$$\langle \boldsymbol{Y}, \boldsymbol{X} \rangle = \text{trace}(\boldsymbol{Y}^{\text{T}} \boldsymbol{X})$$

denote the (trace) inner product between a pair of matrices.[5] With this notation in hand, the standard form for an SDP is given by

$$\begin{aligned} \underset{\boldsymbol{X}}{\text{minimize}} \quad & \langle \boldsymbol{C}, \boldsymbol{X} \rangle \\ \text{subject to} \quad & \langle \boldsymbol{A}_m, \boldsymbol{X} \rangle \leq b_m, \quad m = 1, \ldots, M \\ & \boldsymbol{X} \in \mathbb{S}_+^N, \end{aligned}$$

where $\boldsymbol{C}, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_M \in \mathbb{S}$.

SDPs are the broadest class of convex problems that we will study in this course. All of the problems we have looked at so far (LPs, QPs, SOCPs) can be shown to be special cases of SDPs. We will see a number of examples of SDPs that arise in applications throughout the course.

Example: Bounding portfolio risk

Let us briefly return to our previous example or portfolio optimization. Before we assumed that we knew the expected returns and the covariance matrix $\boldsymbol{R}$ for the different assets under consideration, and our goal was to determine the optimal allocation. Here we consider a slightly different problem. Suppose that we already have a fixed allocation $\boldsymbol{x}$ across the different assets, but rather than knowing the

[5] This is simply the inner product that would result from reshaping $\boldsymbol{X}$ and $\boldsymbol{Y}$ into vectors and applying the standard inner product.

covariance matrix $\boldsymbol{R}$ exactly, we assume that we have only an estimate of $\boldsymbol{R}$. A natural question is whether we can quantify how large the true risk of our portfolio might be in such a case.

Suppose that we have confidence intervals on how accurate our covariance estimate is of the form

$$L_{mn} \leq R_{mn} \leq U_{mn}.$$

For a given portfolio $\boldsymbol{x}$, we can compute the maximum possible risk of that portfolio that is consistent with the given bounds via the following SDP:

$$\begin{aligned} \underset{\boldsymbol{R}}{\text{maximize}} \quad & \boldsymbol{x}^{\mathrm{T}} \boldsymbol{R} \boldsymbol{x} \\ \text{subject to} \quad & L_{mn} \leq R_{mn} \leq U_{mn}, \quad m, n = 1, \ldots, N \\ & \boldsymbol{R} \in \mathbb{S}_{+}^{N}. \end{aligned}$$

We have to enforce the constraint that $\boldsymbol{R} \in \mathbb{S}_{+}^{N}$ because $\boldsymbol{R}$ must be a covariance matrix, and ignoring this constraint would yield a risk that is not actually achievable.