I. Introduction to Convex Optimization

Introduction to Optimization

In its most general form, an optimization program

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \;\; & f_0(\boldsymbol{x}) \\ \text{subject to} \;\; & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

searches for the vector $\boldsymbol{x} \in \mathbb{R}^N$ that minimizes a given functional $f_0 : \mathbb{R}^N \to \mathbb{R}$ over a set $\mathcal{X} \subset \mathbb{R}^N$.

We will rely on $\mathcal{X}$ to be specified by a series of constraint functionals:

$$\boldsymbol{x} \in \mathcal{X} \quad \Leftrightarrow \quad f_m(\boldsymbol{x}) \leq b_m \;\; \text{for } m = 1, \ldots, M.$$

Solving optimization problems is in general very difficult. In this class, we will develop a framework for analyzing and solving **convex** programs, which simply means each of the functionals above obeys

$$f_m(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \;\; \leq \;\; \theta f_m(\boldsymbol{x}) + (1-\theta) f_m(\boldsymbol{y}),$$

for all $m = 0, 1, \ldots, M$, $0 \leq \theta \leq 1$, and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$. (Much more on this later.)

What does convexity tell us? Two important things:

- Local minimizers are also global minimizers. So we can check if a certain point is optimal by looking in a small neighborhood and seeing if there is a direction to move that decreases f_0.
- First-order necessary conditions for optimality turn out to be sufficient. When the problem is unconstrained and smooth, this means we can find an optimal point by finding $\hat{\boldsymbol{x}}$ such that $\nabla f_0(\hat{\boldsymbol{x}}) = \boldsymbol{0}$.

The upshot of these two things is that if the $f_m(\boldsymbol{x})$ and their derivatives[1] are reasonable to compute, then relatively simple algorithms (e.g. gradient descent) are provably effective at performing the optimization.

The great watershed in optimization is not between linearity and non-linearity, but convexity and non-convexity.

— R. Tyrrell Rockafellar

The material in this course has three major components. The first is the mathematical foundations of convex optimization. We will see that talking about the solution to convex problems requires a beautiful combination of algebraic and geometric ideas.

The second component is algorithms for solving convex programs. We will talk about general purpose algorithms (and their associated computational guarantees), but we will also look at algorithms that are specialized to certain classes of problems, and even certain applications. Rather than focus on the "latest and greatest", we will try to understand the key ideas that are combined in different ways into many solvers.

Finally, we will talk a lot about *modeling*. That is, how convex optimization appears in signal processing, machine learning, statistical inference, etc. We will give many examples of mapping word problem into an optimization program. These examples will be interleaved with the discussion of the first two components, and there are several examples which we may return to several times.

[1] And as we will see, all of what we do is very naturally extended to non-smooth functions which do not have any derivatives.

You might have two questions at this point:

- Can all convex programs be solved efficiently?
  Unfortunately, no. There are many examples of even seemingly innocuous convex programs which are NP-hard. One way this can happen is if the functionals themselves are hard to compute. For example, suppose we were trying to find the matrix with minimal $(\infty, 1)$ norm that obeyed some convex constraints:

  $$f_0(\boldsymbol{X}) = \|\boldsymbol{X}\|_{\infty,1} = \max_{\|\boldsymbol{v}\|_\infty \leq 1} \|\boldsymbol{X}\boldsymbol{v}\|_1.$$

  This is a valid matrix norm, and we will see later that all valid norms are convex. But it is known that computing f_0 is NP-hard (see [Roh00]), as is approximating it to a fixed accuracy. So optimizations involving this quantity are bound to be difficult.

- Are there non-convex programs which can be solved efficiently?
  Of course there are. Here is one for which you already know the answer:

  $$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{maximize}} \ \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x} \ \text{ subject to } \ \|\boldsymbol{x}\|_2 = 1,$$

  where $\boldsymbol{A}$ is an arbitrary $N \times N$ symmetric matrix. This is the *maximization* of an indefinite quadratic form (not necessarily convex or concave) over a nonconvex set. But we know that the optimal value of this program is the largest eigenvalue, and the optimizer is the corresponding eigenvector, and there are well-known practical algorithms for computing these.

When there is a solutions to a nonconvex problem, it often times relies on nice coincidences in the structure of the problem — perturbing the problem just a little bit can disturb these coincidences. Consider

another nonconvex problem that we know how to solve:

$$\underset{\boldsymbol{X}}{\text{minimize}} \ \sum_{i,j=1}^{N} (X_{i,j} - A_{i,j})^2 \ \text{ subject to } \ \text{rank}(\boldsymbol{X}) \leq R.$$

That is, we are looking for the best rank-R approximation (in the least-squares) sense to the given $n \times n$ matrix $\boldsymbol{A}$. The functional we are optimizing above is convex, but the rank constraint definitely is not. Nevertheless, we can compute the answer efficiently using the SVD of $\boldsymbol{A}$:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}} = \sum_{n=1}^{N} \sigma_n \boldsymbol{u}_n \boldsymbol{v}_n^{\mathrm{T}}, \quad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

The program above is solved simply by truncating this sum to its first R terms:

$$\hat{\boldsymbol{X}} = \sum_{n=1}^{R} \sigma_n \boldsymbol{u}_n \boldsymbol{v}_n^{\mathrm{T}}.$$

But now suppose that instead of the matrix $\boldsymbol{A}$, we are given a subset of its entries indexed by $\mathcal{I}$. We now want to find the matrix that is most consistent over this subset while also having rank at most R:

$$\underset{\boldsymbol{X}}{\text{minimize}} \ \sum_{(i,j) \in \mathcal{I}} (X_{i,j} - A_{i,j})^2 \ \text{ subject to } \ \text{rank}(\boldsymbol{X}) \leq R.$$

Despite its similarity to the first problem above, this "matrix completion" problem is NP-hard.

Convex programs tend to be more robust to variations of this type. Things like adding subspace constraints, restricting variables to be positive, and considering functionals of linear transforms of $\boldsymbol{x}$ all preserve the essential convex structure.

For the rest of this introduction, we will introduce a few of the very well-known classes of convex program, and give an example of an application for each.

Least squares

Given a $M \times N$ matrix $\boldsymbol{A}$ and a vector $\boldsymbol{y} \in \mathbb{R}^M$, we solve the unconstrained problem

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

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

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

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

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

The mapping from the data vector $\boldsymbol{y}$ to the solution $\hat{\boldsymbol{x}}$ 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:

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

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

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

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

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

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 (τ 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

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

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

Linear programming

A **linear program** (LP) minimizes a linear functional 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 $\lim \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_k \to -\infty$.

Unlike least squares, there is no formula for the solution of a 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.

Example: Chebyshev approximations

Consider the following tweak to the least-squares problem we looked at previously. 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 cannot solve this problem using the pseudo-inverse, but we can solve it 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

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

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 $H_*(\omega)$ as possible. When the deviation from the optimal response is measured using a uniform error, this is call "equiripple design", since the error in the solution will tend to have ripples a uniform distance away from the ideal. That is, we would like to solve

$$\underset{H}{\text{minimize}} \sup_{\omega \in [-\pi,\pi]} |H_*(\omega) - H(\omega)|, \quad \text{subject to } H(\omega) \text{ being FIR}$$

If we restrict ourselves to the case where $H_*(\omega)$ has linear phase (so the impulse response is symmetric around some time index)[3] we can recast this as a Chebyshev approximation problem.

A symmetric filter with $2K+1$ taps has a real DTFT that can be written as a superposition of a DC term plus K cosines:

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

So we are trying to solve

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

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

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

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

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

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

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. We will see later how this has a direct impact on the number of computations we need to solve the Chebyshev approximation problem above.

Quadratic programming

A **quadratic program** (QP) minimizes a quadratic functional subject to linear constraints:

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

When $\boldsymbol{P}$ is symmetric positive-definite, then the program is convex. If $\boldsymbol{P}$ 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; it might also be considered a mature technology.

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

$$\begin{aligned}\underset{\boldsymbol{x}}{\text{minimize}} \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^{\mathrm{T}}\boldsymbol{x}, \ \text{ subject to } \ & \boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}_m\boldsymbol{x} + \boldsymbol{q}_m^{\mathrm{T}}\boldsymbol{x} \leq b_m, \\ & m = 1, \ldots, M.\end{aligned}$$

This program is convex if all of the $\boldsymbol{P}_m$ are symmetric positive definite; 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"[4] 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

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

[4] 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[5]

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

This is an example of a *quadratic program* with linear constraints. It is convex since the matrix $\boldsymbol{R}$ is covariance matrix, and so by construction is it symmetric positive semi-definite (i.e. symmetric with all its eigenvalues ≥ 0).

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

Our last example (for now) does not fit into any of the categories mentioned above. It is, however, convex, and shows how viewing a well-known problem through the lens of convex programming can allow us to systematically exploit a priori structural information about our problem.

Example: Structured covariance estimation

Let's recall the basics of estimating the covariance matrix for a Gaussian random vector

$$\boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix}.$$

We assume that $\mathrm{E}[\boldsymbol{X}] = \boldsymbol{0}$. We observe independent realizations $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_K$. How do we estimate the covariance matrix $\boldsymbol{\Sigma} = \mathrm{E}[\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}]$?

If you have taken any class in statistics, you know that the standard estimate is given by the sample covariance:

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{X}_k \boldsymbol{X}_k^{\mathrm{T}}.$$

This makes intuitive sense, but let's justify it a little more carefully by showing it is the maximum likelihood estimate (MLE). Given the $\{\boldsymbol{X}_k\}$, we want to find the matrix $\boldsymbol{R}$ that maximizes the likelihood function

$$\underset{\boldsymbol{R} \in S_{++}^N}{\text{maximize}} \quad \prod_{k=1}^{K} (2\pi)^{-N/2} (\det \boldsymbol{R})^{-1/2} \exp(-\boldsymbol{y}_k^{\mathrm{T}} \boldsymbol{R}^{-1} \boldsymbol{y}_k / 2).$$

The set S^N_{++} is the set of valid covariance matrices (i.e. the set of positive definite matrices).

Since the log function is monotonic, we can equivalently maximize the log-likelihood

$$\underset{\boldsymbol{R}\in S^N_{++}}{\text{maximize}} \ \frac{-KN}{2}\log 2\pi + \frac{K}{2}\log\det \boldsymbol{R}^{-1} - \frac{1}{2}\sum_{k=1}^{K} \boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{y}_k. \quad (1)$$

The first term does not depend on $\boldsymbol{R}$, so we can ignore it. Since the inverse of every valid covariance matrix is also a valid covariance matrix, we can optimize over $\boldsymbol{S} = \boldsymbol{R}^{-1}$. This makes the optimization program

$$\underset{\boldsymbol{S}\in S^N_{++}}{\text{maximize}} \ \frac{K}{2}\log\det \boldsymbol{S} - \frac{1}{2}\sum_{k=1}^{K} \boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{S}\boldsymbol{y}_k.$$

Using the (easily checked) fact that $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{y} = \mathrm{trace}(\boldsymbol{S}\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}})$, we can write the above as

$$\underset{\boldsymbol{S}\in S^N_{++}}{\text{maximize}} \ \log\det \boldsymbol{S} - \mathrm{trace}(\boldsymbol{S}\hat{\boldsymbol{\Sigma}}),$$

where $\hat{\boldsymbol{\Sigma}}$ is the sample covariance matrix (and we have also dropped the constant $K/2$ factors).

The functional $\log\det \boldsymbol{S}$ is concave in its matrix argument $\boldsymbol{S}$ (check this), and with $\hat{\boldsymbol{\Sigma}}$ fixed, $\mathrm{trace}(\boldsymbol{S}\hat{\boldsymbol{\Sigma}})$ is linear in $\boldsymbol{S}$ (both convex and concave). Thus maximizing this functional is the same as minimizing the convex functional $-\log\det \boldsymbol{S} + \mathrm{trace}(\boldsymbol{S}\hat{\boldsymbol{\Sigma}})$. Moreover, the set S^N_{++} is convex, and so it is fair to call the optimization program above a convex program.

Since the functional we are maximizing is smooth, we know that we have a solution $\hat{\boldsymbol{S}}$ if $\hat{\boldsymbol{S}}$ is feasible (in S^N_{++}) and the gradient is equal to zero. It is easy to see that

$$\nabla \operatorname{trace}(\boldsymbol{S}\hat{\boldsymbol{\Sigma}}) = \hat{\boldsymbol{\Sigma}},$$

and it can be shown (do this at home) that

$$\nabla \log\det \boldsymbol{S} = \boldsymbol{S}^{-1}.$$

Thus the optimal solution obeys

$$\hat{\boldsymbol{S}}^{-1} - \hat{\boldsymbol{\Sigma}} = \mathbf{0} \quad \Rightarrow \quad \hat{\boldsymbol{S}} = \hat{\boldsymbol{\Sigma}}^{-1}.$$

This means that the optimal solution to the original program (1) is

$$\hat{\boldsymbol{R}} = \hat{\boldsymbol{\Sigma}}.$$

So the sample covariance really is the maximum likelihood estimate.

You don't need to take a course in convex optimization to tell you that forming the sample covariance matrix is a good idea. But this formulation comes in handy if we want to specify additional constraints on the (inverse) convariance. For example, we might know that the variance of all of the variables is less than some positive number $\nu_{\max}$. This means that $R_{n,n} = S^{-1}_{n,n} \leq \nu_{\max}$ for all $n = 1, \ldots, N$. It turns out (and you should again think about this at home) that functionals which take a matrix in S^N_{++} and return an entry along the diagonal,

$$f_n(\boldsymbol{S}) = S^{-1}_{n,n},$$

are convex in $\boldsymbol{S}$. So these variance constraints can be incorporated while still keeping the program tractable.

Another example is if we know different pairs of variables $(i,j) \in \mathcal{I}$ are *conditionally independent* of one another given the other variables. Two entries of a Gaussian random vector are conditionally independent if the corresponding entry in the *inverse* covariance matrix is zero; so this information can be captured by introducing the constraints

$$S_{i,j} = 0, \quad \text{for all} \quad (i,j) \in \mathcal{I}.$$

Constraints like these come in useful when we are trying to estimate the structure of Gaussian graphical models. (We will say more about this as we revisit this example at different points in the course.)

References

[Roh00] J. Rohn. Computing the norm $\|A\|_{\infty,1}$ is NP-Hard. *Linear and Multilinear Algebra*, 47:195–204, 2000.