0. Introduction and Examples

Introduction to Optimization

Optimization problems are ubiquitous in science and engineering.

Optimization problems arise any time we have a collection of elements and wish to select the "best" one (according to some criterion). The process of casting a real world problem as being one of mathematical optimization consists of three main components

1. a set of variables, often called **decision variables**, that we have control over;
2. an **objective function** that maps the decision variables to some quality that we want to maximize (goodness of fit, profit, etc.) or some cost that we want to minimize (error, loss, etc.); and
3. a **constraint set** that dictates restrictions on the decision variables imposed by physical limitations, budgets on resources, design requirements, etc.

In its most general form, we can express such an optimization problem mathematically as

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; f(\boldsymbol{x}) \;\; \text{subject to} \;\; \boldsymbol{x} \in \mathcal{X}, \tag{1}$$

where $f : \mathcal{X} \to \mathbb{R}$ is our objective function and $\mathcal{X}$ is our constraint set.

In order to solve this optimization problem, we must find an $\widehat{\boldsymbol{x}} \in \mathcal{X}$ such that

$$f(\widehat{\boldsymbol{x}}) \leq f(\boldsymbol{x}) \text{ for all } \boldsymbol{x} \in \mathcal{X}. \tag{2}$$

We call an $\widehat{\boldsymbol{x}}$ satisfying (2) a **minimizer** of f in $\mathcal{X}$, and a **solution** to the optimization problem (1).

By convention, we will focus only on *minimization* problems, noting that $\widehat{\boldsymbol{x}}$ *maximizes* f in $\mathcal{X}$ if and only if $\widehat{\boldsymbol{x}}$ minimizes $-f$ in $\mathcal{X}$ — thus any maximization problem can be easily turned into an equivalent minimization problem.

There are a number of fundamental questions that arise when considering an optimization problem of the form (1):

1. **Existence.** Does a solution to (1) even exist? It could be that f is not bounded from below, or that $\mathcal{X}$ has been defined in such a way as to be empty. How can we guarantee the existence of a solution?
2. **Uniqueness.** Note that an $\widehat{\boldsymbol{x}}$ satisfying (2) need not be unique. Only when the inequality is strict can we conclude that there is a unique (strict) minimizer. When can we conclude that there is a unique solution?
3. **Verification.** Given a candidate solution $\widehat{\boldsymbol{x}}$, is there a simple condition we can check to determine if it is a/the solution to to (1)?
4. **Solution.** Can we find a closed-form expression for a/the solution to (1)? Can we provide an efficient algorithm for computing a/the solution to (1)?

Throughout this course we will devote significant attention to all of these questions, primarily in the context of **convex** problems.

Convex Optimization

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

— R. Tyrrell Rockafellar

Solving optimization problems is in general very difficult. In this class, we will develop a framework for analyzing and solving **convex** programs.[1] To state precisely what we mean by this, recall that a set $\mathcal{C}$ is convex if

$$\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C} \quad \Rightarrow \quad (1-\theta)\boldsymbol{x} + \theta \boldsymbol{y} \in \mathcal{C}$$

for all $\theta \in [0,1]$. A function f is convex if

$$f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \ \leq \ \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y})$$

for all $\boldsymbol{x}, \boldsymbol{y}$ and for all $\theta \in [0,1]$. (If either of these notions are new to you, don't worry. We will have much more on this later!) With these definitions in hand, a convex program simply corresponds to one where

1. The constraint set $\mathcal{X}$ is a convex subset of a real vector space (in this class we will focus exclusively on $\mathcal{X} \subseteq \mathbb{R}^N$).
2. The objective function $f : \mathcal{X} \to \mathbb{R}$ is a convex function.

Often (but not always) we will specify $\mathcal{X}$ using a set of constraint functionals:

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

[1] Throughout this course will use the terminology "optimization/convex *program*" interchangeably with "optimization/convex problem."

In this case, an equivalent way to characterize a convex program is for each of the g_m to be convex functions.

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.
- First-order necessary conditions for optimality turn out to be sufficient. For example, when the problem is unconstrained and smooth, this means we can find an optimal point by finding $\boldsymbol{x}^\star$ such that $\nabla f(\boldsymbol{x}^\star) = \mathbf{0}$.

The upshot of these two things is that if $f(\boldsymbol{x})$ and its derivative (as well as the $g_m(\boldsymbol{x})$ and their derivatives in the case of a constrained problem)[2] are easy to compute, then relatively simple algorithms (e.g., gradient descent) are provably effective at performing the optimization.

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 programs 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 exclusively on the "latest and greatest", we will try to understand the key ideas that are combined in different ways in many solvers.

[2] And as we will see, much of what we do can be naturally extended to non-smooth functions which do not have any derivatives.

Finally, we will talk a lot about *modeling*. That is, how convex optimization appears in signal processing, control systems, machine learning, statistical inference, etc. We will give many examples of mapping a 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.

Convexity and Efficiency

Before going any further, there are two natural questions that you might have that we ought to explicitly address.

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 objective function f and/or its derivative themselves are hard to compute. For example, consider the $(\infty, 1)$ norm:

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

This is a valid matrix norm, and we will see later that all valid norms are convex. But it is known that computing f is NP-hard (see [Roh00]), as is approximating it to a fixed accuracy. Thus, optimization problems involving this quantity (as either the objective function or in the constraints) are bound to be difficult, despite being convex.

Are there any non-convex programs that 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 solution to a nonconvex program, 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 program that we know how to solve:

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

That is, we want the best rank-R approximation (in the least-squares sense) to the $N \times N$ 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:

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

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:

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

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

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.

Note, however, that nonconvex problems can often still be solved in many cases. For instance, consider the "matrix completion" problem described above. Despite being NP-hard *in general*, there are important special cases where we can still solve this problem efficiently. One common trick for dealing with non-convex problems that works here and that we will see later in this course is *convex relaxation*. This approach replaces a non-convex constraint (e.g., the rank constraint above) with a (cleverly chosen) convex surrogate. In some cases (e.g., for a restricted class of matrices $\boldsymbol{A}$ above) one can show that the convex relaxation will have the same solution as the original non-convex problem.

Another approach to non-convex optimization, and one that is popular both in solving the matrix completion problem above as well as in training neural networks, is to simply ignore this non-convexity and to apply standard algorithms like gradient descent that, while derived with convex problems in mind, do not explicitly require convexity to

be applied. While we lose the kinds of theoretical guarantees that we will derive for the convex case, these can still be effective tools in practice.

Next we continue our introduction to convex optimization by introducing a few of the very well-known classes of convex optimization programs and giving some example applications.

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:

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

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:

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

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

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

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)[3] 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:

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

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

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

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

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

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2. \tag{3}$$

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 $\widehat{\boldsymbol{x}} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}$ 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[4] (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:

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

[4] 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

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

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 (3). 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"[5] 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

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

[5] 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[6]

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

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

[6] 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

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

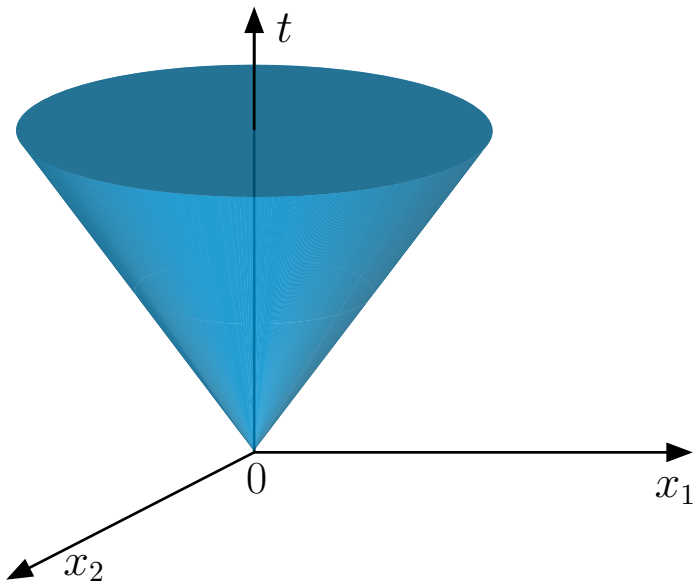
This is another example of a QP with linear constraints.

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.[7] 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

[7] 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.

References

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

I. Convexity

Convex sets

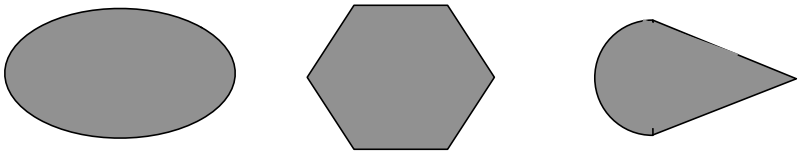
In this section, we will be introduced to some of the mathematical fundamentals of convex sets. In order to motivate some of the definitions, we will look at the *closest point problem* from several different angles. The tools and concepts we develop here, however, have many other applications both in this course and beyond.

A set $\mathcal{C} \subset \mathbb{R}^N$ is **convex** if

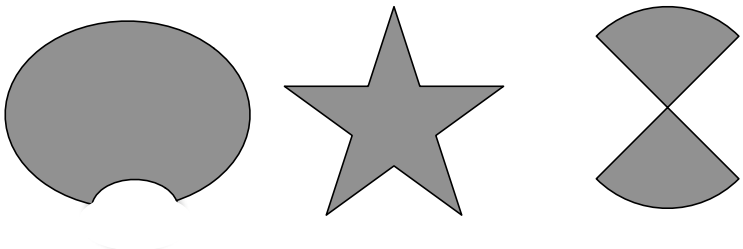
$$\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C} \quad \Rightarrow \quad (1-\theta)\boldsymbol{x} + \theta \boldsymbol{y} \in \mathcal{C} \quad \text{for all } \theta \in [0,1].$$

In English, this means that if we travel on a straight line between any two points in $\mathcal{C}$, then we never leave $\mathcal{C}$.

These sets in $\mathbb{R}^2$ are convex:



These sets are not:



Examples of convex (and nonconvex) sets

- Subspaces. Recall that if $\mathcal{S}$ is a subspace of $\mathbb{R}^N$, then $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S} \Rightarrow a\boldsymbol{x} + b\boldsymbol{y} \in \mathcal{S}$ for all $a, b \in \mathbb{R}$.
  So $\mathcal{S}$ is clearly convex.
- Affine sets. Affine sets are just subspaces that have been offset by the origin:

  $$\{\boldsymbol{x} \in \mathbb{R}^N \ : \ \boldsymbol{x} = \boldsymbol{y} + \boldsymbol{v}, \ \boldsymbol{y} \in \mathcal{T}\}, \ \ \mathcal{T} = \text{subspace},$$

  for some fixed vector $\boldsymbol{v}$. An equivalent definition is that $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C} \Rightarrow \theta\boldsymbol{x} + (1-\theta)\boldsymbol{y} \in \mathcal{C}$ for all $\theta \in \mathbb{R}$ — the difference between this definition and that for a subspace is that subspaces must include the origin.
- Bound constraints. Rectangular sets of the form

  $$\mathcal{C} = \{\boldsymbol{x} \in \mathbb{R}^N : \ell_1 \leq x_1 \leq u_1, \ell_2 \leq x_2 \leq u_2, \ldots, \ell_N \leq x_N \leq u_N\}$$

  for some $\ell_1, \ldots, \ell_N, u_1, \ldots, u_N \in \mathbb{R}$ are convex.
- The *"filled in" simplex* in $\mathbb{R}^N$

  $$\{\boldsymbol{x} \in \mathbb{R}^N : x_1 + x_2 + \cdots + x_N \leq 1, \ x_1, x_2, \ldots, x_N \geq 0\}$$

  is convex.
- Any subset of $\mathbb{R}^N$ that can be expressed as a set of linear inequality constraints

  $$\{\boldsymbol{x} \in \mathbb{R}^N : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$$

  is convex. Notice that both rectangular sets and the simplex

fall into this category — for the previous example, take

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ -1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \\ 0 & \cdots & & & -1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

In general, when sets like these are bounded, the result is a polyhedron.

- Norm balls. If $\|\cdot\|$ is a valid norm on $\mathbb{R}^N$, then

$$\mathcal{B}_r = \{\boldsymbol{x} \ : \ \|\boldsymbol{x}\| \leq r\},$$

is a convex set.

- Ellipsoids. An ellipsoid is a set of the form

$$\mathcal{E} = \{\boldsymbol{x} \ : \ (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \boldsymbol{P}^{-1} (\boldsymbol{x} - \boldsymbol{x}_0) \leq r\},$$

for a symmetric positive-definite matrix $\boldsymbol{P}$. Geometrically, the ellipsoid is centered at $\boldsymbol{x}_0$, its axes are oriented with the eigenvectors of $\boldsymbol{P}$, and the relative widths along these axes are proportional to the eigenvalues of $\boldsymbol{P}$.

- A single point $\{\boldsymbol{x}\}$ is convex.
- The empty set is convex.
- The set

$$\{\boldsymbol{x} \in \mathbb{R}^2 \ : \ x_1^2 - 2x_1 - x_2 + 1 \leq 0\}$$

is convex. (Sketch it!)

- The set
$$\{\boldsymbol{x} \in \mathbb{R}^2 \ : \ x_1^2 - 2x_1 - x_2 + 1 \geq 0\}$$
is **not** convex.
- The set
$$\{\boldsymbol{x} \in \mathbb{R}^2 \ : \ x_1^2 - 2x_1 - x_2 + 1 = 0\}$$
is certainly not convex.
- Sets defined by linear equality constraints where only some of the constraints have to hold are in general not convex. For example
$$\{\boldsymbol{x} \in \mathbb{R}^2 \ : \ x_1 - x_2 \leq -1 \ \text{ and } \ x_1 + x_2 \leq -1\}$$
is convex, while
$$\{\boldsymbol{x} \in \mathbb{R}^2 \ : \ x_1 - x_2 \leq -1 \ \text{ or } \ x_1 + x_2 \leq -1\}$$
is not convex.

Cones

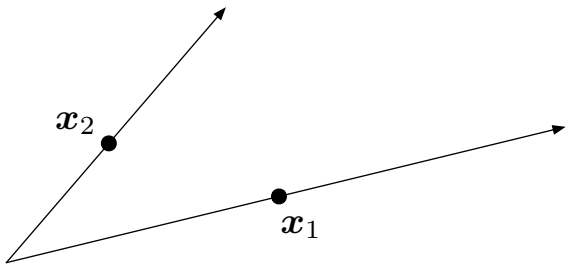
A **cone** is a set $\mathcal{C}$ such that

$$\boldsymbol{x} \in \mathcal{C} \quad \Rightarrow \quad \theta \boldsymbol{x} \in \mathcal{C} \ \text{ for all } \theta \geq 0.$$

Convex cones are sets which are both convex and a cone. $\mathcal{C}$ is a convex cone if

$$\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{C} \quad \Rightarrow \quad \theta_1 \boldsymbol{x}_1 + \theta_2 \boldsymbol{x}_2 \in \mathcal{C} \ \text{ for all } \theta_1, \theta_2 \geq 0.$$

Given an $\boldsymbol{x}_1, \boldsymbol{x}_2$, the set of all linear combinations with positive weights makes a wedge. For practice, sketch the region below that consists of all such combinations of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$:



We will mostly be interested in **proper cones**, which in addition to being convex, are closed, have a non-empty interior[1] ("solid"), and do not contain entire lines ("pointed").

Examples:

Non-negative orthant. The set of vectors whose entries are non-negative,

$$\mathbb{R}^N_+ = \{\boldsymbol{x} \in \mathbb{R}^N \ : \ x_n \geq 0, \text{ for } n = 1, \ldots, N\},$$

is a proper cone.

Positive semi-definite cone. The set of $N \times N$ symmetric matrices with non-negative eigenvalues is a proper cone.

Non-negative polynomials. Vectors of coefficients of non-negative polynomials on $[0, 1]$,

$$\{\boldsymbol{x} \in \mathbb{R}^N \ : \ x_1 + x_2 t + x_3 t^2 + \cdots + x_N t^{N-1} \geq 0 \text{ for all } 0 \leq t \leq 1\},$$

form a proper cone. Notice that it is not necessary that all the $x_n \geq 0$; for example $t - t^2$ $(x_1 = 0, x_2 = 1, x_3 = -1)$ is non-negative on $[0, 1]$.

Norm cones. The subset of $\mathbb{R}^{N+1}$ defined by

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

[1] See Technical Details for precise definition.

is a proper cone for any valid norm $\|\cdot\|$ and $t > 0$. (We encountered this cone in our discussion about SOCPs in the last set of notes.)

Every proper cone $\mathcal{K}$ defines a **partial ordering** or **generalized inequality**. We write

$$\boldsymbol{x} \preceq_{\mathcal{K}} \boldsymbol{y} \quad \text{when} \quad \boldsymbol{y} - \boldsymbol{x} \in \mathcal{K}.$$

For example, for vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$, we say

$$\boldsymbol{x} \preceq_{\mathbb{R}^N_+} \boldsymbol{y} \quad \text{when} \quad x_n \leq y_n \ \text{ for all } n = 1, \ldots, N.$$

For symmetric matrices $\boldsymbol{X}, \boldsymbol{Y}$, we say

$$\boldsymbol{X} \preceq_{S^N_+} \boldsymbol{Y} \quad \text{when} \quad \boldsymbol{Y} - \boldsymbol{X} \text{ has non-negative eigenvalues.}$$

We will typically just use $\preceq$ when the context makes it clear. In fact, for $\mathbb{R}^N_+$ we will just write $\boldsymbol{x} \leq \boldsymbol{y}$ (as we did above) to mean that the entries in $\boldsymbol{x}$ are component-by-component upper-bounded by the entries in $\boldsymbol{y}$.

Partial orderings obey share of the properties of the standard $\leq$ on the real line. For example:

$$\boldsymbol{x} \preceq \boldsymbol{y}, \ \ \boldsymbol{u} \preceq \boldsymbol{v} \quad \Rightarrow \quad \boldsymbol{x} + \boldsymbol{u} \preceq \boldsymbol{y} + \boldsymbol{v}.$$

But other properties do not hold; for example, it is not necessary that either $\boldsymbol{x} \preceq \boldsymbol{y}$ or $\boldsymbol{y} \preceq \boldsymbol{x}$. For an extensive list of properties of partial orderings (most of which will make perfect sense on sight) can be found in [BV04, Chapter 2.4].

Affine sets

Recall the definition of a linear subspace: a set $\mathcal{T} \subset \mathbb{R}^N$ is a subspace if

$$\boldsymbol{x}, \boldsymbol{y} \in \mathcal{T} \quad \Rightarrow \quad \alpha \boldsymbol{x} + \beta \boldsymbol{y} \in \mathcal{T}, \ \text{ for all } \alpha, \beta \in \mathbb{R}.$$

Affine sets (also referred to as affine spaces) are not fundamentally different than subspaces. An affine set $\mathcal{S}$ is simply a subspace that has been offset from the origin:

$$\mathcal{S} = \mathcal{T} + \boldsymbol{v}_0,$$

for some subspace $\mathcal{T}$ and $\boldsymbol{v}_0 \in \mathbb{R}^N$. (It thus make sense to talk about the dimension of $\mathcal{S}$ as being the dimension of this underlying subspace.) We can recast this as a definition similar to the above: a set $\mathcal{S} \subset \mathbb{R}^N$ is affine if

$$\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S} \quad \Rightarrow \quad \lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in \mathcal{S}, \ \text{ for all } \lambda \in \mathbb{R}.$$

Just as we can find the smallest subspace that contains a finite set of vector $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K\}$ by taking their span,

$$\text{Span}(\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K\}) = \left\{ \boldsymbol{x} \in \mathbb{R}^N \ : \ \boldsymbol{x} = \sum_{k=1}^K \alpha_k \boldsymbol{v}_k, \quad \alpha_k \in \mathbb{R} \right\},$$

we can define the **affine hull** (the smallest affine set that contains the vectors) as

$$\text{Aff}(\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K\}) = \left\{ \boldsymbol{x} \in \mathbb{R}^N \ : \ \boldsymbol{x} = \sum_{k=1}^K \lambda_k \boldsymbol{v}_k, \ \lambda_k \in \mathbb{R}, \ \sum_{k=1}^K \lambda_k = 1 \right\}.$$

Example: Let

$$\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{v}_2 = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}.$$

Then $\text{Span}(\{\boldsymbol{v}_1, \boldsymbol{v}_2\})$ is all of $\mathbb{R}^2$ while $\text{Aff}(\{\boldsymbol{v}_1, \boldsymbol{v}_2\})$ is the line that connects $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$,

$$\text{Aff}(\{\boldsymbol{v}_1, \boldsymbol{v}_2\}) = \left\{\boldsymbol{x} \in \mathbb{R}^2 \ : \ x_1 + x_2 = 1\right\}.$$

Just as any linear subspace $\mathcal{T}$ of dimension K can be described using a homogeneous set of equations,

$$\boldsymbol{x} \in \mathcal{T} \ \Leftrightarrow \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0},$$

using any $(N-K) \times N$ matrix $\boldsymbol{A}$ whose nullspace is $\mathcal{T}$, any affine set $\mathcal{S}$ of dimension K can be described as the solution to a linear system of equations

$$\boldsymbol{x} \in \mathcal{S} \ \Leftrightarrow \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

for some $(N-K) \times N$ matrix $\boldsymbol{A}$ and $\boldsymbol{b} \in \mathbb{R}^{N-K}$.

It should be clear that every subspace is an affine set, but not every affine set is a subspace. It is easy to show that an affine set is a subspace if and only if it contains the $\boldsymbol{0}$ vector.

Affine sets are of course convex.

Hyperplanes and halfspaces

Hyperplanes and halfspaces are both very simple constructs, but they will be crucial to our understanding to convex sets, functions, and optimization problems.

A **hyperplane** is an affine set of dimension $N-1$; it has the form

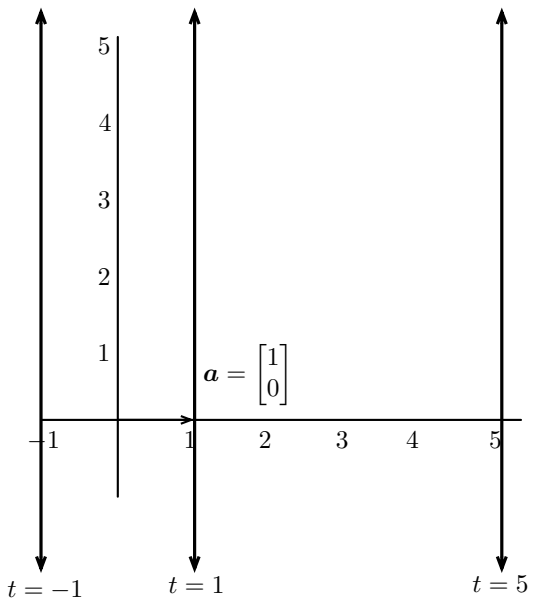
$$\{\boldsymbol{x} \in \mathbb{R}^N \ : \ \langle \boldsymbol{x}, \boldsymbol{a} \rangle = t\}$$

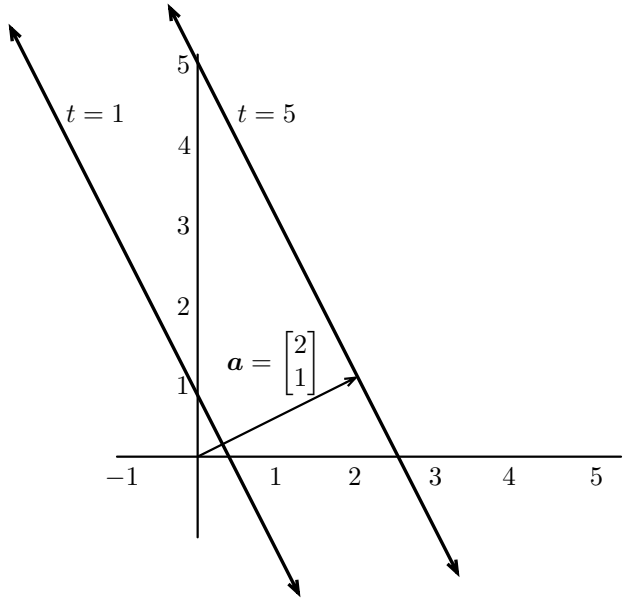
for some fixed vector $\boldsymbol{a} \neq \mathbf{0}$ and scalar t. When $t = 0$, this set is a subspace of dimension $N - 1$, and contains all vectors that are orthogonal to $\boldsymbol{a}$. For $t \neq 0$, this is an affine space consisting of all the vectors orthogonal to $\boldsymbol{a}$ (call this set $\mathcal{A}^\perp$) offset to some $\boldsymbol{x}_0$:

$$\{\boldsymbol{x} \in \mathbb{R}^N \ : \ \langle \boldsymbol{x}, \boldsymbol{a} \rangle = t\} = \{\boldsymbol{x} \in \mathbb{R}^N \ : \ \boldsymbol{x} = \boldsymbol{x}_0 + \mathcal{A}^\perp\},$$

for any $\boldsymbol{x}_0$ with $\langle \boldsymbol{x}_0, \boldsymbol{a} \rangle = t$. We might take $\boldsymbol{x}_0 = t \cdot \boldsymbol{a}/\|\boldsymbol{a}\|_2^2$, for instance. The point is, $\boldsymbol{a}$ is a **normal vector** of the set.

Here are some examples in $\mathbb{R}^2$:

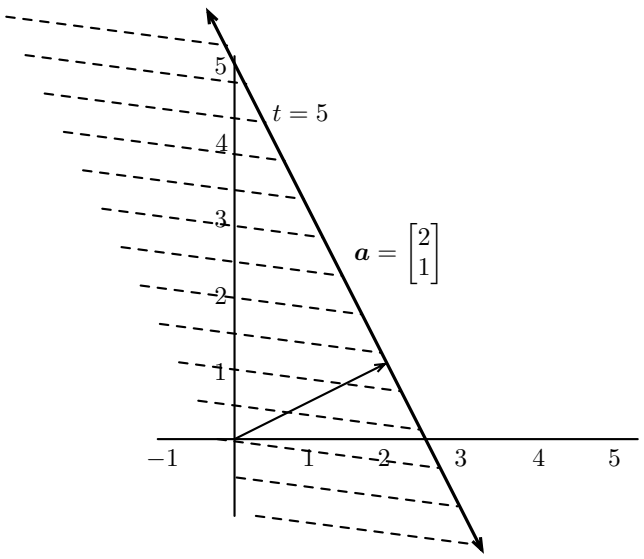




A **halfspace** is a set of the form

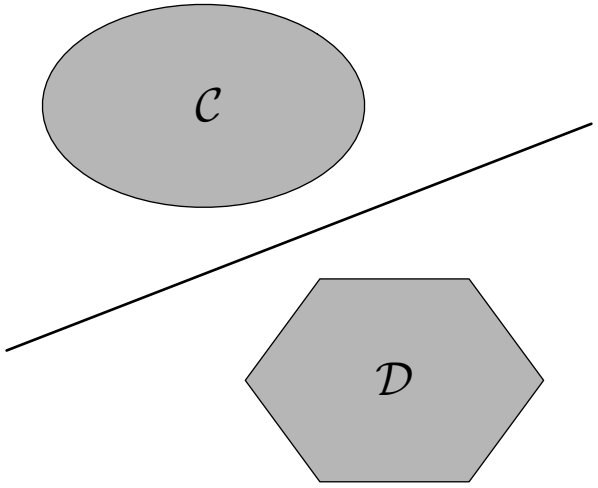
$$\{\boldsymbol{x} \in \mathbb{R}^N \ : \ \langle \boldsymbol{x}, \boldsymbol{a} \rangle \leq t\}$$

for some fixed vector $\boldsymbol{a} \neq \mathbf{0}$ and scalar t. For $t = 0$, the halfspace contains all vectors whose inner product with $\boldsymbol{a}$ is negative (i.e. the angle between $\boldsymbol{x}$ and $\boldsymbol{a}$ is greater than 90°). Here is a simple example:



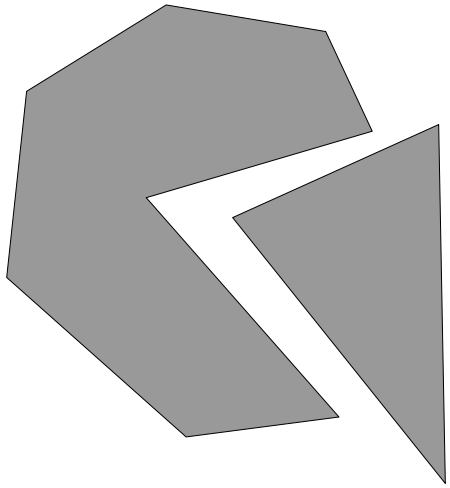
Separating hyperplanes

If two convex sets are disjoint, then there is a hyperplane that separates them. Here is a picture:



This fact is intuitive, and is incredibly useful in understanding the solutions to convex optimization programs (we will see this even in

the next section). It is also not true in general if one of the sets is nonconvex; observe:



For sets $\mathcal{C}, \mathcal{D} \subset \mathbb{R}^N$, we say that a hyperplane $\mathcal{H} = \{\boldsymbol{x} \ : \ \langle \boldsymbol{x}, \boldsymbol{a} \rangle = t\}$

- *separates* $\mathcal{C}$ and $\mathcal{D}$ if for all $\boldsymbol{c} \in \mathcal{C}, \ \boldsymbol{d} \in \mathcal{D}$

$$\langle \boldsymbol{c}, \boldsymbol{a} \rangle \leq t \leq \langle \boldsymbol{d}, \boldsymbol{a} \rangle \quad \text{for all} \ \ \boldsymbol{c} \in \mathcal{C}, \ \boldsymbol{d} \in \mathcal{D}; \tag{1}$$

- *properly separates* $\mathcal{C}$ and $\mathcal{D}$ if (1) holds and both $\mathcal{C}$ and $\mathcal{D}$ are not contained in $\mathcal{H}$ themselves;
- *strictly separates* $\mathcal{C}$ and $\mathcal{D}$ if

$$\langle \boldsymbol{c}, \boldsymbol{a} \rangle < t < \langle \boldsymbol{d}, \boldsymbol{a} \rangle \quad \text{for all} \ \ \boldsymbol{c} \in \mathcal{C}, \ \boldsymbol{d} \in \mathcal{D};$$

- *strongly separates* $\mathcal{C}$ and $\mathcal{D}$ if there exists $\epsilon > 0$ such that

$$\langle \boldsymbol{c}, \boldsymbol{a} \rangle \leq t - \epsilon \ \ \text{and} \ \ \langle \boldsymbol{d}, \boldsymbol{a} \rangle \geq t + \epsilon \quad \text{for all} \ \ \boldsymbol{c} \in \mathcal{C}, \ \boldsymbol{d} \in \mathcal{D}.$$

Note that we can switch the roles of $\mathcal{C}$ and $\mathcal{D}$ above, i.e. we also say $\mathcal{H}$ separates $\mathcal{C}$ and $\mathcal{D}$ if $\langle \boldsymbol{d}, \boldsymbol{a} \rangle \leq t \leq \langle \boldsymbol{c}, \boldsymbol{a} \rangle$ for all $\boldsymbol{c} \in \mathcal{C}, \boldsymbol{d} \in \mathcal{D}$.

Let us start by showing the following:

Strong separating hyperplane theorem
Let $\mathcal{C}$ and $\mathcal{D}$ be disjoint nonempty closed convex sets and let $\mathcal{C}$ be bounded. Then there is a hyperplane that strongly separates $\mathcal{C}$ and $\mathcal{D}$.

To prove this, we show how to explicitly construct a strongly separating hyperplane. Let $d(\boldsymbol{x}, \mathcal{D})$ be the distance of a point $\boldsymbol{x}$ to the set $\mathcal{D}$:

$$d(\boldsymbol{x}, \mathcal{D}) = \inf_{\boldsymbol{y} \in \mathcal{D}} \|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

As we will see below, since $\mathcal{D}$ is closed, there is a unique closest point to $\boldsymbol{x}$ that achieves the infimum on the right. It is also true that $d(\boldsymbol{x}, \mathcal{D})$ is continuous as a function of $\boldsymbol{x}$, so by the Weierstrauss extreme value theorem it achieves its minimum value over the compact set $\mathcal{C}$. That is to say, there exist points $\boldsymbol{c} \in \mathcal{C}$ and $\boldsymbol{d} \in \mathcal{D}$ that achieve the minimum distance

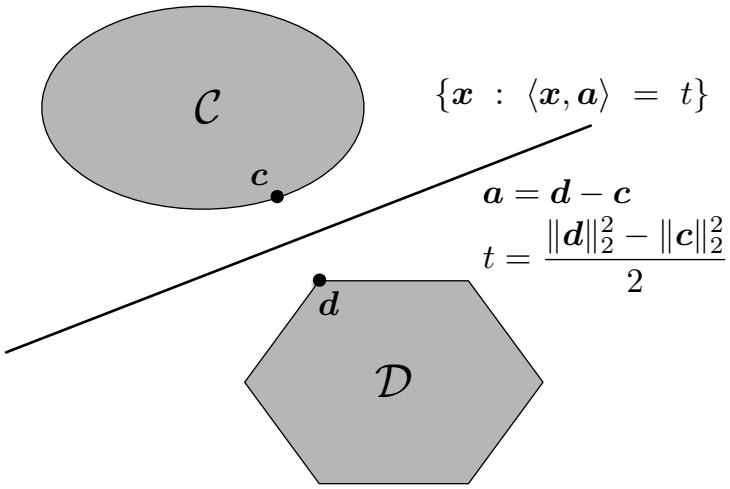
$$\|\boldsymbol{c} - \boldsymbol{d}\|_2 = \inf_{\boldsymbol{x} \in \mathcal{C}, \boldsymbol{y} \in \mathcal{D}} \|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

Since $\mathcal{C}$ and $\mathcal{D}$ are disjoint, we have $\boldsymbol{c} - \boldsymbol{d} \neq 0$.

Define

$$\boldsymbol{a} = \boldsymbol{d} - \boldsymbol{c}, \quad t = \frac{\|\boldsymbol{d}\|_2^2 - \|\boldsymbol{c}\|_2^2}{2}, \quad \epsilon = \frac{\|\boldsymbol{c} - \boldsymbol{d}\|_2^2}{2}.$$

Here is a picture to help visualize these quantities:



We will show that for these choices,

$$\langle \boldsymbol{x}, \boldsymbol{a} \rangle \leq t - \epsilon \text{ for } \boldsymbol{x} \in \mathcal{C}, \quad \langle \boldsymbol{x}, \boldsymbol{a} \rangle \geq t + \epsilon \text{ for } \boldsymbol{x} \in \mathcal{D},$$

for $\epsilon = \|\boldsymbol{c} - \boldsymbol{d}\|_2^2/2$. To see this, we will set

$$f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{a} \rangle - t,$$

and show that for any point $\boldsymbol{u} \in \mathcal{D}$, we have $f(\boldsymbol{u}) \geq \epsilon$.

First, we prove the basic geometric fact that for any two vectors $\boldsymbol{x}, \boldsymbol{y}$,

$$\text{if } \|\boldsymbol{x} + \theta \boldsymbol{y}\|_2 \geq \|\boldsymbol{x}\|_2 \text{ for all } \theta \in [0,1] \quad \text{then} \quad \langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq 0. \quad (2)$$

To establish this, we expand the norm as

$$\|\boldsymbol{x} + \theta \boldsymbol{y}\|_2^2 = \|\boldsymbol{x}\|_2^2 + \theta^2 \|\boldsymbol{y}\|_2^2 + 2\theta \langle \boldsymbol{x}, \boldsymbol{y} \rangle,$$

from which we can immediately deduce that

$$\begin{aligned} &\frac{\theta}{2}\|\boldsymbol{y}\|_2^2 + \langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq 0 \quad \text{for all } \theta \in [0,1] \\ &\qquad \Rightarrow \langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq 0. \end{aligned}$$

Now let $\boldsymbol{u}$ be an arbitrary point in $\mathcal{D}$. Since $\mathcal{D}$ is convex, we know that $\boldsymbol{d} + \theta(\boldsymbol{u} - \boldsymbol{d}) \in \mathcal{D}$ for all $\theta \in [0, 1]$. Since $\boldsymbol{d}$ is as close to $\boldsymbol{c}$ as any other point in $\mathcal{D}$, we have

$$\|\boldsymbol{d} + \theta(\boldsymbol{u} - \boldsymbol{d}) - \boldsymbol{c}\|_2 \ \geq \ \|\boldsymbol{d} - \boldsymbol{c}\|_2,$$

and so by (2), we know that

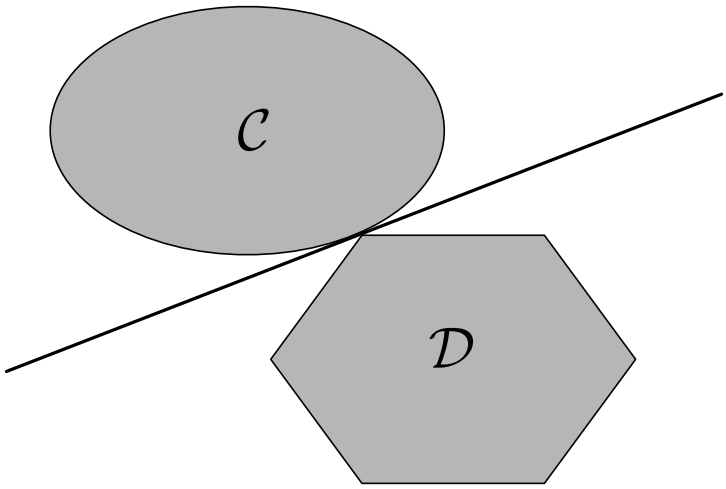
$$\langle \boldsymbol{d} - \boldsymbol{c}, \boldsymbol{u} - \boldsymbol{d} \rangle \ \geq \ 0.$$

This means

$$\begin{aligned}
f(\boldsymbol{u}) &= \langle \boldsymbol{u}, \boldsymbol{d} - \boldsymbol{c} \rangle - \frac{\|\boldsymbol{d}\|_2^2 - \|\boldsymbol{c}\|_2^2}{2} \\
&= \langle \boldsymbol{u}, \boldsymbol{d} - \boldsymbol{c} \rangle - \frac{\langle \boldsymbol{d} + \boldsymbol{c}, \boldsymbol{d} - \boldsymbol{c} \rangle}{2} \\
&= \langle \boldsymbol{u} - (\boldsymbol{d} + \boldsymbol{c})/2, \ \boldsymbol{d} - \boldsymbol{c} \rangle \\
&= \langle \boldsymbol{u} - \boldsymbol{d} + \boldsymbol{d}/2 - \boldsymbol{c}/2, \ \boldsymbol{d} - \boldsymbol{c} \rangle \\
&= \langle \boldsymbol{u} - \boldsymbol{d}, \boldsymbol{d} - \boldsymbol{c} \rangle + \frac{\|\boldsymbol{c} - \boldsymbol{d}\|_2^2}{2} \\
&\geq \frac{\|\boldsymbol{c} - \boldsymbol{d}\|_2^2}{2}.
\end{aligned}$$

The argument that $f(\boldsymbol{v}) \leq -\epsilon$ for every $\boldsymbol{v} \in \mathcal{C}$ is exactly the same.

We will not prove it here, but there is an even more interesting result that says that the sets $\mathcal{C}$ and $\mathcal{D}$ do not even have to be disjoint — they can intersect at one or more points along their boundaries as shown here:



Separating hyperplane theorem

Nonempty convex sets $\mathcal{C}, \mathcal{D} \subset \mathbb{R}^N$ can be (properly) separated by a hyperplane if and only if their relative interiors are disjoint:

$$\text{relint}(\mathcal{C}) \cap \text{relint}(\mathcal{D}) = \emptyset.$$

See the Technical Details for what exactly is meant by "relative interior" but it is basically everything not on the natural boundary of the set once we account for the fact that it might have dimension smaller than N.

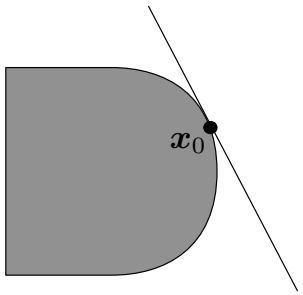
Supporting hyperplanes

A direct consequence of the separating hyperplane theorem is that every point on the boundary of a convex set $\mathcal{C}$ can be separated from its interior.

If $\boldsymbol{a} \neq \mathbf{0}$ satisfies $\langle \boldsymbol{x}, \boldsymbol{a} \rangle \leq \langle \boldsymbol{x}_0, \boldsymbol{a} \rangle$ for all $\boldsymbol{x} \in \mathcal{C}$, then

$$\mathcal{H} = \{\boldsymbol{x} : \langle \boldsymbol{x}, \boldsymbol{a} \rangle = \langle \boldsymbol{x}_0, \boldsymbol{a} \rangle\}$$

is called a **supporting hyperplane** to $\mathcal{C}$ at $\boldsymbol{x}_0$. Here's a picture:



The hyperplane is tangent to $\mathcal{C}$ at $\boldsymbol{x}_0$, and the halfspace $\{\boldsymbol{x} : \langle \boldsymbol{x}, \boldsymbol{a} \rangle \leq \langle \boldsymbol{x}_0, \boldsymbol{a} \rangle\}$ contains $\mathcal{C}$. We call $\mathcal{H}$ a **proper** supporting hyperplane if $\langle \boldsymbol{y}, \boldsymbol{a} \rangle < \langle \boldsymbol{x}_0, \boldsymbol{a} \rangle$ for at least one $y \in \mathcal{C}$.

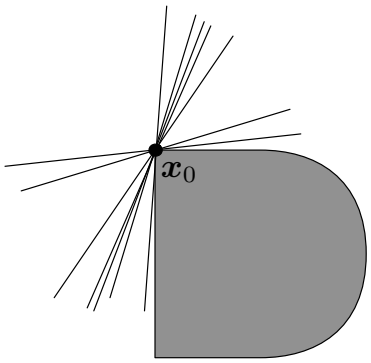
The supporting hyperplane theorem says that a supporting hyperplane exists at every point $\boldsymbol{x}_0$ on the boundary of a (non-empty) convex set $\mathcal{C}$.

Supporting hyperplane theorem

Let $\mathcal{C} \subset \mathbb{R}^N$ be a convex set, and let $\boldsymbol{x}_0 \in \mathcal{C}$. Then there is a supporting hyperplane at $\boldsymbol{x}_0$ if and only if $\boldsymbol{x}_0 \notin \text{relint}(\mathcal{C})$.

Proof of this theorem (and the general separating hyperplane theorem above) can be found in [Roc70, Chap. 11].

Note that there might be more than one supporting hyperplane at an boundary point:



The closest point problem

Let $\boldsymbol{x}_0 \in \mathbb{R}^N$ be given, and let $\mathcal{C}$ be a non-empty, closed, convex set. The **projection** of $\boldsymbol{x}_0$ onto $\mathcal{C}$ is the closest point (in the standard Euclidean distance, for now) in $\mathcal{C}$ to $\boldsymbol{x}_0$:

$$P_{\mathcal{C}}(\boldsymbol{x}_0) = \arg\min_{\boldsymbol{y} \in \mathcal{C}} \|\boldsymbol{x}_0 - \boldsymbol{y}\|_2$$

We will see below that there is a unique minimizer to this problem, and that the solution has geometric properties that are analogous to the case where $\mathcal{C}$ is a subspace.

Projection onto a subspace

Let's recall how we solve this problem in the special case where $\mathcal{C} := \mathcal{T}$ is a K-dimensional *subspace*. In this case, the solution

$\hat{\boldsymbol{x}} = P_{\mathcal{T}}(\boldsymbol{x}_0)$ is unique, and is characterized by the **orthogonality principle**:

$$\boldsymbol{x}_0 - \hat{\boldsymbol{x}} \perp \mathcal{T}$$

meaning that $\langle \boldsymbol{y}, \boldsymbol{x}_0 - \hat{\boldsymbol{x}} \rangle = 0$ for all $\boldsymbol{y} \in \mathcal{T}$. The proof of this fact is reviewed in the Technical Details section at the end of these notes.

The orthogonality principle leads immediately to an algorithm for calculating $P_{\mathcal{T}}(\boldsymbol{x}_0)$. Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K$ be a basis for $\mathcal{T}$; we can write the solution as

$$\hat{\boldsymbol{x}} = \sum_{k=1}^{K} \alpha_k \boldsymbol{v}_k;$$

solving for the expansion coefficients α_k is the same as solving for $\hat{\boldsymbol{x}}$. We know that

$$\langle \boldsymbol{x}_0 - \sum_{j=1}^{K} \alpha_j \boldsymbol{v}_j, \boldsymbol{v}_k \rangle = 0, \quad \text{for } k = 1, \ldots, K,$$

and so the α_k must obey the linear system of equations

$$\sum_{j=1}^{K} \alpha_j \langle \boldsymbol{v}_j, \boldsymbol{v}_k \rangle = \langle \boldsymbol{x}_0, \boldsymbol{v}_k \rangle, \quad \text{for } k = 1, \ldots, K.$$

Concatenating the $\boldsymbol{v}_k$ as columns in the $N \times K$ matrix $\boldsymbol{V}$, and the entries α_k into the vector $\boldsymbol{\alpha} \in \mathbb{R}^K$, we can write the equations above as

$$\boldsymbol{V}^{\mathrm{T}} \boldsymbol{V} \boldsymbol{\alpha} = \boldsymbol{V}^{\mathrm{T}} \boldsymbol{x}_0.$$

Since the $\{\boldsymbol{v}_k\}$ are a basis for $\mathcal{T}$ (i.e. they are linearly independent), $\boldsymbol{V}^{\mathrm{T}} \boldsymbol{V}$ is invertible, and we can solve for the best expansion coefficients:

$$\hat{\boldsymbol{\alpha}} = (\boldsymbol{V}^{\mathrm{T}} \boldsymbol{V})^{-1} \boldsymbol{V}^{\mathrm{T}} \boldsymbol{x}_0.$$

Using these expansion coefficient to reconstruct $\hat{\boldsymbol{x}}$ yields

$$\hat{\boldsymbol{x}} = \boldsymbol{V}\hat{\boldsymbol{\alpha}} = \boldsymbol{V}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V})^{-1}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{x}_0.$$

In this case, the projector $P_{\mathcal{T}}(\cdot)$ is a **linear map**, specified by $N \times N$ matrix $\boldsymbol{V}(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V})^{-1}\boldsymbol{V}^{\mathrm{T}}$.

Projection onto an affine set

As discussed above, affine sets are not fundamentally different than subspaces; any affine set $\mathcal{C}$ can be written as a subspace $\mathcal{T}$ plus an offset $\boldsymbol{v}_0$:

$$\mathcal{C} = \mathcal{T} + \boldsymbol{v}_0 = \{\boldsymbol{x} \ : \ \boldsymbol{x} = \boldsymbol{y} + \boldsymbol{v}_0, \ \boldsymbol{y} \in \mathcal{C}\}.$$

This makes it easy to translate the results for subspaces above to say that the projection onto an affine set is unique, and obeys the orthogonality principle

$$\langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \boldsymbol{x}_0 - \hat{\boldsymbol{x}} \rangle = 0, \quad \text{for all } \ \boldsymbol{y} \in \mathcal{C}. \tag{3}$$

You can solve this problem by shifting $\boldsymbol{x}_0$ and $\mathcal{C}$ by negative $\boldsymbol{v}_0$, projecting $\boldsymbol{x}_0 - \boldsymbol{v}_0$ onto the subspace $\mathcal{C} - \boldsymbol{v}_0$, and then shifting the answer back.

Projection onto a general convex set

In general, there is no closed-form expression for the projector onto a given convex set. However, the concepts above (orthogonality, projection onto a subspace) can help us understand the solution for an arbitrary convex set.

Uniqueness of closest point

If $\mathcal{C} \subset \mathbb{R}^N$ is closed and convex, then for any $\boldsymbol{x}_0$, the program

$$\underset{\boldsymbol{y} \in \mathcal{C}}{\text{minimize}} \, \|\boldsymbol{x}_0 - \boldsymbol{y}\|_2 \tag{4}$$

has a unique solution.

First, let's argue that at least one minimizer to (4). Let $\boldsymbol{x}'$ be any point in $\mathcal{C}$, and set $\mathcal{B} = \{\boldsymbol{x} \ : \ \|\boldsymbol{x} - \boldsymbol{x}_0\| \leq \|\boldsymbol{x}' - \boldsymbol{x}_0\|_2\}$. By construction, if a minimizer exists, it must be in the set $\mathcal{C} \cap \mathcal{B}$. Since $\mathcal{C} \cap \mathcal{B}$ is closed and bounded and $\|\boldsymbol{x}_0 - \boldsymbol{y}\|_2$ is a continuous function of $\boldsymbol{y}$, by the Weierstrass extreme value theorem we know that there is at least one point in the set where this functions achieves it minimum. Hence there exists at least one solution $\hat{\boldsymbol{x}}$ to (4).

We can now argue that $|hat\boldsymbol{x}$ is the only minimizer of (4). Consider first all the points $\boldsymbol{y} \in \mathcal{C}$ such that $\boldsymbol{y} - \boldsymbol{x}_0$ is co-aligned with $\hat{\boldsymbol{x}} - \boldsymbol{x}_0$. Let

$$\mathcal{I} = \{\alpha \in \mathbb{R} \ : \ \hat{\boldsymbol{x}} + \alpha(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}) \in \mathcal{C}\}.$$

(Note that if $\boldsymbol{y} = \hat{\boldsymbol{x}} + \alpha(\boldsymbol{x}_0 - \hat{\boldsymbol{x}})$, then $\boldsymbol{y} - \boldsymbol{x}_0 = (1 - \alpha)(\hat{\boldsymbol{x}} - \boldsymbol{x}_0)$ and so the two difference vectors are co-aligned.) Since $\mathcal{C}$ is convex and closed, this is a closed interval of the real line (that contains at least the point $\alpha = 0$). The function

$$g(\alpha) = \|\boldsymbol{x}_0 - \hat{\boldsymbol{x}} - \alpha(\boldsymbol{x}_0 - \hat{\boldsymbol{x}})\|_2^2 = (1 - \alpha)^2 \|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2^2,$$

captures the distance of the co-aligned vector for every value of α. Since, as a function of α, this a parabola with strictly positive second derivative, it takes its minima at exactly one place on the interval $\mathcal{I}$ and by construction this is $\alpha = 0$. So any $\boldsymbol{y} \neq \hat{\boldsymbol{x}}$ with $\boldsymbol{y} - \boldsymbol{x}_0$ co-aligned with $\hat{\boldsymbol{x}} - \boldsymbol{x}_0$ cannot be another minimizer of (4).

Now let $\boldsymbol{y}$ be any point in $\mathcal{C}$ such that the difference vectors are not co-aligned. We will show that $\boldsymbol{y}$ cannot minimize (4) because the point $\hat{\boldsymbol{x}}/2 + \boldsymbol{y}/2 \in \mathcal{C}$ is definitively closer to $\boldsymbol{x}_0$. We have

$$\begin{aligned}
\left\| \boldsymbol{x}_0 - \frac{\hat{\boldsymbol{x}}}{2} - \frac{\boldsymbol{y}}{2} \right\|_2^2 &= \left\| \frac{\boldsymbol{x}_0 - \hat{\boldsymbol{x}}}{2} + \frac{\boldsymbol{x}_0 - \boldsymbol{y}}{2} \right\|_2^2 \\
&= \frac{\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2^2}{4} + \frac{\|\boldsymbol{x}_0 - \hat{\boldsymbol{y}}\|_2^2}{4} + \frac{\langle \boldsymbol{x}_0 - \hat{\boldsymbol{x}}, \boldsymbol{x}_0 - \boldsymbol{y} \rangle}{2} \\
&< \frac{\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2^2}{4} + \frac{\|\boldsymbol{x}_0 - \hat{\boldsymbol{y}}\|_2^2}{4} + \frac{\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2 \, \|\boldsymbol{x}_0 - \boldsymbol{y}\|_2}{2} \\
&= \left(\frac{\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2}{2} + \frac{\|\boldsymbol{x}_0 - \boldsymbol{y}\|_2}{2} \right)^2 \\
&\leq \|\boldsymbol{x}_0 - \boldsymbol{y}\|_2^2.
\end{aligned}$$

The strict inequality above follows from Cauchy-Schwarz, while the last inequality follows from the fact that $\hat{\boldsymbol{x}}$ is a minimizer. This shows that no $\boldsymbol{y} \neq \hat{\boldsymbol{x}}$ can also minimize (4), and so $\hat{\boldsymbol{x}}$ is unique.

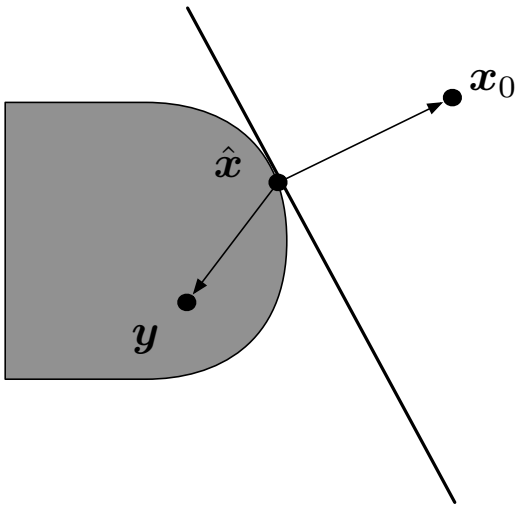
Similar to the orthogonality principle for projecting onto an affine set or linear space, there is a clean geometric optimality condition for the closest point in a convex set.

Obtuseness principle

$P_{\mathcal{C}}(\boldsymbol{x}_0) = \hat{\boldsymbol{x}}$ if and only if

$$\langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \boldsymbol{x}_0 - \hat{\boldsymbol{x}} \rangle \leq 0 \quad \text{for all} \quad \boldsymbol{y} \in \mathcal{C}. \tag{5}$$

Compare (5) with (3) above. Here is a picture:



We first prove that (5)$\Rightarrow P_{\mathcal{C}}(\boldsymbol{x}_0) = \hat{\boldsymbol{x}}$. For any $\boldsymbol{y} \in \mathcal{C}$, we have

$$\begin{aligned}\|\boldsymbol{y} - \boldsymbol{x}_0\|_2^2 &= \|\boldsymbol{y} - \hat{\boldsymbol{x}} + \hat{\boldsymbol{x}} - \boldsymbol{x}_0\|_2^2 \\ &= \|\boldsymbol{y} - \hat{\boldsymbol{x}}\|_2^2 + \|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|_2^2 + 2\langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \hat{\boldsymbol{x}} - \boldsymbol{x}_0 \rangle \\ &\geq \|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|_2^2 + 2\langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \hat{\boldsymbol{x}} - \boldsymbol{x}_0 \rangle.\end{aligned}$$

Note that the inner product term above is the same as (5), but with the sign of the second argument flipped, so we know this term must be non-negative. Thus

$$\|\boldsymbol{y} - \boldsymbol{x}_0\|_2^2 \;\geq\; \|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|_2^2.$$

Since this holds uniformly over all $\boldsymbol{y} \in \mathcal{C}$, $\hat{\boldsymbol{x}}$ must be the closest point in $\mathcal{C}$ to $\boldsymbol{x}_0$.

We now show that $P_{\mathcal{C}}(\boldsymbol{x}_0) = \hat{\boldsymbol{x}} \;\Rightarrow\; (5)$. Let $\boldsymbol{y}$ be an arbitrary point in $\mathcal{C}$. Since $\mathcal{C}$ is convex, the point $\hat{\boldsymbol{x}} + \theta(\boldsymbol{y} - \hat{\boldsymbol{x}})$ must also be in $\mathcal{C}$ for all $\theta \in [0, 1]$. Since $\hat{\boldsymbol{x}} = P_{\mathcal{C}}(\boldsymbol{x}_0)$,

$$\|\hat{\boldsymbol{x}} + \theta(\boldsymbol{y} - \hat{\boldsymbol{x}}) - \boldsymbol{x}_0\|_2 \;\geq\; \|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|_2 \quad \text{for all} \quad \theta \in [0, 1].$$

By our intermediate result (2) a few pages ago, this means

$$\langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \hat{\boldsymbol{x}} - \boldsymbol{x}_0 \rangle \;\geq\; 0 \quad \Rightarrow \quad \langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \boldsymbol{x}_0 - \hat{\boldsymbol{x}} \rangle \;\leq\; 0.$$

Technical Details: closest point to a subspace

In this section, we establish the orthogonality principle for projection of a point $\boldsymbol{x}_0$ onto a subspace $\mathcal{T}$. Let $\hat{\boldsymbol{x}}$ be a vector which obeys

$$\hat{\boldsymbol{e}} = \boldsymbol{x} - \hat{\boldsymbol{x}} \ \perp \ \mathcal{T}.$$

We will show that $\hat{\boldsymbol{x}}$ is the unique closest point to $\boldsymbol{x}_0$ in $\mathcal{T}$. Let $\boldsymbol{y}$ be any other vector in $\mathcal{T}$, and set

$$\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{y}.$$

We will show that

$$\|\boldsymbol{e}\| \ > \ \|\hat{\boldsymbol{e}}\| \quad (\text{i.e. that } \|\boldsymbol{x} - \boldsymbol{y}\| \ > \ \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|)\,.$$

Note that

$$\begin{aligned}\|\boldsymbol{e}\|^2 = \|\boldsymbol{x} - \boldsymbol{y}\|^2 &= \|\hat{\boldsymbol{e}} - (\boldsymbol{y} - \hat{\boldsymbol{x}})\|^2 \\ &= \langle \hat{\boldsymbol{e}} - (\boldsymbol{y} - \hat{\boldsymbol{x}}) \ , \ \hat{\boldsymbol{e}} - (\boldsymbol{y} - \hat{\boldsymbol{x}}) \rangle \\ &= \|\hat{\boldsymbol{e}}\|^2 + \|\boldsymbol{y} - \hat{\boldsymbol{x}}\|^2 - \langle \hat{\boldsymbol{e}}, \boldsymbol{y} - \hat{\boldsymbol{x}} \rangle - \langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \hat{\boldsymbol{e}} \rangle.\end{aligned}$$

Since $\boldsymbol{y} - \hat{\boldsymbol{x}} \in \mathcal{T}$ and $\hat{\boldsymbol{e}} \perp \mathcal{T}$,

$$\langle \hat{\boldsymbol{e}}, \boldsymbol{y} - \hat{\boldsymbol{x}} \rangle = 0, \quad \text{and} \quad \langle \boldsymbol{y} - \hat{\boldsymbol{x}}, \hat{\boldsymbol{e}} \rangle = 0,$$

and so

$$\|\boldsymbol{e}\|^2 = \|\hat{\boldsymbol{e}}\|^2 + \|\boldsymbol{y} - \hat{\boldsymbol{x}}\|^2.$$

Since all three quantities in the expression above are positive and

$$\|\boldsymbol{y} - \hat{\boldsymbol{x}}\| > 0 \quad \Leftrightarrow \quad \boldsymbol{y} \neq \hat{\boldsymbol{x}},$$

we see that

$$\boldsymbol{y} \neq \hat{\boldsymbol{x}} \quad \Leftrightarrow \quad \|\boldsymbol{e}\| > \|\hat{\boldsymbol{e}}\|.$$

We leave it as an exercise to establish the converse; that if $\langle \boldsymbol{y}, \hat{\boldsymbol{x}} - \boldsymbol{x}_0 \rangle = 0$ for all $\boldsymbol{y} \in \mathcal{T}$, then $\hat{\boldsymbol{x}}$ is the projection of $\boldsymbol{x}_0$ onto $\mathcal{T}$.

Technical Details: Basic analysis in $\mathbb{R}^N$

This section contains a brief review of basic topological concepts in $\mathbb{R}^N$. Our discussion will take place using the standard Euclidean distance measure (i.e. ℓ_2 norm), but all of these definitions can be generalized to other metrics. An excellent source for this material is [Rud76].

Basic topology

We say that a sequence of vectors $\{\boldsymbol{x}_k,\ k = 1, 2, \ldots\}$ **converges** to $\hat{\boldsymbol{x}}$ if

$$\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|_2 \to 0 \quad \text{as} \quad k \to \infty.$$

More precisely, this means that for every $\epsilon > 0$, there exists an n_ϵ such that

$$\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|_2 \leq \epsilon \ \text{ for all } \ k \geq n_\epsilon.$$

It is easy to show that a sequence of vectors converge if and only if their individual components converge point-by-point. For a convergent sequence like the above, we often write $\lim_{k\to\infty} \boldsymbol{x}_k = \hat{\boldsymbol{x}}$.

A set $\mathcal{X}$ is **open** if we can draw a small ball around every point in $\mathcal{X}$ which is also entirely contained in $\mathcal{X}$. More precisely, let $\mathcal{B}(\boldsymbol{x}, \epsilon)$ be the set of all points within ϵ of $\boldsymbol{x}$:

$$\mathcal{B}(\boldsymbol{x}, \epsilon) = \{\boldsymbol{y} \in \mathbb{R}^N \ : \ \|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \epsilon\}.$$

Then $\mathcal{X}$ is open if for every $\boldsymbol{x} \in \mathcal{X}$, there exists an $\epsilon_{\boldsymbol{x}} > 0$ such that $\mathcal{B}(\boldsymbol{x}, \epsilon_{\boldsymbol{x}}) \subset \mathcal{X}$. The standard example here is open intervals of the real line, e.g. $(0, 1)$.

There are many ways to define **closed** sets. The easiest is that a set $\mathcal{X}$ is closed if its complement is open. A more illuminating (and

equivalent) definition is that $\mathcal{X}$ is closed if it contains all of its limit points. A vector $\hat{\boldsymbol{x}}$ is a **limit point** of $\mathcal{X}$ if there exists a sequence of vectors $\{\boldsymbol{x}_k\} \subset \mathcal{X}$ that converge to $\hat{\boldsymbol{x}}$.

The **closure** of general set $\mathcal{X}$, denoted $\text{cl}(\mathcal{X})$, is the set of all limit points of $\mathcal{X}$. Note that every $\boldsymbol{x} \in \mathcal{X}$ is trivially a limit point (take the sequence $\boldsymbol{x}_k = \boldsymbol{x}$), so $\mathcal{X} \subset \text{cl}(\mathcal{X})$. By construction, $\text{cl}(\mathcal{X})$ is the smallest closed set that contains $\mathcal{X}$.

The set $\mathcal{X}$ is **bounded** if we can find a uniform upper bound on the distance between two points it contains; this upper bound is commonly referred to as the **diameter** of the set:

$$\text{diam}\,\mathcal{X} = \sup_{\boldsymbol{x},\boldsymbol{y} \in \mathcal{X}} \|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

The set $\mathcal{X} \subset \mathbb{R}^N$ is **compact** if it is closed and bounded. A key fact about compact sets is that every sequence has a convergent subsequence — this is known as the Bolzano-Weierstrauss theorem.

Interiors and boundaries of sets

Related to the definition of open and closed sets are the technical definitions of boundary and interior. The **interior** of a set $\mathcal{X}$ is the collection of points around which we can place a ball of finite width which remains in the set:

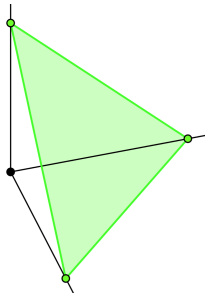
$$\text{int}(\mathcal{X}) = \{\boldsymbol{x} \in \mathcal{X} \ : \ \exists \epsilon > 0 \ \text{ such that } \ \mathcal{B}(\boldsymbol{x}, \epsilon) \subset \mathcal{X}\}.$$

We will actually be more concerned with the concept of **relative interior**. Let's motivate this quickly with an example. Consider

the unit simplex in $\mathbb{R}^3$

$$\Delta = \{\boldsymbol{x} \in \mathbb{R}^3 \ : \ x_1 + x_2 + x_3 = 1, \ \ x_i \geq 0\}.$$

This is essentially a "flat" (two dimensional) triangle embedded into three dimensional space. Here is a picture:



By the technical definition of interior above, no point in Δ is an interior point, as any ball drawn around a $\boldsymbol{x}_0 \in \Delta$ will contain points points with $x_1 + x_2 + x_3 \neq 1$. But somehow this does not capture the fact that the 2D triangle itself has "points on the edges" and "points in the middle".

To rectify this, we introduce the relative interior of a set as

$$\text{relint}(\mathcal{X}) = \{\boldsymbol{x} \in \mathcal{X} \ : \ \exists \epsilon > 0 \ \text{ such that } \ \mathcal{B}(\boldsymbol{x}, \epsilon) \cap \text{Aff}(\mathcal{X}) \subset \mathcal{X}\}$$

where $\text{Aff}(\mathcal{X})$ is the smallest affine set that contains $\mathcal{X}$. This means that if the set we are analyzing can be embedded in a low-dimensional affine space, then we define interior points relative to this set. For the simplex, we have

$$\text{Aff}(\Delta) = \{\boldsymbol{x} \ : \ x_1 + x_2 + x_3 = 1\},$$

and

$$\text{relint}(\Delta) = \{\boldsymbol{x} \in \Delta \ : \ x_1, x_2, x_3 > 0\}.$$

For convex sets, we also have the equivalent and perhaps more intuitive definition of relative interior,

$$\text{relint}(\mathcal{C}) = \{\boldsymbol{x} \in \mathcal{C} \,:\, \exists \boldsymbol{y} \in \mathcal{C},\, \exists \lambda > 1 \text{ such that } \lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in \mathcal{C}\}.$$

The **boundary** of $\mathcal{X}$ is the set of points in $\text{cl}(\mathcal{X})$ that are not in the (relative) interior:

$$\text{bd}(\mathcal{X}) = \text{cl}(\mathcal{X}) \backslash \text{relint}(\mathcal{X}).$$

Functions, continuity, and extrema

A function $f : \mathbb{R}^N \to \mathbb{R}$ is **continuous** if for every $\epsilon > 0$ there exists a $\delta > 0$ such that

$$\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2 \leq \delta \quad \Rightarrow \quad |f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| \leq \epsilon,$$

for all $\boldsymbol{x}_1, \boldsymbol{x}_2$. f is called **Lipschitz** if the δ can be taken proportional to ϵ; there exists an L such that

$$|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| \leq L\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2, \quad \text{for all } \boldsymbol{x}_1, \boldsymbol{x}_2.$$

A function f is called **bounded** on a set $\mathcal{X}$ if there exists an M such that $|f(\boldsymbol{x})| \leq M$ for all $\boldsymbol{x} \in \mathcal{X}$. The **supremum** of f on $\mathcal{X}$ is the smallest upper bound on f and the **infimum** of f is that greatest lower bound. We have these terms since maximizers and minimizers do not always exist. For example

$$\min_{x \geq 0} \; e^{-x}$$

does not exist; there is no value x_0 that we can choose where we can definitively say that $e^{-x_0} \leq e^{-x}$ for all $x \geq 0$. The infimum, however, always exists:

$$\inf_{x \geq 0} \; e^{-x} = 0.$$

When there is a point in $\mathcal{X}$ where f achieves its infimum, then of course the operations agree, e.g.

$$\inf_{x\in[0,1]}(x-1/2)^2 = \min_{x\in[0,1]}(x-1/2)^2 = 0.$$

It is also true that every continuous function on a compact set is bounded. The **Weierstrass extreme value theorem** tells us even more; it states that a continuous function always achieves its infimum (minimizer) and supremem (maximizer) over a compact set $\mathcal{X}$. That is, there exists $x^\star \in \mathcal{X}$ such that

$$f(\boldsymbol{x}^\star) = \sup_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}),$$

and $x_\star \in \mathcal{X}$ such that

$$f(\boldsymbol{x}_\star) = \inf_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}).$$

Because of this, we can freely replace sup with max and inf with min. This might be viewed as a fundamental results in optimization, as it gives loose conditions

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Roc70] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[Rud76] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 1976.

Convex functions

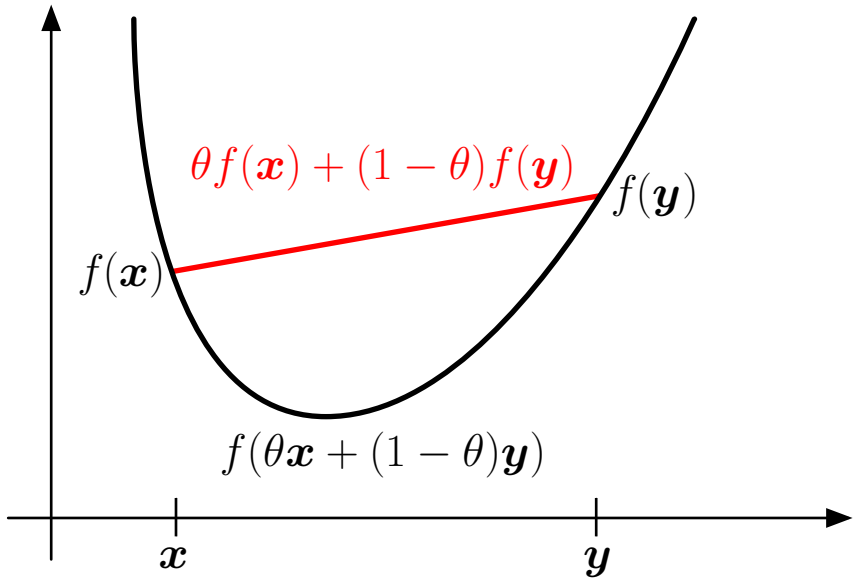
We have seen what it means for a set to be convex. In this set of notes, we start working towards what it means to be a convex *function*.

To define this concept rigorously, we must be specific about the subset of $\mathbb{R}^N$ where a function can be applied. Specifically, the **domain** $\text{dom}\, f$ of a function $f : \mathbb{R}^N \to \mathbb{R}^M$ is the subset of $\mathbb{R}^N$ where f is well-defined. We then say that a function f is **convex** if $\text{dom}\, f$ is a convex set, and

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

for all $\boldsymbol{x}, \boldsymbol{y} \in \text{dom}\, f$ and $0 \leq \theta \leq 1$.

This inequality is easier to interpret with a picture. The left-hand side of the inequality above is simply the function f evaluated along a line segment between $\boldsymbol{x}$ and $\boldsymbol{y}$. The right-hand side represents a straight line segment between $f(\boldsymbol{x})$ and $f(\boldsymbol{y})$ as we move along this line segment, which for a convex function must lie above f.



We say that f is **strictly convex** if dom f is convex and

$$f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \;<\; \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y})$$

for all $\boldsymbol{x} \neq \boldsymbol{y} \in \text{dom } f$ and $0 < \theta < 1$.

Note also that we say that a function is f is **concave** if $-f$ is convex, and similarly for strictly concave functions. We are mostly interested in convex functions, but this is only because we are mostly restricting our attention to *minimization* problems. We justified this because any maximization problem can be converted to a minimization one by multiplying the objective function by -1. Everything that we say about minimizing convex functions also applies maximizing concave ones.

We make a special note here that *affine* functions of the form

$$f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{a} \rangle + b,$$

are both convex and concave (but neither strictly convex nor strictly concave). This is the only kind of function that has this property. (Why?)

Note that in the definition above, the domain matters. For example,

$$f(x) = x^3$$

is convex if dom $f = \mathbb{R}_+ = [0, \infty]$ but not if dom $f = \mathbb{R}$.

It will also sometimes be useful to consider the **extension** of f from dom f to all of $\mathbb{R}^N$, defined as

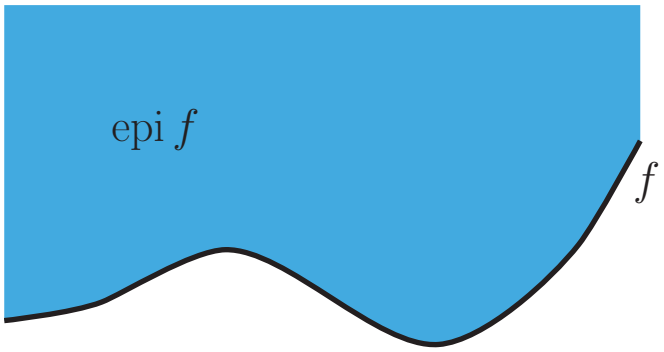
$$\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \text{dom } f, \quad \tilde{f}(\boldsymbol{x}) = +\infty, \quad \boldsymbol{x} \notin \text{dom } f.$$

If f is convex on dom f, then its extension is also convex on $\mathbb{R}^N$.

The epigraph

A useful notion that we will encounter later in the course is that of the **epigraph** of a function. The epigraph of a function $f : \mathbb{R}^N \to \mathbb{R}$ is the subset of $\mathbb{R}^{N+1}$ created by filling in the space above f:

$$\operatorname{epi} f = \left\{ \begin{bmatrix} \boldsymbol{x} \\ t \end{bmatrix} \in \mathbb{R}^{N+1} \ : \ \boldsymbol{x} \in \operatorname{dom} f, \quad f(\boldsymbol{x}) \leq t \right\}.$$



It is not hard to show that f is convex if and only if epi f is a convex set. This connection should help to illustrate how even though the definitions of a convex set and convex function might initially appear quite different, they actually follow quite naturally from each other.

Examples of convex functions

Here are some standard examples for functions on $\mathbb{R}$:

- $f(x) = x^2$ is (strictly) convex.
- affine functions $f(x) = ax + b$ are both convex and concave for $a, b \in \mathbb{R}$.
- exponentials $f(x) = e^{ax}$ are convex for all $a \in \mathbb{R}$.

- powers x^α are:
  - convex on $\mathbb{R}_+$ for $\alpha \geq 1$,
  - concave for $0 \leq \alpha \leq 1$,
  - convex for $\alpha \leq 0$.
- $|x|^\alpha$ is convex on all of $\mathbb{R}$ for $\alpha \geq 1$.
- logarithms: $\log x$ is concave on $\mathbb{R}_{++} := \{x \in \mathbb{R} : x > 0\}$.
- the entropy function $-x \log x$ is concave on $\mathbb{R}_{++}$.

Here are some standard examples for functions on $\mathbb{R}^N$:

- affine functions $f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{a} \rangle + b$ are both convex and concave on all of $\mathbb{R}^N$.
- any valid norm $f(\boldsymbol{x}) = \|\boldsymbol{x}\|$ is convex on all of $\mathbb{R}^N$.
- if $f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x})$ are both convex, then the sum $f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ is also convex.

A useful tool for showing that a function $f : \mathbb{R}^N \to \mathbb{R}$ is convex is the fact that f is convex if and only if the function $g_{\boldsymbol{v}} : \mathbb{R} \to \mathbb{R}$,

$$g_{\boldsymbol{v}}(t) = f(\boldsymbol{x} + t\boldsymbol{v}), \quad \operatorname{dom} g = \{t \ : \ \boldsymbol{x} + t\boldsymbol{v} \in \operatorname{dom} f\}$$

is convex for every $\boldsymbol{x} \in \operatorname{dom} f$, $\boldsymbol{v} \in \mathbb{R}^N$.

Example:

Let $f(\boldsymbol{X}) = -\log\det \boldsymbol{X}$ with $\operatorname{dom} f = \mathcal{S}^N_{++}$, where $\mathcal{S}^N_{++}$ denotes the set of symmetric and (strictly) positive definite matrices. For any $\boldsymbol{X} \in \mathcal{S}^N_{++}$, we know that

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{\mathrm{T}},$$

for some diagonal, positive $\boldsymbol{\Lambda}$, so we can define

$$\boldsymbol{X}^{1/2} = \boldsymbol{U}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^{\mathrm{T}}, \quad \text{and} \quad \boldsymbol{X}^{-1/2} = \boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^{\mathrm{T}}.$$

Now consider any symmetric matrix $\boldsymbol{V}$ and t such that $\boldsymbol{X} + t\boldsymbol{V} \in \mathcal{S}^N_{++}$:

$$\begin{aligned} g_{\boldsymbol{V}}(t) &= -\log\det(\boldsymbol{X} + t\boldsymbol{V}) \\ &= -\log\det(\boldsymbol{X}^{1/2}(\mathbf{I} + t\boldsymbol{X}^{-1/2}\boldsymbol{V}\boldsymbol{X}^{-1/2})\boldsymbol{X}^{1/2}) \\ &= -\log\det\boldsymbol{X} - \log\det(\mathbf{I} + t\boldsymbol{X}^{-1/2}\boldsymbol{V}\boldsymbol{X}^{-1/2}) \\ &= -\log\det\boldsymbol{X} - \sum_{n=1}^{N}\log(1+\sigma_i t), \end{aligned}$$

where the σ_i are the eigenvalues of $\boldsymbol{X}^{-1/2}\boldsymbol{V}\boldsymbol{X}^{-1/2}$. The function $-\log(1+\sigma_i t)$ is convex, so the above is a sum of convex functions, which is convex.

Operations that preserve convexity

There are a number of useful operations that we can perform on a convex function while preserving convexity. Some examples include:

- **Positive weighted sum:** A **positive** weighted sum of convex functions is also convex, i.e., if $f_1, \ldots, f_m$ are convex and $w_1, \ldots, w_m \geq 0$, then $w_1 f_1 + \ldots + w_m f_m$ is also convex.
- **Composition with an affine function:** If $f : \mathbb{R}^N \to \mathbb{R}$ is convex, then $g : \mathbb{R}^D \to \mathbb{R}$ defined by

  $$g(\boldsymbol{x}) = f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}),$$

  where $\boldsymbol{A} \in \mathbb{R}^{N\times D}$ and $b \in \mathbb{R}^N$, is convex.

- **Composition with scalar functions:** Consider the function $f(\boldsymbol{x}) = h(g(\boldsymbol{x}))$, where $g : \mathbb{R}^N \to \mathbb{R}$ and $h : \mathbb{R} \to \mathbb{R}$.
  - f is convex if g is convex and h is convex and non-decreasing. Example: $e^{g(\boldsymbol{x})}$ is convex if g is convex.
  - f is convex if g is concave and h is convex and non-increasing.
    Example: $\frac{1}{g(\boldsymbol{x})}$ is convex if g is concave and positive.
- **Max of convex functions:** If f_1 and f_2 are convex, then $f(\boldsymbol{x}) = \max(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}))$ is convex.

First-order conditions for convexity

We say that f is **differentiable** if $\operatorname{dom} f$ is an open set (all of $\mathbb{R}^N$, for example), and the gradient

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_N} \end{bmatrix}$$

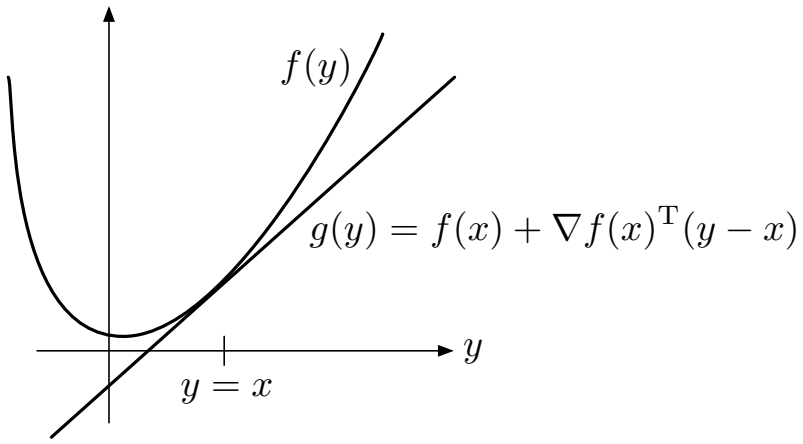
exists for each $\boldsymbol{x} \in \operatorname{dom} f$. The gradient of a function is a core concept in optimization and as such we review a little bit of what it means at the end of these notes.

The following characterization of convexity is an incredibly useful fact, and if we never had to worry about functions that were not differentiable, we might actually just take this as the definition of a convex function.

If f is differentiable, then it is convex if and only if

$$f(\boldsymbol{y}) \;\geq\; f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) \tag{1}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom} f$.



This means that the linear approximation

$$g(\boldsymbol{y}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}),$$

is a **global underestimator** of $f(\boldsymbol{y})$.

It is easy to show that f convex, differentiable $\Rightarrow$ (1). Since f is convex,

$$f(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) \;\leq\; (1-t)f(\boldsymbol{x}) + tf(\boldsymbol{y}), \quad 0 \leq t \leq 1,$$

and so

$$f(\boldsymbol{y}) \;\geq\; f(\boldsymbol{x}) + \frac{f(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) - f(\boldsymbol{x})}{t}, \quad \forall 0 < t \leq 1.$$

Taking the limit as $t \to 0$ on the right yields (1).

It is also true that (1) $\Rightarrow$ f convex. To see this, choose arbitrary $\boldsymbol{x}, \boldsymbol{y}$ and set $\boldsymbol{z}_\theta = (1-\theta)\boldsymbol{x} + \theta\boldsymbol{y}$; then (1) tells us

$$f(\boldsymbol{w}) \;\geq\; f(\boldsymbol{z}_\theta) + \nabla f(\boldsymbol{z}_\theta)^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{z}_\theta).$$

Applying this at $\boldsymbol{w} = \boldsymbol{y}$ and multiplying by θ, then applying it at $\boldsymbol{w} = \boldsymbol{x}$ and multiplying by $(1-\theta)$ yields

$$\begin{aligned}\theta f(\boldsymbol{y}) &\geq \theta f(\boldsymbol{z}_\theta) + \theta\nabla f(\boldsymbol{z}_\theta)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{z}_\theta),\\ (1-\theta) f(\boldsymbol{x}) &\geq (1-\theta) f(\boldsymbol{z}_\theta) + (1-\theta)\nabla f(\boldsymbol{z}_\theta)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{z}_\theta).\end{aligned}$$

Adding these inequalities together establishes the result.

Second-order conditions for convexity

We say that $f : \mathbb{R}^N \to \mathbb{R}$ is twice differentiable if dom f is an open set, and the $N \times N$ Hessian matrix

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_N} \\ \vdots & \ddots & & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_N \partial x_1} & \cdots & & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_N^2} \end{bmatrix}$$

exists for every $\boldsymbol{x} \in$ dom f.

If f is twice differentiable, then it is convex if and only if

$$\nabla^2 f(\boldsymbol{x}) \succeq \mathbf{0} \quad (\text{i.e. } \nabla^2 f(\boldsymbol{x}) \in \mathcal{S}_+^N).$$

for all $\boldsymbol{x} \in$ dom f.

Note that for a one-dimensional function $f : \mathbb{R} \to \mathbb{R}$, the above condition just reduces to $f''(x) \geq 0$. You can prove the one-dimensional version relatively easy (although we will not do so here) using the first-order characterization of convexity described above and the definition of the second derivative. You can then prove the general case by considering the function $g(t) = f(\boldsymbol{x} + t\boldsymbol{v})$. To see how, note that if f is convex and twice differentiable, then so is g. Using the chain rule, we have

$$g''(t) = \boldsymbol{v}^{\mathrm{T}} \nabla^2 f(\boldsymbol{x} + t\boldsymbol{v}) \boldsymbol{v}.$$

Since g is convex, the one-dimensional result above tells us that $g''(0) \geq 0$, and hence $\boldsymbol{v}^{\mathrm{T}} \nabla^2 f(\boldsymbol{x}) \boldsymbol{v} \geq 0$. Since this has to hold for any $\boldsymbol{v}$, this means that $\nabla^2 f(\boldsymbol{x}) \succeq \boldsymbol{0}$. The proof that $\nabla^2 f(\boldsymbol{x}) \succeq \boldsymbol{0}$ implies convexity follows a similar strategy.

In addition, it is strictly convex if an only if

$$\nabla^2 f(\boldsymbol{x}) \succ \boldsymbol{0} \quad (\text{i.e. } \nabla^2 f(\boldsymbol{x}) \in S^N_{++}).$$

Standard examples (from [BV04])

Quadratic functionals:

$$f(\boldsymbol{x}) = \frac{1}{2} \boldsymbol{x}^{\mathrm{T}} \boldsymbol{P} \boldsymbol{x} + \boldsymbol{q}^{\mathrm{T}} \boldsymbol{x} + r,$$

where $\boldsymbol{P}$ is symmetric, has

$$\nabla f(\boldsymbol{x}) = \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}, \quad \nabla^2 f(\boldsymbol{x}) = \boldsymbol{P},$$

so $f(\boldsymbol{x})$ is convex iff $\boldsymbol{P} \succeq \boldsymbol{0}$.

Least-squares:

$$f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2,$$

where $\boldsymbol{A}$ is an arbitrary $M \times N$ matrix, has

$$\nabla f(\boldsymbol{x}) = 2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}), \quad \nabla^2 f(\boldsymbol{x}) = 2\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A},$$

and is convex for any $\boldsymbol{A}$.

Quadratic-over-linear:
In $\mathbb{R}^2$, if

$$f(\boldsymbol{x}) = x_1^2/x_2,$$

then

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2x_1/x_2 \\ -x_1^2/x_2^2 \end{bmatrix}, \quad \nabla^2 f(\boldsymbol{x}) = \frac{2}{x_2^3}\begin{bmatrix} x_2^2 & -x_1x_2 \\ -x_1x_2 & x_1 \end{bmatrix} = \frac{2}{x_2^3}\begin{bmatrix} x_2 \\ -x_1 \end{bmatrix}\begin{bmatrix} x_2 & -x_1 \end{bmatrix},$$

and so f is convex on $\mathbb{R} \times [0, \infty]$ ($x_1 \in \mathbb{R}, x_2 \geq 0$).

Strong convexity and smoothness

We say that a function f is **strongly convex** if there is a $\mu > 0$ such that

$$f(\boldsymbol{x}) - \frac{\mu}{2}\|\boldsymbol{x}\|_2^2 \quad \text{is convex.} \tag{2}$$

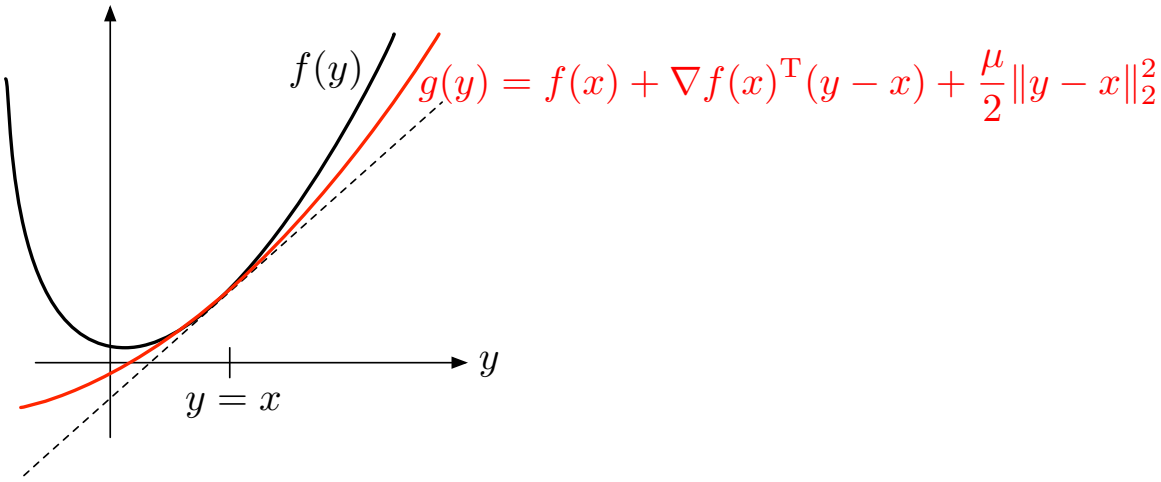
We call μ the *strong convexity parameter* and will sometimes say that f is μ-strongly convex. In a sense, what we are saying is that f is so convex that we can subtract off a quadratic function and still preserve convexity.

If f is differentiable, there is another interpretation of strong convexity. We have seen that an equivalent definition of regular convexity

is that the linear approximation formed using the gradient at a point $\boldsymbol{x}$ is a global underestimator of the function — see (1) and the picture below. If f obeys (2), then we can form a *quadratic* global underestimator as

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\mu}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2. \tag{3}$$

Here is a picture



We will show that (2) implies (3) in a future homework.

If f is twice differentiable, there is yet another interpretation of strong convexity. If f obeys (2) then we know that the Hessian of $f(\boldsymbol{x}) - \frac{\mu}{2}\|\boldsymbol{x}\|_2^2$ does not have any negative eigenvalues, i.e.

$$\nabla^2 \left(f(\boldsymbol{x}) - \frac{\mu}{2}\|\boldsymbol{x}\|_2^2 \right) \succeq \mathbf{0}.$$

Thus (since $\nabla^2(\|\boldsymbol{x}\|_2^2) = 2\mathbf{I}$),

$$\begin{gathered} \nabla^2 f(\boldsymbol{x}) - \mu\mathbf{I} \succeq \mathbf{0}, \\ \Downarrow \\ \nabla^2 f(\boldsymbol{x}) \succeq \mu\mathbf{I}. \end{gathered}$$

This is just a fancy way of saying that the smallest eigenvalue of the Hessian $\nabla^2 f(\boldsymbol{x})$ is uniformly bounded below by μ for all $\boldsymbol{x}$.

In addition to convexity, there is one more type of structure that we consider for functions $f : \mathbb{R}^N \to \mathbb{R}$. We say that differentiable f has a **Lipschitz gradient** if there is a L such that

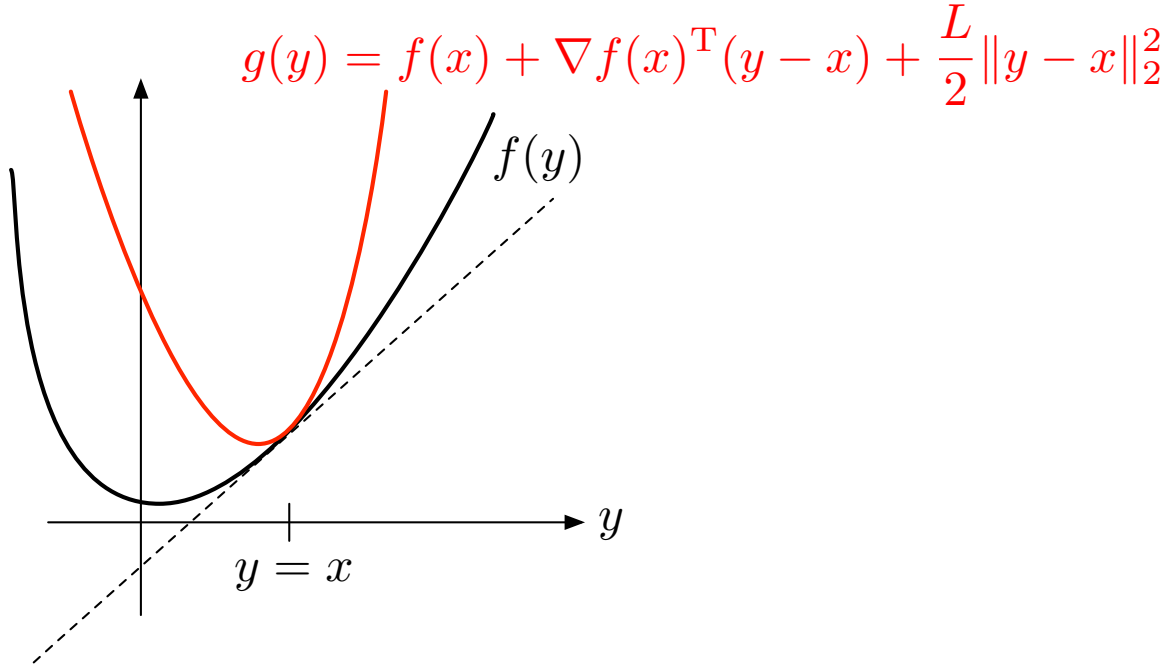
$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \;\leq\; L\|\boldsymbol{x} - \boldsymbol{y}\|_2, \quad \text{for all} \;\; \boldsymbol{x}, \boldsymbol{y}. \tag{4}$$

This means that the gradient $\nabla f(\boldsymbol{x})$ does not change radically as we change $\boldsymbol{x}$. Functions f that obey (4) are also referred to as **L-smooth**. This definition applies whether or not the function f is convex.

Whether or not f is convex, if it is L-smooth, it there is a natural quadtratic *over*estimator. Around any point $\boldsymbol{x}$, we have the upper bound

$$f(\boldsymbol{y}) \;\leq\; f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2. \tag{5}$$

Here is a picture



We will show that (4) implies (5) in a future homework.

If f is twice differentiable, then there is another way to interpret L-smoothness. If f obeys (4), then we have a uniform upper bound on the *largest* eigenvalue of the Hessian at every point:

$$\nabla^2 f(\boldsymbol{x}) \preceq L\mathbf{I}, \quad \text{for all } \boldsymbol{x}. \tag{6}$$

This makes intuitive sense, as (4) tells us that the first derivative cannot change too quickly, so there must be some kind of bound on the second derivative. We will establish that (4) implies (6) (again, regardless of whether f is convex) in a future homework.

Review: The gradient

First, recall that a function $f : \mathbb{R} \to \mathbb{R}$ is differentiable if its derivative, defined as

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta},$$

exists for all $x \in \operatorname{dom} f$. To extend this notion to functions of multiple variables, we must first extend our notion of a derivative. For a function $f : \mathbb{R}^N \to \mathbb{R}$ that is defined on N-dimensional vectors, recall that the **partial derivative** with respect to x_n is

$$\frac{\partial f(\boldsymbol{x})}{\partial x_n} = \lim_{\delta \to 0} \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_n) - f(\boldsymbol{x})}{\delta},$$

where $\boldsymbol{e}_n$ is the n^{th} "standard basis element", i.e., the vector of all zeros with a single 1 in the n^{th} entry.

The **gradient** of a function $f : \mathbb{R}^N \to \mathbb{R}$ is the vector of partial derivatives given by:

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_N} \end{bmatrix}.$$

Similar to the scalar case, we say that f is differentiable if the gradient exists for each $\boldsymbol{x} \in \operatorname{dom} f$.

We will use the term gradient in two subtly different ways. Sometimes we use $\nabla f(\boldsymbol{x})$ to describe a *vector-valued function* or a *vector field*,

i.e., a function that takes an arbitrary $\boldsymbol{x} \in \mathbb{R}^N$ and produces another vector. When referring to this vector-valued function, we sometimes use the words **gradient map**, but sometimes we will overload the term "gradient"; we will use the notation $\nabla f(\boldsymbol{x})$ to refer to the vector given by the gradient map evaluated at a particular point $\boldsymbol{x}$. So sometimes when we say "gradient" we mean a vector-valued function, and sometimes we mean a single vector, and in both cases we use the notation $\nabla f(\boldsymbol{x})$. Which one will usually be obvious by the context.[1]

Note that in some cases we will use the notation $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ to indicate that we are taking the gradient with respect to $\boldsymbol{x}$. This can be helpful when f is a function of more variables than just $\boldsymbol{x}$, but most of the time this is not necessary so we will typically use the simpler $\nabla f(\boldsymbol{x})$.

Here we adopt the convention that the gradient is a *column vector*. This is the most common choice and is most convenient in this class, but some texts will instead treat the gradient as a row vector. The reason for this is to align with the standard convention for the *Jacobian*.[2] Thus, it is always worth double-checking what notation is being used when consulting outside resources.

[1] This is just like in the scalar case, where the notation $f(x)$ can sometimes refer to the function f and sometimes the function evaluated at x.

[2] The Jacobian of a vector-valued function $f : \mathbb{R}^N \to \mathbb{R}^M$ is the $M \times N$ matrix of partial derivatives with respect to each dimension in the range. In this course we will mostly be concerned with functions mapping to a single dimension, in which case the Jacobian would be the $1 \times N$ matrix $\nabla^{\mathrm{T}} f(\boldsymbol{x})$, i.e., the gradient but treated as a row vector. Directly defining the gradient as a row vector instead of a column vector is thus more convenient in some contexts.

Interpretation of the gradient

The gradient is one of the most fundamental concepts of this course. We can interpret the gradient in many ways. One way to think of the gradient when evaluated at a particular point $\boldsymbol{x}$ is that it defines a linear mapping from $\mathbb{R}^N$ to $\mathbb{R}$. Specifically, given a $\boldsymbol{u} \in \mathbb{R}^N$, we can use $\nabla f(\boldsymbol{x})$ to define a mapping of $\boldsymbol{u}$ to $\mathbb{R}$ by simply taking the inner product between the two vectors:

$$\langle \boldsymbol{u}, \nabla f(\boldsymbol{x}) \rangle.$$

What does this mapping tell us? It computes the **directional derivative** of f in the direction of $\boldsymbol{u}$, i.e.,

$$\langle \boldsymbol{u}, \nabla f(\boldsymbol{x})) \rangle = \lim_{\delta \to 0} \frac{f(\boldsymbol{x} + \delta \boldsymbol{u}) - f(\boldsymbol{x})}{\delta}. \tag{7}$$

This tells us how fast f is changing at $\boldsymbol{x}$ when we move in the direction of $\boldsymbol{u}$.

This fundamental fact is a direct consequence of Taylor's theorem (see the Technical Details section below). Specifically, let $f : \mathbb{R}^N \to \mathbb{R}$ be any differentiable function. Then for any $\boldsymbol{u} \in \mathbb{R}^N$, we can write

$$f(\boldsymbol{x} + \boldsymbol{u}) = f(\boldsymbol{x}) + \langle \boldsymbol{u}, \nabla f(\boldsymbol{x}) \rangle + h(\boldsymbol{u}) \|\boldsymbol{u}\|_2,$$

where $h(\boldsymbol{u}) : \mathbb{R}^N \to \mathbb{R}$ is some function satisfying $h(\boldsymbol{u}) \to 0$ as $\boldsymbol{u} \to \boldsymbol{0}$.

If we substitute $\delta \boldsymbol{u}$ in place of $\boldsymbol{u}$ above and rearrange, we obtain the identity

$$\begin{aligned} \langle \boldsymbol{u}, \nabla f(\boldsymbol{x}) \rangle &= \frac{f(\boldsymbol{x} + \delta \boldsymbol{u}) - f(\boldsymbol{x}) - h(\delta \boldsymbol{u}) \|\delta \boldsymbol{u}\|_2}{\delta} \\ &= \frac{f(\boldsymbol{x} + \delta \boldsymbol{u}) - f(\boldsymbol{x})}{\delta} - h(\delta \boldsymbol{u}) \|\boldsymbol{u}\|_2. \end{aligned}$$

Note that this holds for any $\delta > 0$. Since $h(\delta \boldsymbol{u}) \to 0$ as $\delta \to 0$, we can arrive at (7) by simply taking the limit as $\delta \to 0$.

A related way to think of $\nabla f(\boldsymbol{x})$ is as a vector that is pointing in the direction of *steepest ascent*, i.e., the direction in which f increases the fastest when starting at $\boldsymbol{x}$. To justify this, note that we just observed that we can interpret $\langle \boldsymbol{u}, \nabla f(\boldsymbol{x}) \rangle$ as measuring how quickly f increases when we move in the direction of $\boldsymbol{u}$. How can we find the direction $\boldsymbol{u}$ that maximizes this quantity? You may recall that the Cauchy-Schwarz inequality tells us that

$$|\langle \boldsymbol{u}, \nabla f(\boldsymbol{x}) \rangle| \leq \|\nabla f(\boldsymbol{x})\|_2 \|\boldsymbol{u}\|_2,$$

and that this holds with equality when $\boldsymbol{u}$ is co-linear with $\nabla f(\boldsymbol{x})$, i.e., when $\boldsymbol{u}$ points in the same direction as $\nabla f(\boldsymbol{x})$. Specifically, this implies that $\nabla f(\boldsymbol{x})$ is the direction of steepest *ascent*, and $-\nabla f(\boldsymbol{x})$ is the direction of steepest *descent*.

More broadly, this characterizes the entire sets of ascent/descent directions. Suppose that $f : \mathbb{R}^N \to \mathbb{R}$ is differentiable at $\boldsymbol{x}$. If $\boldsymbol{u} \in \mathbb{R}^N$ is a vector obeying $\langle \boldsymbol{u}, \nabla f(\boldsymbol{x}) \rangle < 0$, then we say that $\boldsymbol{u}$ is a **descent direction** from $\boldsymbol{x}$, as we can find a $t > 0$ small enough so that

$$f(\boldsymbol{x} + t\boldsymbol{u}) < f(\boldsymbol{x}).$$

Similarly, if $\langle \boldsymbol{u}, \nabla f(\boldsymbol{x}) \rangle > 0$, then we say that $\boldsymbol{u}$ is an **ascent direction** from $\boldsymbol{x}$, as again for $t > 0$ small enough,

$$f(\boldsymbol{x} + t\boldsymbol{u}) > f(\boldsymbol{x}).$$

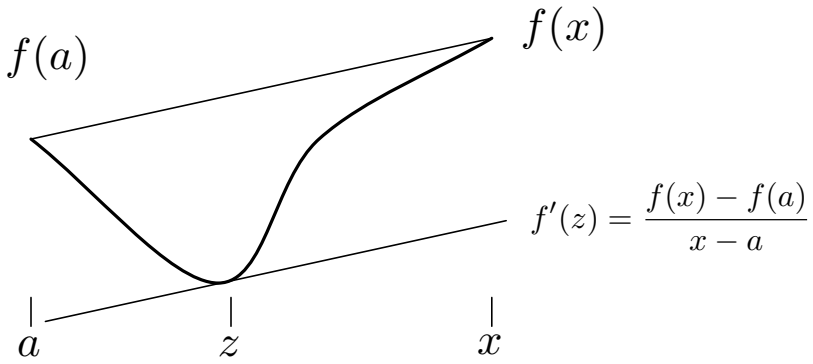
It should hopefully not be a huge stretch of the imagination to see that being able to compute the direction of steepest ascent (or steepest descent) will be useful in the context of finding a maximum/minimum of a function.

Technical Details: Taylor's Theorem

You might recall the mean-value theorem from your first calculus class. If $f : \mathbb{R} \to \mathbb{R}$ is a differentiable function on the interval $[a, x]$, then there is a point inside this interval where the derivative of f matches the line drawn between $f(a)$ and $f(x)$. More precisely, there exists a $z \in [a, x]$ such that

$$f'(z) = \frac{f(x) - f(a)}{x - a}.$$

Here is a picture:



We can re-arrange the expression above to say that there is some z between a and x such that

$$f(x) = f(a) + f'(z)(x - a).$$

The mean-value theorem extends to derivatives of higher order; in this case it is known as *Taylor's theorem*. For example, suppose that f is twice differentiable on $[a, x]$, and that the first derivative f' is continuous. Then there exists a z between a and x such that

$$f(x) = f(a) + f'(a)(x - a) + \frac{f''(z)}{2}(x - a)^2.$$

In general, if f is $k+1$ times differentiable, and the first k derivatives are continuous, then there is a point z between a and x such that

$$f(x) = p_{k,a}(x) + \frac{f^{(k+1)}(z)}{k!}(x-a)^{k+1},$$

where $p_{k,a}(x)$ polynomial formed from the first k terms of the Taylor series expansion around a:

$$p_{k,a}(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k.$$

These results give us a way to quantify the accuracy of the Taylor approximation around a point. For example, if f is twice differentiable with f' continuous, then

$$f(x) = f(a) + f'(a)(x-a) + h_1(x)(x-a),$$

for a function $h_1(x)$ goes to zero as x goes to a:

$$\lim_{x \to a} h_1(x) = 0.$$

In fact, you do not even need two derivatives for this to be true. If f has a single derivative, then we can find such an h_1. When f has two derivatives, then we have an explicit form for h_1:

$$h_1(x) = \frac{f''(z_x)}{2}(x-a),$$

where z_x is the point returned by the (generalization of) the mean value theorem for a given x.

In general, if f has k derivatives, then there exists an $h_k(x)$ with $\lim_{x \to a} h_k(x) = 0$ such that

$$f(x) = p_{k,a}(x) + h_k(x)(x-a)^k.$$

All of the results above extend to functions of multiple variables. For example, if $f(\boldsymbol{x}) : \mathbb{R}^N \to \mathbb{R}$ is differentiable, then around any point $\boldsymbol{a}$,

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + \langle \boldsymbol{x} - \boldsymbol{a}, \nabla f(\boldsymbol{a}) \rangle + h_1(\boldsymbol{x}) \|\boldsymbol{x} - \boldsymbol{a}\|_2,$$

where $h_1(\boldsymbol{x}) \to 0$ as $\boldsymbol{x}$ approaches $\boldsymbol{a}$ from any direction. If $f(\boldsymbol{x})$ is twice differentiable and the first derivative is continuous, then there exists $\boldsymbol{z}$ on the line between $\boldsymbol{a}$ and $\boldsymbol{x}$ such that

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + \langle \boldsymbol{x} - \boldsymbol{a}, \nabla f(\boldsymbol{a}) \rangle + \frac{1}{2} (\boldsymbol{x} - \boldsymbol{a})^{\mathrm{T}} \nabla^2 f(\boldsymbol{z}) (\boldsymbol{x} - \boldsymbol{a}).$$

We will use these two particular multidimensional results in this course, referring to them generically as "Taylor's theorem".

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

II. Unconstrained Optimization

Unconstrained minimization of convex functions

We will start our discussion about solving convex optimization programs by considering the unconstrained case. Our template problem is

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}), \tag{1}$$

where f is convex. While we state this problem as a search over all of $\mathbb{R}^N$, almost everything we say here can be applied to minimized a convex function over an *open* set[1].

In these notes, we discuss two fundamental results. First, we will show that when f is convex, all local minimizers of (1) must also be global minimizers. Second, under the conditions that $f(\boldsymbol{x})$ is convex and differentiable, we will show that $\boldsymbol{x}^\star$ is a minimizer of (1) if and only if the derivative is equal to zero:

$$\boldsymbol{x}^\star \text{ is a global minimizer} \quad \Leftrightarrow \quad \nabla f(\boldsymbol{x}^\star) = \mathbf{0}.$$

Finally, we will touch briefly on conditions for the existence and uniqueness of solutions to (1) when f is convex, strictly convex, and strongly convex.

Throughout this section of the notes, we will assume that f is differentiable. Similar statements to the gradient condition above are also true for non-differentiable (but still convex) f; we will discuss these in detail at the end of this chapter.

[1] Intuition: Open sets don't have boundaries, closed sets do. The entire point of the study of constrained optimization, which we will get to next, comes down to treating the fact that the solution can (and probably is) on the boundary of your constraint set.

Local minima are also global minima

The most important property of convex functions from an optimization perspective is that any local minimum is also a global minimum, or more formally:

> Let $f(\boldsymbol{x})$ be a convex function on $\mathbb{R}^N$, and suppose that $\boldsymbol{x}^\star$ is a local minimizer of f in that there exists an $\epsilon > 0$ such that
>
> $$f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x}) \quad \text{for all} \quad \|\boldsymbol{x} - \boldsymbol{x}^\star\|_2 \leq \epsilon.$$
>
> Then $\boldsymbol{x}^\star$ is also a global minimizer: $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^N$.

To prove this, suppose that $\boldsymbol{x}^\star$ is a local minimum. We want to show that $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x}')$ for any $\boldsymbol{x}'$. We already have that $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x}')$ if $\|\boldsymbol{x}' - \boldsymbol{x}^\star\|_2 \leq \epsilon$, so all we need to do is show that this also holds for $\boldsymbol{x}'$ with $\|\boldsymbol{x}' - \boldsymbol{x}^\star\|_2 > \epsilon$. Note that from convexity, we have

$$f(\theta \boldsymbol{x}' + (1-\theta)\boldsymbol{x}^\star) \leq \theta f(\boldsymbol{x}') + (1-\theta) f(\boldsymbol{x}^\star)$$

for any $\theta \in [0,1]$. In particular, the above holds for $\theta = \epsilon / \|\boldsymbol{x}' - \boldsymbol{x}^\star\|_2$ (which is less than 1 since $\|\boldsymbol{x}' - \boldsymbol{x}^\star\|_2 > \epsilon$). For this choice of θ we have

$$\|\theta \boldsymbol{x}' + (1-\theta)\boldsymbol{x}^\star - \boldsymbol{x}^\star\|_2 = \theta \|\boldsymbol{x}' - \boldsymbol{x}^\star\|_2 = \epsilon,$$

thus $\theta \boldsymbol{x}' + (1-\theta)\boldsymbol{x}^\star$ lives in the neighborhood where $\boldsymbol{x}^\star$ is a local minimum, and hence

$$f(\boldsymbol{x}^\star) \leq f(\theta \boldsymbol{x}' + (1-\theta)\boldsymbol{x}^\star).$$

Combining this with the inequality above we have

$$f(\boldsymbol{x}^\star) \leq \theta f(\boldsymbol{x}') + (1-\theta) f(\boldsymbol{x}^\star).$$

Rearranging this gives us $\theta f(\boldsymbol{x}^\star) \leq \theta f(\boldsymbol{x}')$, or simply $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x}')$, which is exactly what we wanted to prove.

Note that for functions f that are *not* convex, any number of things are possible. It *might* be the case that there is only one local minimum and that it corresponds to the global minimum. We are typically not so lucky, though. There may be many local minima, some of which may be very far from actually minimizing f.

We close this section by re-emphasizing that the entire discussion above would stay the same if we replaced $\text{minimize}_{\boldsymbol{x}\in\mathbb{R}^N} f(\boldsymbol{x})$ with $\text{minimize}_{\boldsymbol{x}\in\mathcal{U}} f(\boldsymbol{x})$ for any open set $\mathcal{U} \subset \mathbb{R}^N$.

Optimality conditions for differentiable functions

We have just shown that if we want to find a global minimum of a convex function, it is sufficient to find any local minimum. This raises the question: How do we know when we have found a minimum of a function (local or global)? Here we provide an answer to this question in the case where f is differentiable.

Let f be convex and differentiable on $\mathbb{R}^N$. Then $\boldsymbol{x}^\star$ solves

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x})$$

if and only if $\nabla f(\boldsymbol{x}^\star) = \mathbf{0}$.

To prove this, we first assume that $\boldsymbol{x}^\star$ is a local minimum of f and show that this implies that $\nabla f(\boldsymbol{x}^\star) = \mathbf{0}$. This follows almost immediately. If $\boldsymbol{x}^\star$ is a local minimum of f, then this means that *every*

direction must be an ascent direction, i.e., $\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}) \rangle \geq 0$ for all $\boldsymbol{d} \in \mathbb{R}^N$. However, the only way we can make $\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0$ for all $\boldsymbol{d}$ is if $\nabla f(\boldsymbol{x}^\star) = \boldsymbol{0}$. Thus, for differentiable f

$$\boldsymbol{x}^\star \text{ is a (local or global) minimizer} \quad \Rightarrow \quad \nabla f(\boldsymbol{x}^\star) = \boldsymbol{0}.$$

Note that this fact does not actually require f to be convex.

Now we will show that for convex f we also have that $\nabla f(\boldsymbol{x}^\star) = \boldsymbol{0}$ implies that f is a minimizer. Recall our first-order characterization of convexity

$$f(\boldsymbol{x}^\star + \boldsymbol{u}) \geq f(\boldsymbol{x}^\star) + \langle \boldsymbol{u}, \nabla f(\boldsymbol{x}^\star) \rangle,$$

for all choices of $\boldsymbol{u} \in \mathbb{R}^N$. This now makes it clear that for convex f

$$\nabla f(\boldsymbol{x}^\star) = \boldsymbol{0} \quad \Rightarrow \quad \boldsymbol{x}^\star \text{ is a (global) minimizer.}$$

This fact will lie at the heart of the algorithms for unconstrained convex optimization that we will begin discussing in the next set of notes — if we can find an $\boldsymbol{x}$ that makes the gradient vanish, then we have solved the problem.

Existence and uniqueness

We close this set of notes with some discussion about when solutions to the unconstrained minimization problem (1) exist and are unique.

To begin, it is important to realize that it is not always the case that a convex function will actually have a minimizer. That is, there may be sometimes be no $\boldsymbol{x}^\star$ such that $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^N$. For

example, $f(x) = e^{-x}$ does not have a minimizer for $x \in \mathbb{R}$, even though it is convex (and differentiable). A more dramatic example is taking f to be affine in $\mathbb{R}^N$, $f(\boldsymbol{x}) = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b$. It is clear that although this function is convex, it is unbounded below on all of $\mathbb{R}^N$, and so does not have a minimizer.

We will cover some basis existence and uniqueness results for three different types of functions: convex, strictly convex, and strongly convex.

Convex

We begin with a simple fact that I might ask you to prove on the homework next week. Suppose that $f(\boldsymbol{x})$ is convex and is well-defined[2] on all of $\mathbb{R}^N$, that is $|f(\boldsymbol{x})| < \infty$ for all $\boldsymbol{x} \in \mathbb{R}^N$. Then $f(\boldsymbol{x})$ is continuous on $\mathbb{R}^N$. This result is actually true for general open sets: if $f(\boldsymbol{x})$ is convex on $\mathcal{U}$, then $f(\boldsymbol{x})$ is continuous at every point in $\mathcal{U}$.

As a direct result of this continuity, we know that $f(\boldsymbol{x})$ has a minimizer over any compact subset of $\mathbb{R}^N$. Recall again the Weierstrass extreme value theorem: if $f(\boldsymbol{x})$ is a continuous function on a compact set $\mathcal{K} \subset \mathbb{R}^N$, then it attains its minimum (and maximum) value in at least one point. That is,

$$\underset{\boldsymbol{x} \in \mathcal{K}}{\text{minimize}}\, f(\boldsymbol{x})$$

has a minimizer on $\mathcal{K}$ — there exists a $\boldsymbol{x}^\star \in \mathcal{K}$ such that $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{K}$.

[2]Note that we are not asking that f is bounded, simply that it does not "jump" to ∞ anywhere.

Here we are interested in the unconstrained setting, and we have already seen examples of convex f which do not have a minimizer in this setting. There is, however, a class of functions for which we can guarantee a global minimizer in the unconstrained setting. If the sublevel sets of f,

$$\mathcal{S}(f, \beta) = \{\boldsymbol{x} \in \mathbb{R}^N \ : \ f(\boldsymbol{x}) \leq b\}$$

are compact (again, this means closed and bounded), then there will be at least one global minimizer. This should be easy to see — just choose β such that $\mathcal{S}(f, \beta)$ is non-empty, then

$$\underset{\boldsymbol{x} \in \mathcal{S}(f,\beta)}{\text{minimize}} \ f(\boldsymbol{x})$$

has a minimizer (by the extreme value theorem), and this also clearly corresponds to a minimizer of f over $\mathbb{R}^N$. If f is continuous (which all convex functions with $\operatorname{dom} f = \mathbb{R}^N$ are), then having compact sublevel sets is the same as being *coercive*: for every sequence $\{\boldsymbol{x}_k\} \subset \mathbb{R}^N$ with $\|\boldsymbol{x}_k\|_2 \to \infty$, we have $f(\boldsymbol{x}_k) \to \infty$ as well. (I will let you prove that at home.)

To summarize:

> If $f(\boldsymbol{x})$ is convex and has compact sublevel sets, then it has at least one minimizer on $\mathbb{R}^N$.

Strictly convex

In general, there can be multiple minimizers for a convex function. However, there are certainly lots of scenarios where there is only one unique minimizer. One prominent example is when f is *strictly* convex.

Let f be strictly convex on $\mathbb{R}^N$. If f has a global minimizer, then it is unique.

This is easy to argue by contradiction. Let $\boldsymbol{x}^\star$ be a global minimizer, and suppose that there existed an $\boldsymbol{x}' \neq \boldsymbol{x}^\star$ with $f(\boldsymbol{x}') = f(\boldsymbol{x}^\star)$. But then there would be many $\boldsymbol{x}$ which achieve smaller values, as for all $0 < \theta < 1$,

$$\begin{aligned} f(\theta \boldsymbol{x}^\star + (1-\theta)\boldsymbol{x}') &< \theta f(\boldsymbol{x}^\star) + (1-\theta) f(\boldsymbol{x}') \\ &= f(\boldsymbol{x}^\star). \end{aligned}$$

This would contradict the assertion that $\boldsymbol{x}^\star$ is a global minimizer, and hence no such $\boldsymbol{x}'$ can exist.

Note that it is not necessary for a minimizer of a strictly convex function to exist. Our previous example of $f(x) = e^{-x}$ for $x \in \mathbb{R}$ falls into this category.

Strongly convex

On the other hand, strongly convex functions always have unique minimizers. Let's start by considering special case of a unconstrained quadratic program:

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{x}) := \frac{1}{2}\boldsymbol{x}^{\mathrm{T}} \boldsymbol{P} \boldsymbol{x} + \boldsymbol{q}^{\mathrm{T}} \boldsymbol{x} + r,$$

where $\boldsymbol{P}$ is a symmetric positive definite matrix. We know that in this case, f has the same Hessian everywhere, $\nabla^2 f(\boldsymbol{x}) = \boldsymbol{P}$, and so f is μ-strongly convex, where μ is the smallest eigenvalue of $\boldsymbol{P}$. We also know that f has a unique minimizer: the gradient is zero when

$\boldsymbol{P}\boldsymbol{x}^\star = -\boldsymbol{q}$, and since $\boldsymbol{P}$ is invertible, there is exactly one $\boldsymbol{x}^\star$ that obeys this condition. We also know that the sublevel sets of such a quadratic function are ellipsoids and hence are compact.

To extend to a general strongly convex f, recall that for any $\boldsymbol{x}_0$, strongly convex functions obey

$$f(\boldsymbol{x}) \;\geq\; \underbrace{f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_0) + \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{x}_0\|_2^2}_{q(\boldsymbol{x})}. \tag{2}$$

That is, f can be *lower bounded* by a (strongly convex) quadratic function $q(\boldsymbol{x})$. We know that the sublevel sets of $q(\boldsymbol{x})$ are compact, and the inequality tells us that the sublevel sets of f will be subsets of the corresponding sets for q:

$$\mathcal{S}(f, \beta) \subseteq \mathcal{S}(q, \beta).$$

As f is continuous, $\mathcal{S}(f, \beta)$ is closed, and the boundedness of $\mathcal{S}(f, \beta)$ follows from the boundedness of $\mathcal{S}(q, \beta)$. Thus $\mathcal{S}(f, \beta)$ is compact for all β, and $f(\boldsymbol{x})$ has at least one minimizer.

Let $\boldsymbol{x}^\star$ be one such minimizer. It is now an easy argument to show that $\boldsymbol{x}^\star$ must be the unique minimizer. Since $\nabla f(\boldsymbol{x}^\star) = \boldsymbol{0}$, we can use $\boldsymbol{x}_0 = \boldsymbol{x}^\star$ in (2) to get

$$f(\boldsymbol{x}) - f(\boldsymbol{x}^\star) \;\geq\; \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{x}^\star\|_2^2.$$

Thus $f(\boldsymbol{x}) = f(\boldsymbol{x}^\star)$ only when $\boldsymbol{x} = \boldsymbol{x}^\star$.

To summarize:

If f is a strongly convex function on $\mathbb{R}^N$, then it has a unique global minimizer.

Algorithms for unconstrained minimization

One of the benefits of minimizing convex functions is that we can often use very simple algorithms to find solutions. Specifically, we want to solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}),$$

where f is convex. For now we will assume that f is also differentiable.[1] We have just seen that, in this case, a necessary and sufficient condition for $\boldsymbol{x}^\star$ to be a minimizer is that the gradient vanishes:

$$\nabla f(\boldsymbol{x}^\star) = \mathbf{0}.$$

Thus, we can equivalently think of the problem of minimizing $f(\boldsymbol{x})$ as finding an $\boldsymbol{x}^\star$ that $\nabla f(\boldsymbol{x}^\star) = \mathbf{0}$. As noted before, it is not a given that such an $\boldsymbol{x}^\star$ exists, but for now we will assume that f does have (at least one) minimizer.

Many general-purpose optimization algorithms are iterative, and have the following basic form:

Iterative descent

Initialize: $k = 0$, $\boldsymbol{x}_0 =$ initial guess
while not converged **do**
  calculate a direction to move $\boldsymbol{d}_k$
  calculate a step size $\alpha_k \geq 0$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \, \boldsymbol{d}_k$
  $k = k + 1$
end while

[1] We will cover the case where f is not differentiable a little later in the notes.

The central challenge in designing a good algorithm mostly boils down to computing the direction $\boldsymbol{d}_k$. As a preview, here are some choices that we will discuss:

1. **Gradient descent**: We take

$$\boldsymbol{d}_k = -\nabla f\left(\boldsymbol{x}_k\right).$$

   This is the direction of "steepest descent" (where "steepest" is defined relative to the Euclidean norm). Gradient descent iterations are cheap, but many iterations may be required for convergence.

2. **Accelerated gradient descent**: We can sometimes reduce the number of iterations required by gradient descent by incorporating a *momentum* term. Specifically, we first compute

$$\boldsymbol{p}_k = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}$$

   and then take

$$\boldsymbol{d}_k = -\nabla f\left(\boldsymbol{x}_k\right) + \frac{\beta_k}{\alpha_k}\boldsymbol{p}_k$$

   or

$$\boldsymbol{d}_k = -\nabla f\left(\boldsymbol{x}_k + \beta_k \boldsymbol{p}_k\right) + \frac{\beta_k}{\alpha_k}\boldsymbol{p}_k.$$

   The "heavy ball" method and conjugate gradient descent use the former update rule; Nesterov's method uses the latter. We will see later that by incorporating this momentum term, we can sometimes dramatically reduce the number of iterations required for convergence.

3. **Newton's method**: Gradient descent methods are based on building linear approximations to the function at each iteration. We can also build a quadratic model around $\boldsymbol{x}_k$ then compute

the exact minimizer of this quadratic by solving a system of equations. This corresponds to taking

$$\boldsymbol{d}_k = -\left(\nabla^2 f(\boldsymbol{x}_k)\right)^{-1} \nabla f\left(\boldsymbol{x}_k\right),$$

that is, the inverse of the Hessian evaluated at $\boldsymbol{x}_k$ applied to the gradient evaluated at the same point. Newton iterations tend to be expensive (as they require a system solve), but they typically converge in far fewer iterations than gradient descent.

4. **Quasi-Newton methods**: If the dimension N of $\boldsymbol{x}$ is large, Newton's method is not computationally feasible. In this case we can replace the Newton iteration with

$$\boldsymbol{d}_k = -\boldsymbol{Q}_k \nabla f\left(\boldsymbol{x}_k\right)$$

where $\boldsymbol{Q}_k$ is an approximation or estimate of $\left(\nabla^2 f(\boldsymbol{x}_k)\right)^{-1}$. Quasi-Newton methods may require more iterations than a pure Newton approach, but can still be very effective.

Whichever direction we choose, it should be a **descent direction**, i.e., $\boldsymbol{d}_k$ should satisfy

$$\left\langle \boldsymbol{d}_k, \nabla f\left(\boldsymbol{x}_k\right)\right\rangle \leq 0.$$

Since f is convex, it is always true that

$$f\left(\boldsymbol{x} + \alpha \boldsymbol{d}\right) \geq f(\boldsymbol{x}) + \alpha \left\langle \boldsymbol{d}, \nabla f(\boldsymbol{x})\right\rangle,$$

and so to decrease the value of the function while moving in direction $\boldsymbol{d}$, it is necessary that the inner product above be negative.

Line search methods

Given a starting point $\boldsymbol{x}_k$ and a direction $\boldsymbol{d}_k$, we still need to decide on α_k, i.e., how far to move. With $\boldsymbol{x}_k$ and $\boldsymbol{d}_k$ fixed, we can think of the remaining problem as a one-dimensional optimization problem where we would like to choose α to minimize (or at least reduce)

$$\phi(\alpha) = f\left(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k\right).$$

Note that we don't necessarily need to find the true minimum – we aren't even sure that we are moving in the right direction at this point – but we would generally still like to make as much progress as possible before calculating a new direction $\boldsymbol{d}_{k+1}$. There are many methods for doing this, here are three:

Fixed step size

We can just use a constant step size $\alpha_k = \alpha$. This will work if the step size is small enough, but usually this results in using more iterations than necessary. This is actually a very commonly used approach since if your problem is small enough, this may not matter.

Exact line search

Another approach is to solve the one-dimensional optimization program

$$\underset{\alpha \geq 0}{\text{minimize}} \ \phi(\alpha).$$

There are a variety of strategies you could take here (e.g., applying a bisection search or some similar one-dimensional optimization strategy) to try to solve this problem. This is typically not worth

the trouble. However, there are certain instances (e.g., least squares and other unconstrained convex quadratic programs) when it can be solved analytically, in which case it is generally a good idea.

Example: Minimizing a quadratic function Suppose we wish to solve the optimization problem

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}.$$

For example, this optimization problem arises in the context of solving least squares problems. Suppose that we have selected a step direction $\boldsymbol{d}_k$. In this case

$$\phi(\alpha) = \frac{1}{2}(\boldsymbol{x}_k + \alpha\boldsymbol{d}_k)^{\mathrm{T}}\boldsymbol{Q}(\boldsymbol{x}_k + \alpha\boldsymbol{d}_k) - (\boldsymbol{x}_k + \alpha\boldsymbol{d}_k)^{\mathrm{T}}\boldsymbol{b}.$$

This is a quadratic function of α, and thus we can compute the optimal step size by finding the α such that $\phi'(\alpha) = 0$. By expanding out the quadratic term, it is easy to show that

$$\phi'(\alpha) = \alpha\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{d}_k + \boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x}_k - \boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{b}.$$

Setting this equal to zero and solving for α yields the step size

$$\alpha_k = \frac{\boldsymbol{d}_k^{\mathrm{T}}(\boldsymbol{b} - \boldsymbol{Q}\boldsymbol{x}_k)}{\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{d}_k}.$$

Backtracking

Exact line search is generally not worth the trouble, but the problem with a fixed step size is that we cannot guarantee convergence of α is too large, but when α is too small we may not make much progress on

each iteration. A popular strategy is to do a rudimentary search for an α that gives us sufficient progress as measured by the inequality

$$f\left(\boldsymbol{x}_k\right) - f\left(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k\right) \geq -c_1 \alpha \left\langle \boldsymbol{d}_k, \nabla f\left(\boldsymbol{x}_k\right) \right\rangle,$$

where $c_1 \in (0,1)$. This is known as the **Armijo condition**. For α satisfying the inequality we have that the reduction in f is proportional to both the step length α and the directional derivative in the direction $\boldsymbol{d}_k$.

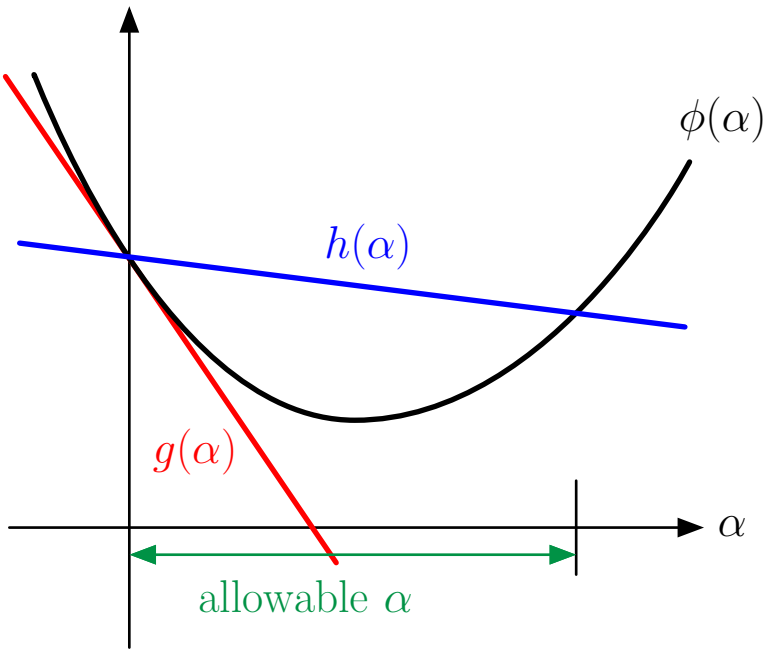
Note that we can equivalently write this condition as

$$\phi(\alpha) \leq h(\alpha) := \phi(0) + c_1 \alpha \phi'(0).$$

Recall that from convexity, we also have that

$$\phi(\alpha) \geq g(\alpha) := \phi(0) + \alpha \phi'(0).$$

Since $c_1 < 1$, we always have $\phi(\alpha) \leq h(\alpha)$ for sufficiently small α. An example is illustrated below:



We still haven't said anything about how to actually use the Armijo condition to pick α. Within the set of allowable α satisfying the condition, the (guaranteed) reduction in f is proportional to α, so we would generally like to select α to be large.

This inspires the following very simple **backtracking** algorithm: start with a step size of $\alpha = \bar{\alpha}$, and then decrease by a factor of ρ until the Armijo condition is satisfied.

Backtracking line search

Input: $\boldsymbol{x}_k$, $\boldsymbol{d}_k$, $\bar{\alpha} > 0$, $c_1 \in (0,1)$, and $\rho \in (0,1)$.

Initialize: $\alpha = \bar{\alpha}$

while $\phi(\alpha) > \phi(0) + c_1 \alpha \phi'(0)$ **do**

$\alpha = \rho\alpha$

end while

The backtracking line search tends to be cheap, and works very well in practice. A common choice for $\bar{\alpha}$ is $\bar{\alpha} = 1$, but this can vary somewhat depending on the algorithm. The choice of c_1 can range from extremely small (10^{-4}, encouraging larger steps) to relatively large (0.3, encouraging smaller steps), and typical values of ρ range from 0.1, (corresponding to a relatively coarse search) to 0.8 (corresponding to a finer search).

Wolfe conditions

The Armijo condition above guarantees that the selected step size provides some progress in terms of reducing f. A potential drawback,

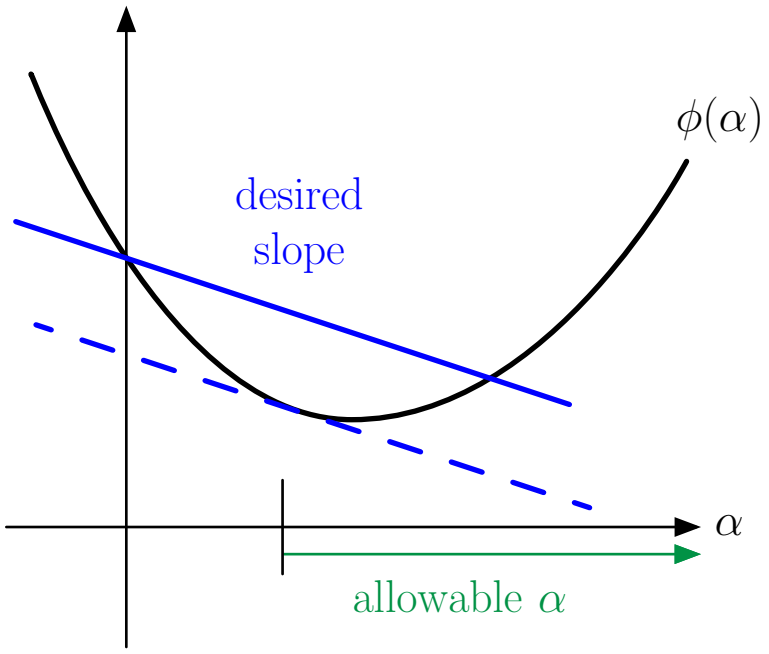
as can be seen in the figure, is that the Armijo condition does not rule out extremely small steps. To address this, it is sometimes helpful to impose an additional requirement on the step size:

$$\langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \rangle \geq c_2 \langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k) \rangle,$$

where $c_2 \in (0, 1)$. This condition is easier to interpret if we again recall that both sides of this inequality correspond to a directional derivative (in the direction of $\boldsymbol{d}_k$), and so this condition is equivalent to

$$\phi'(\alpha) \geq c_2 \phi'(0).$$

In words, this condition tells us to select a step size such that the slope of ϕ has increased by a certain factor compared to the initial slope $\phi'(0)$. For convex functions this translates to a minimum allowable step size, as illustrated below:



This condition together with the Armijo condition are collectively called the **Wolfe conditions**:

$$\begin{aligned} \phi(\alpha) &\leq \phi(0) + c_1 \alpha \phi'(0) \\ \phi'(\alpha) &\geq c_2 \phi'(0), \end{aligned}$$

where $0 < c_1 < c_2 < 1$.

In gradient descent (as well as other methods we will see soon, such as accelerated gradient descent and Newton's method), we can often dispense with the second of these conditions – the standard backtracking search already biases us away from making the step size much smaller than is required by the Armijo condition. However, in some cases (such as quasi-Newton methods) it will be important to explicitly enforce the second condition. Fortunately, the standard backtracking search can be easily modified to handle this by simply introducing an additional step at each iteration to check if the condition fails, in which case we must *increase* α to some value between the last two iterates.

Convergence of gradient descent

Here we will prove convergence guarantees for **gradient descent**, where we find a minimizer[1] of

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{x})$$

using our generic iterative algorithm choosing the direction to move as

$$\boldsymbol{d}_k = -\nabla f\left(\boldsymbol{x}_k\right),$$

resulting in the update rule

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f\left(\boldsymbol{x}_k\right).$$

Our goal is to establish the convergence rate of gradient descent. This can be measured in many different ways. One way is to establish

$$\begin{aligned} f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) &\leq \text{some function that decreases to 0 as } k \to \infty \\ &:= g(k) \end{aligned}$$

This established convergence of the *function values* to the minimum. With a result like this in hand, you can ask

How many iterations do we need to be within ϵ of a solution?

and the answer is

$$k \geq g^{-1}(\epsilon) \text{ iterations will suffice.}$$

[1] In this section, we will always assume that a minimizer exists.

For example, if we establish

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq 5/k^2$$

then we know that

$$k \geq \sqrt{\frac{5}{\epsilon}} \quad \Rightarrow \quad f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \epsilon.$$

Note that the $g(k)$ we derive will in general be monotonically decreasing and hence invertible.

If we know that there is a unique solution $\boldsymbol{x}^\star$, we might also bound

$$\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2 \leq \text{some function that decreases to 0 as } k \to \infty.$$

The bounds we develop will depend on the structural properties of the function f. In the mathematical optimization literature, there are results for all different kinds of structure on f. In this set of notes, we will consider two cases: convex differentiable f that

1. have an L-Lipschitz gradient map, i.e.

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2, \quad \text{for all } \boldsymbol{x}, \boldsymbol{y};$$

2. have an L-Lipschitz gradient and in addition are μ-strongly convex, i.e.

$$f(\boldsymbol{y}) \;\geq\; f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x}) \rangle + \frac{\mu}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 \quad \text{for all } \boldsymbol{x}, \boldsymbol{y}.$$

We will see that the additional structure added in the second case makes a dramatic difference in convergence rate.

Convergence of gradient descent: f smooth

As we have discussed before, having an L-Lipschitz gradient is akin to the function being smooth: if the derivative changes in a controlled manner as we move from point to point, the function itself will be very well-behaved.

On the homework, you showed that

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2, \tag{1}$$

means that we have the pointwise quadratic upper bound

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x})\rangle + \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 \tag{2}$$

This provides some intuition for what kind of structure the Lipschitz gradient condition imposes on f. Recall that for any convex function, we have

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x})\rangle.$$

So if f is convex, then at any point $\boldsymbol{x}$ we can bound f from *below* by a linear approximation. If in addition, if f has a Lipschitz gradient, (2) we can also bound it from *above* using a quadratic approximation. We will often refer to functions that obey (1) as *L-smooth*.

Now, let's consider running gradient descent on such a function with a **fixed step size**[2] $\alpha_k = 1/L$. Recall that the central gradient descent iteration is just

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k).$$

[2]This requires that you know L, which may not be possible in practice. In fact, if $\alpha < 1/L$ you will still get convergence, it will simply be slower. Moreover, it is not too hard to extend this approach to get a similar guarantee when using a backtracking line search.

From our assumption that f is L-smooth, we know that f satisfies (2), and thus plugging in $\boldsymbol{y} = \boldsymbol{x}_{k+1}$, we obtain

$$\begin{aligned} f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) + \left\langle -\frac{1}{L}\nabla f(\boldsymbol{x}_k), \nabla f(\boldsymbol{x}_k) \right\rangle + \frac{L}{2}\left\| \frac{1}{L}\nabla f(\boldsymbol{x}_k) \right\|_2^2 \\ &= f(\boldsymbol{x}_k) - \frac{1}{L}\|\nabla f(\boldsymbol{x}_k)\|_2^2 + \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|_2^2 \\ &= f(\boldsymbol{x}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|_2^2. \end{aligned} \tag{3}$$

Note that (3) shows that $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ as long as we are not already at the solution, so we are at least guaranteed to make some progress at each iteration. In fact, it says a bit more, giving us a guarantee regarding *how much* progress we are making, namely that

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|_2^2,$$

so that if the gradient is large we are guaranteed to make a large amount of progress.

In the Technical Details section at the end of these notes, we show that by combining this result with the definition of convexity and doing some clever manipulations, we can get a guarantee of the form

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{L}{2k}\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Thus, for L-smooth functions, we can guarantee that the error is $O(1/k)$ after k iterations. Another way to put this is to say that we can guarantee accuracy

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \epsilon$$

as long as

$$k \geq \frac{L}{2\epsilon}\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Note that if ϵ is very small, this says we can expect to need a very large number of iterations.

Convergence of gradient descent: smooth and strongly convex

We will now show that the convergence rate is much faster if f is strongly convex in addition to being smooth. Recall that for a μ-strongly convex function, we have

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x}) \rangle + \frac{\mu}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2. \tag{4}$$

for all $\boldsymbol{x}, \boldsymbol{y}$.

We will use the same fixed step size $\alpha_k = 1/L$, and begin our analysis in the same way as before, in which we derived the intermediate result (3) that the L-smoothness of f implies

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|_2^2.$$

We can now use strong convexity to obtain a lower bound on $\|\nabla f(\boldsymbol{x})\|_2^2$.

We can obtain a simpler lower bound for $f(\boldsymbol{y})$ by determining the smallest value that the right-hand side of (4) could ever take over all possible choices of $\boldsymbol{y}$. To do this, we simply minimize this lower bound by taking the gradient with respect to $\boldsymbol{y}$ and setting it equal to zero:

$$\nabla f(\boldsymbol{x}) + \mu(\boldsymbol{y} - \boldsymbol{x}) = 0,$$

From this we obtain that the lower bound in (4) will be minimized by

$$\boldsymbol{y} - \boldsymbol{x} = -\frac{1}{\mu}\nabla f(\boldsymbol{x}).$$

Plugging this into (4) yields

$$\begin{aligned} f(\boldsymbol{y}) &\geq f(\boldsymbol{x}) - \frac{1}{\mu}\|\nabla f(\boldsymbol{x})\|_2^2 + \frac{1}{2\mu}\|\nabla f(\boldsymbol{x})\|_2^2 \\ &= f(\boldsymbol{x}) - \frac{1}{2\mu}\|\nabla f(\boldsymbol{x})\|_2^2. \end{aligned}$$

In particular, this applies when $\boldsymbol{y} = \boldsymbol{x}^\star$, which after some rearranging yields

$$\|\nabla f(\boldsymbol{x})\|_2^2 \geq 2\mu\left(f(\boldsymbol{x}) - f(\boldsymbol{x}^\star)\right). \tag{PL}$$

This is a famous and useful result, often referred to as the **Polyak-Łojasiewicz inequality**.

Combining the PL inequality with (3) we obtain

$$\begin{aligned} f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^\star) &\leq f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) - \frac{\mu}{L}\left(f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star)\right) \\ &= \left(1 - \frac{\mu}{L}\right)\left(f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star)\right). \end{aligned}$$

That is, the gap between the current value of the objective function and the optimal value is cut down by a factor of $1 - \mu/L < 1$ at each iteration. (Note that (2) and (4) imply that $L \geq \mu$.)

This is an example of *linear convergence*; it is easy to apply the above iteratively to show that

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \;\leq\; \left(1 - \frac{\mu}{L}\right)^k \left(f(\boldsymbol{x}_0) - f(\boldsymbol{x}^\star)\right). \tag{5}$$

If we use $\epsilon_0 = f(\boldsymbol{x}_0) - f(\boldsymbol{x}^\star)$ to denote the initial error, this means that we can guarantee that

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \epsilon$$

for

$$\begin{aligned} k &\geq \frac{\log(\epsilon/\epsilon_0)}{\log(1-\mu/L)} \\ &\geq \frac{L}{\mu} \log\left(\frac{\epsilon_0}{\epsilon}\right), \end{aligned}$$

where the second inequality uses the fast that $-\log(1-\alpha) \geq \alpha$ for all $0 \leq \alpha < 1$.

Let's step back for a moment, and compare

$$\frac{1}{\epsilon} \quad \text{versus} \quad \log\left(\frac{1}{\epsilon}\right).$$

What are these quantities when $\epsilon = 10^{-2}$? What about 10^{-6}? This is all to say that the performance guarantees for gradient descent are dramatically better when f is strictly convex than when it is not.

We can also use (5) to characterize the convergence of the iterates $\boldsymbol{x}_k$ to the unique solution $\boldsymbol{x}^\star$. Applying (4) with $\boldsymbol{x} = \boldsymbol{x}^\star$ and $\boldsymbol{y} = \boldsymbol{x}_k$ yields (after noting $\nabla f(\boldsymbol{x}^\star) = \boldsymbol{0}$)

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \geq \frac{\mu}{2}\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2,$$

while applying (2) with $\boldsymbol{x} = \boldsymbol{x}^\star$ and $\boldsymbol{y} = \boldsymbol{x}_0$ yields

$$f(\boldsymbol{x}_0) - f(\boldsymbol{x}^\star) \leq \frac{L}{2}\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Combining these with (5) yields

$$\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 \;\leq\; \frac{L}{\mu}\left(1 - \frac{\mu}{L}\right)^k \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2,$$

so $\boldsymbol{x}_k \to \boldsymbol{x}^\star$ at a linear rate as well. I will note that a more careful analysis (which we won't go into here) can also remove the factor of L/μ in front, yielding

$$\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 \;\leq\; \left(1 - \frac{\mu}{L}\right)^k \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Finally, we also note that the PL inequality above also provides some guidance in terms of setting a stopping criterion. Specifically, if we declare convergence when $\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \epsilon$ then the PL inequality allows us to conclude that

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{1}{2\mu}\|\nabla f(\boldsymbol{x}_k)\|_2^2 \leq \frac{\epsilon^2}{2\mu}.$$

This provides a principled way of declaring convergence.

Technical Details: L-smooth convergence

Here we complete the convergence analysis for gradient descent on L-smooth functions that is summarized above. Specifically, recall that above in (3) we showed that if f is L-smooth then

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|_2^2.$$

Moreover, by the convexity of f,

$$f(\boldsymbol{x}_k) \leq f(\boldsymbol{x}^\star) + \langle \boldsymbol{x}_k - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}_k)\rangle,$$

where $\boldsymbol{x}^\star$ is a minimizer of f, and so we have

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}^\star) + \langle \boldsymbol{x}_k - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x})\rangle - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|_2^2.$$

Substituting $\nabla f(\boldsymbol{x}_k) = L(\boldsymbol{x}_k - \boldsymbol{x}_{k+1})$ then yields

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^\star) \leq L\langle \boldsymbol{x}_k - \boldsymbol{x}^\star, \boldsymbol{x}_k - \boldsymbol{x}_{k+1}\rangle - \frac{L}{2}\|\boldsymbol{x}_k - \boldsymbol{x}_{k+1}\|_2^2. \quad (6)$$

We can re-write this in a slightly more convenient way using the fact that

$$\|\boldsymbol{a} - \boldsymbol{b}\|_2^2 = \|\boldsymbol{a}\|_2^2 - 2\langle \boldsymbol{a}, \boldsymbol{b}\rangle + \|\boldsymbol{b}\|_2^2$$

and thus

$$2\langle \boldsymbol{a}, \boldsymbol{b}\rangle - \|\boldsymbol{b}\|_2^2 = \|\boldsymbol{a}\|_2^2 - \|\boldsymbol{a} - \boldsymbol{b}\|_2^2.$$

Setting $\boldsymbol{a} = \boldsymbol{x}_k - \boldsymbol{x}^\star$ and $\boldsymbol{b} = \boldsymbol{x}_k - \boldsymbol{x}_{k+1}$ and applying this to (6), we obtain the bound

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^\star) \leq \frac{L}{2}\left(\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2^2\right).$$

This result bounds how far away $f(\boldsymbol{x}_{k+1})$ is from the optimal $f(\boldsymbol{x}^\star)$ in terms (primarily) of the error in the previous iteration: $\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2$. We can use this result to bound $f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^\star)$ in terms of the initial error $\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2$ by a clever argument.

Specifically, this bound holds not only for iteration k, but for all iterations $i = 1, \ldots, k$, so we can write down k inequalities and then sum them up to obtain

$$\sum_{i=1}^{k} f(\boldsymbol{x}_i) - f(\boldsymbol{x}^\star) \leq \frac{L}{2} \left(\sum_{i=1}^{k} \|\boldsymbol{x}_{i-1} - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 \right).$$

The right-hand side of this inequality is what is called a *telescopic sum*: each successive term in the sum cancels out part of the previous term. Once you write this out, all the terms cancel except for two (one component from the $i = 1$ term and one from the $i = k$ term) giving us:

$$\begin{aligned} \sum_{i=1}^{k} f(\boldsymbol{x}_i) - f(\boldsymbol{x}^\star) &\leq \frac{L}{2} \left(\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 \right) \\ &\leq \frac{L}{2} \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2. \end{aligned}$$

Since, as noted above, $f(\boldsymbol{x}_i)$ is monotonically decreasing in i, we also have that

$$k \left(f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \right) \leq \sum_{i=1}^{k} f(\boldsymbol{x}_i) - f(\boldsymbol{x}^\star),$$

and thus

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{L}{2k} \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2,$$

which is exactly what we wanted to show.

Accelerated first-order methods

In the last lecture we provided convergence guarantees for gradient descent under two different assumptions. Under the stronger assumption that f was both L-smooth *and* strongly convex with parameter μ, we showed that convergence to a tolerance of ϵ was possible in $O(\frac{L}{\mu}\log(1/\epsilon))$ iterations. Under the weaker assumption where we only assume that f is L-smooth, we were able to show that $O(L/\epsilon)$ iterations would be sufficient.

In this lecture we show that there are small changes we can make to gradient descent that can dramatically improve its performance, both in theory (resulting in improvements on the bounds above) and in practice. We will talk about two of these here: the heavy ball method and Nesterov's "optimal algorithm." Both of these strategies incorporate the idea of *momentum*, although in subtly different ways.

Momentum

One way to interpret gradient descent is as a discretization to the *gradient flow* differential equation

$$\begin{aligned} \boldsymbol{x}'(t) &= -\nabla f(\boldsymbol{x}(t)), \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0. \end{aligned} \tag{1}$$

The solution to (1) is a curve that tracks the direction of steepest descent directly to the minimizer, where it arrives at a fixed point (where $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$). To see how gradient descent arises as a discretization of (1), suppose we approximate the derivative with a forward difference

$$\boldsymbol{x}'(t) \approx \frac{\boldsymbol{x}(t+h) - \boldsymbol{x}(t)}{h},$$

for some small h. So if we think of $\boldsymbol{x}_{k+1}$ and $\boldsymbol{x}_k$ as closely spaced time points, we can interpret

$$\frac{1}{\alpha}\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right) = -\nabla f(\boldsymbol{x}_k),$$

as a discrete approximation to gradient flow. Re-arranging the equation above yields the gradient descent iteration $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k)$.

The problem is once we perform this discretization, the path tends to oscillate. One way to get a more regular path is to consider an alternative differential equation that also has a fixed point where $\nabla f(\boldsymbol{x}) = 0$ but also incorporates a second-order term:

$$m\boldsymbol{x}''(t) + \boldsymbol{x}'(t) = -\nabla f(\boldsymbol{x}(t)). \tag{2}$$

From a physical perspective, this is a model for a particle with mass m moving in a potential field with friction. This results in trajectories that develop momentum (a heavy ball will move down a hill faster than a light one in the presence of friction). In the case where $m = 0$ we recover (1), but in general the inclusion of the mass term above will result in a more accelerated trajectory towards the solution.

We can discretize the dynamics as before by setting

$$\boldsymbol{x}''(t) \approx \frac{\boldsymbol{x}_{k+1} - 2\boldsymbol{x}_k + \boldsymbol{x}_{k-1}}{h_1}, \quad \boldsymbol{x}'(t) \approx \frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{h_2}.$$

If we plug these into (2) and rearrange we obtain an update rule of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \beta_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) - \alpha_k \nabla f(\boldsymbol{x}_k), \tag{3}$$

where $\beta = h_1/h_2 m$ and $\alpha = h_1/m$. This is the core iteration for the **heavy ball method**, introduced by Polyak in 1964 [Pol64]. The $\boldsymbol{x}_k - \boldsymbol{x}_{k-1}$ term above adds a little bit of the last step $\boldsymbol{x}_k - \boldsymbol{x}_{k-1}$ direction into the new step direction $\boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ – this method is also referred to as *gradient descent with momentum*.

Convergence of the heavy ball method

In the previous lecture we showed that if $f(\boldsymbol{x})$ is L-smooth and strongly convex, then we can obtain a bound of the form

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^\star) \leq \left(1 - \frac{1}{\kappa}\right)^k \left(f(\boldsymbol{x}_0) - f(\boldsymbol{x}^\star)\right),$$

where $\kappa = L/\mu$ is the "condition number." From this we showed that we can guarantee

$$\frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star)}{f(\boldsymbol{x}_0) - f(\boldsymbol{x}^\star)} \leq \epsilon$$

provided that

$$k \geq \kappa \log\left(1/\epsilon\right).$$

In the Technical Details at the end of these notes we also provide an alternative argument for the convergence of gradient descent that begins by showing that

$$\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2 \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^k \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2.$$

Using a similar argument as before, we can use this to show that

$$\frac{\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2}{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2} \leq \epsilon$$

provided that

$$k \geq \kappa \log(1/\epsilon).$$

(Note that

$$\frac{\kappa - 1}{\kappa + 1} = 1 - \frac{2}{\kappa + 1} \leq 1 - 1/\kappa = 1 - \mu/L,$$

where the inequality comes from the fact that $\kappa \geq 1$.)

The heavy ball method significantly improves on this result in terms of its dependence on κ.

Specifically, under the same assumptions as before (L-smoothness and strong convexity), in the Technical Details section we show (for the quadratic case) that for the heavy ball method with

$$\alpha_k = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2} \quad \text{and} \quad \beta_k = \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}} \right)^2$$

we can achieve

$$\frac{\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2}{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2} \leq \epsilon \quad \text{when} \quad k \gtrsim \sqrt{\kappa}\, \log(1/\epsilon).$$

The difference with gradient descent can be significant. When $\kappa = 10^2$, we are asking for $\approx 100 \log(1/\epsilon)$ iterations for gradient descent, as compared with $\approx 10 \log(1/\epsilon)$ from the heavy ball method.

Conjugate gradients

If you are familiar with the *method of conjugate gradients* (CG), some of this may feel vaguely familiar. If you have never heard of CG, I highly recommend reading through the tutorial "An introduction to the conjugate gradient method without the agonizing pain" [She94].

The CG method was developed for minimizing quadratic functions of the form $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}$. While it is normally presented in quite a different fashion, it ultimately boils down to being a variant of

the heavy ball method that is particularly well-suited to minimizing quadratic functions. To see this connection, note that the core CG iteration can be expressed[1] as

$$\begin{aligned} \boldsymbol{d}_k &= -\nabla f(\boldsymbol{x}_k) + \beta_k \boldsymbol{d}_{k-1} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k, \end{aligned}$$

where we start with $\boldsymbol{d}_0 = -\nabla f(\boldsymbol{x}_0)$. In CG, the β_k are set as

$$\beta_k = \frac{\|\nabla f(\boldsymbol{x}_k)\|_2^2}{\|\nabla f(\boldsymbol{x}_{k-1})\|_2^2}.$$

If $f(\boldsymbol{x})$ is a quadratic function this choice ensures that at each iteration $\boldsymbol{d}_k$ is *conjugate* to $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{k-1}$. We won't worry about saying more about this beyond the fact that this is a good idea *if* $f(\boldsymbol{x})$ *is quadratic*. Once β_k is fixed, α_k can then be chosen using a line search. Again, if $f(\boldsymbol{x})$ is quadratic, there is a simple closed form solution for this (which we have previously derived).

While CG is parameterized differently than the heavy ball method as described in (3), they are fundamentally the same. To see this note that we can also write

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \left(-\nabla f(\boldsymbol{x}_k) + \beta_k \boldsymbol{d}_{k-1}\right) \\ &= \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) + \alpha_k \beta_k \frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{\alpha_{k-1}}. \end{aligned}$$

This is precisely the same iteration as (3), but with a slightly different way of parameterizing the weight being applied to the momentum term.

[1] You will typically see this algorithm described specifically for the quadratic case, in which case $\nabla f(\boldsymbol{x}) = Q\boldsymbol{x} - \boldsymbol{b}$ and these calculations are carefully broken up to re-use as many calculations as possible and avoid any unnecessary matrix-vector multiplies, so it may initially look quite different.

If you are trying to minimize a quadratic function, CG is the way to go. The convergence guarantees you get for CG when minimizing a quadratic function are just as good (but not actually better than) what you have for the heavy ball method, but you don't need to know anything like Lipschitz or strong convexity parameters (which would correspond to the maximum and minimum eigenvalues of $\boldsymbol{Q}$) in order to choose the α_k and β_k.

However, if you are trying to minimize *anything else* CG is not necessarily a good choice. The choices for α_k and β_k are highly tuned to the quadratic setting and can yield unstable results in general.

Nesterov's "optimal" method

In the case where f is strictly convex, you can come up with examples that show that the convergence rate of the heavy ball method can't be improved in general. For non-strictly convex f, the story is more complicated.

Recall that we also have a convergence result for gradient descent in the case where we only assume L-smoothness. In particular, last time we showed that for a fixed step size $\alpha = 1/L$,

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{L}{2k} \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Thus, to reduce the error by a factor of ϵ requires

$$k \geq \frac{L}{2\epsilon}$$

iterations.

In 1983, Yuri Nesterov proposed a slight variation on the heavy ball method that can improve on this theory, and often works better in

practice [Nes83].[2] Specifically, recall the heavy ball method, which can be represented via the iteration:

$$\begin{aligned} \boldsymbol{p}_k &= \beta_k \left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1} \right) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{p}_k - \alpha_k \nabla f(\boldsymbol{x}_k), \end{aligned}$$

where we start with $\boldsymbol{p}_0 = \mathbf{0}$. Nesterov's method makes a subtle, but significant, change to this iteration:

$$\begin{aligned} \boldsymbol{p}_k &= \beta_k \left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1} \right) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{p}_k - \alpha_k \nabla f(\boldsymbol{x}_k + \boldsymbol{p}_k). \end{aligned} \tag{4}$$

Notice that this is the same as heavy ball *except* that there is also a momentum term *inside* the gradient expression. With this iteration, we will show that (for a suitable choice of α_k and β_k)

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \lesssim \frac{L}{k^2} \left\| \boldsymbol{x}_0 - \boldsymbol{x}^\star \right\|_2^2,$$

meaning that we can reduce the error by a factor of ϵ in

$$k \gtrsim \frac{1}{\sqrt{\epsilon}},$$

iterations. When $\epsilon \sim 10^{-4}$, this is much, much better than $1/\epsilon$.

Nesterov's method is called "optimal" because it is impossible to beat the $1/k^2$ rate using only function and gradient evaluations. There are careful demonstrations of this in the literature (e.g., in [Nes04]).

Note that in practice, α_k can be chosen using a standard line search, and a good choice of β_k (both in practice, and as we will show below,

[2] Note that this method remained to a large extent unknown in the wider community until his 2004 publication (in English) of [Nes04].

in theory) turns out to be

$$\beta_k = \frac{k-1}{k+2}. \tag{5}$$

This tells us that we should initially not provide much weight to the momentum term, which makes intuitive sense as the initial gradients may not be pushing us in the right direction, but as we proceed we should have increased confidence that we are headed in the right direction and increase how much weight we place on the momentum term.

Significantly, note that in setting β_k we do *not* need to know anything about the function we are minimizing (such as strong convexity parameters). This represents an important advantage compared to the heavy ball method described above.

Convergence analysis of Nesterov's method

Analyzing the convergence of Nesterov's method under the assumption of L-smoothness is a little more involved than for gradient descent, but the overall approach is the same and contains many of the same elements, so we will start by recalling the main building blocks that we used in analyzing gradient descent.

Consequences of convexity and L-smoothness

First, we recall some basic facts that hold for any $\boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom} f$. Since f is convex we have

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x}) \rangle. \tag{6}$$

Since f is L-smooth we have

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x}) \rangle + \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2. \tag{7}$$

As a consequence of (7) (by setting $\boldsymbol{y} = \boldsymbol{x} - \frac{1}{L}\nabla f(\boldsymbol{x})$), we have that for any $\boldsymbol{x}$,

$$f\left(\boldsymbol{x} - \frac{\nabla f(\boldsymbol{x})}{L}\right) \leq f(\boldsymbol{x}) - \frac{\|\nabla f(\boldsymbol{x})\|_2^2}{2L}. \tag{8}$$

Combining this with the upper bound on $f(\boldsymbol{x})$ that you can obtain by rearranging (6), we obtain

$$f\left(\boldsymbol{x} - \frac{\nabla f(\boldsymbol{x})}{L}\right) \leq f(\boldsymbol{y}) + \langle \boldsymbol{x} - \boldsymbol{y}, \nabla f(\boldsymbol{x})\rangle - \frac{\|\nabla f(\boldsymbol{x})\|_2^2}{2L}. \tag{9}$$

As we will see below, this inequality is the foundation of our analysis of both gradient descent and Nesterov's method. By plugging in different choices for $\boldsymbol{y}$ (such as $\boldsymbol{x}_k$ or $\boldsymbol{x}^\star$) we can obtain both *lower* bounds on how much progress we make when we take a gradient step as well as *upper* bounds on how far away we are from a global optimum.

Convergence of gradient descent

Recall that in our analysis for gradient we assume a fixed step size $\alpha = 1/L$, resulting in an update rule of

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{\nabla f(\boldsymbol{x}_k)}{L}.$$

Thus, setting $\boldsymbol{x} = \boldsymbol{x}_k$ and $\boldsymbol{y} = \boldsymbol{x}^\star$ in (9) implies that

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}^\star) + L\langle \boldsymbol{x}_k - \boldsymbol{x}^\star, \boldsymbol{x}_k - \boldsymbol{x}_{k+1}\rangle - \frac{L}{2}\|\boldsymbol{x}_k - \boldsymbol{x}_{k+1}\|_2^2.$$

From this, if we define $\delta_k = f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star)$ and do some algebraic manipulation (see the previous notes) we get a bound of the form

$$\delta_{k+1} \leq \frac{L}{2}\left(\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2^2\right).$$

This yields the *telescopic sum*

$$\begin{aligned}\sum_{i=0}^{k-1} \delta_{i+1} &\leq \frac{L}{2}\left(\sum_{i=0}^{k-1} \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{i+1} - \boldsymbol{x}^\star\|_2^2\right) \\ &= \frac{L}{2}\left(\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2\right) \\ &\leq \frac{L}{2}\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.\end{aligned}$$

The proof for gradient descent concludes by noting that

$$\delta_k \leq \frac{1}{k}\sum_{i=0}^{k-1} \delta_{i+1} \leq \frac{L}{2k}\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Convergence of Nesterov's method

We will follow a similar argument to analyze Nesterov's method. We will again take $\alpha_k = 1/L$, but we will see that the analysis suggests a natural choice for β_k. With this choice of α_k, the main iteration from (4) is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k + \boldsymbol{p}_k).$$

It will be convenient to define

$$\boldsymbol{g}_k = -\frac{1}{L}\nabla f(\boldsymbol{x}_k + \boldsymbol{p}_k),$$

so that the main iteration becomes simply $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k + \boldsymbol{g}_k$. With this notation, by setting $\boldsymbol{x} = \boldsymbol{x}_k + \boldsymbol{p}_k$ in (9) we obtain the bound

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{y}) - L\langle \boldsymbol{x}_k - \boldsymbol{p}_k - \boldsymbol{y}, \boldsymbol{g}_k\rangle - \frac{L}{2}\|\boldsymbol{g}_k\|_2^2. \tag{10}$$

If we set $\boldsymbol{y} = \boldsymbol{x}^\star$ in (10) and again let δ_k denote $f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star)$ we obtain

$$\delta_{k+1} \leq \frac{L}{2}\left(2\langle \boldsymbol{x}^\star - \boldsymbol{x}_k - \boldsymbol{p}_k, \boldsymbol{g}_k\rangle - \|\boldsymbol{g}_k\|_2^2\right). \tag{11}$$

In our analysis of gradient descent, we then tried to rearrange an analogous bound to obtain a telescopic sum, but that doesn't quite work here. Instead we will need to combine (11) with another bound. Noting that $\delta_k - \delta_{k+1} = f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1})$, we observe that setting $\boldsymbol{y} = \boldsymbol{x}_k$ in (10) yields

$$\delta_k - \delta_{k+1} \geq \frac{L}{2}\left(2\langle \boldsymbol{p}_k, \boldsymbol{g}_k\rangle + \|\boldsymbol{g}_k\|_2^2\right). \tag{12}$$

We now consider the inequality formed by adding together (11) and $1 - \lambda_k$ times (12) (where λ_k is something we will choose later, but satisfies $\lambda_k \geq 1$, so that this multiplication switches the direction of the inequality). The left-hand side of the sum will be

$$\delta_{k+1} + (1-\lambda_k)(\delta_k - \delta_{k+1}) = \lambda_k \delta_{k+1} - (\lambda_k - 1)\delta_k.$$

The right-hand side of the sum will be

$$\begin{aligned}
&\frac{L}{2}\left(2\langle \boldsymbol{x}^\star - \boldsymbol{x}_k - \boldsymbol{p}_k + (1-\lambda_k)\boldsymbol{p}_k, \boldsymbol{g}_k\rangle - \|\boldsymbol{g}_k\|_2^2 + (1-\lambda_k)\|\boldsymbol{g}_k\|_2^2\right) \\
&\quad = \frac{L}{2}\left(2\langle \boldsymbol{x}^\star - \boldsymbol{x}_k - \lambda_k\boldsymbol{p}_k, \boldsymbol{g}_k\rangle - \lambda_k\|\boldsymbol{g}_k\|_2^2\right) \\
&\quad = \frac{L}{2\lambda_k}\left(2\langle \boldsymbol{x}^\star - \boldsymbol{x}_k - \lambda_k\boldsymbol{p}_k, \lambda_k\boldsymbol{g}_k\rangle - \|\lambda_k\boldsymbol{g}_k\|_2^2\right) \\
&\quad = \frac{L}{2\lambda_k}\left(\|\boldsymbol{x}^\star - \boldsymbol{x}_k - \lambda_k\boldsymbol{p}_k\|_2^2 - \|\boldsymbol{x}^\star - \boldsymbol{x}_k - \lambda_k\boldsymbol{p}_k - \lambda_k\boldsymbol{g}_k\|_2^2\right),
\end{aligned}$$

where the last equality follows from the easily verified fact that $2\langle \boldsymbol{a}, \boldsymbol{b}\rangle - \|\boldsymbol{b}\|_2^2 = \|\boldsymbol{a}\|_2^2 - \|\boldsymbol{a} - \boldsymbol{b}\|_2^2$. If we make the substitution

$\boldsymbol{u}_k = \boldsymbol{x}_k + \lambda_k \boldsymbol{p}_k$, then combining these yields the inequality

$$\lambda_k^2 \delta_{k+1} - (\lambda_k^2 - \lambda_k)\delta_k \leq \frac{L}{2}\left(\|\boldsymbol{x}^\star - \boldsymbol{u}_k\|_2^2 - \|\boldsymbol{x}^\star - \boldsymbol{u}_k - \lambda_k \boldsymbol{g}_k\|_2^2\right). \tag{13}$$

We will now show that if we choose λ_k and β_k appropriately, (13) will yield a telescopic sum on both sides. This will occur on right-hand side of (13) if

$$\boldsymbol{u}_k + \lambda_k \boldsymbol{g}_k = \boldsymbol{u}_{k+1}.$$

Noting that $\boldsymbol{p}_{k+1} = \beta_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \beta_{k+1}(\boldsymbol{p}_k + \boldsymbol{g}_k)$, we can write

$$\begin{aligned}
\boldsymbol{u}_{k+1} &= \boldsymbol{x}_{k+1} + \lambda_{k+1}\boldsymbol{p}_{k+1} \\
&= \boldsymbol{x}_k + \boldsymbol{p}_k + \boldsymbol{g}_k + \lambda_{k+1}\beta_{k+1}(\boldsymbol{p}_k + \boldsymbol{g}_k) \\
&= \boldsymbol{x}_k + (1 + \lambda_{k+1}\beta_{k+1})(\boldsymbol{p}_k + \boldsymbol{g}_k).
\end{aligned}$$

Thus, to make $\boldsymbol{u}_{k+1}$ equal to $\boldsymbol{u}_k + \lambda_k \boldsymbol{g}_k = \boldsymbol{x}_k + \lambda_k(\boldsymbol{p}_k + \boldsymbol{g}_k)$ we simply need to have

$$\lambda_k = 1 + \lambda_{k+1}\beta_{k+1} \Rightarrow \beta_{k+1} = \frac{\lambda_k - 1}{\lambda_{k+1}}. \tag{14}$$

For β_k satisfying (14), if we sum (13) from $i = 0$ to $k - 1$ we thus have

$$\begin{aligned}
\sum_{i=0}^{k-1} \lambda_i^2 \delta_{i+1} - (\lambda_i^2 - \lambda_i)\delta_i &\leq \frac{L}{2}\left(\|\boldsymbol{x}^\star - \boldsymbol{u}_0\|_2^2 - \|\boldsymbol{x}^\star - \boldsymbol{u}_k\|_2^2\right) \\
&\leq \frac{L}{2}\|\boldsymbol{x}^\star - \boldsymbol{u}_0\|_2^2 \\
&= \frac{L}{2}\|\boldsymbol{x}^\star - \boldsymbol{x}_0\|_2^2.
\end{aligned} \tag{15}$$

Next, one possible approach is to choose the λ_k so as to obtain a telescopic sum on the left-hand side of the inequality as well. This is the approach you will see most often in analyzing the convergence of Nesterov's method, but it is a little involved and leads to a recursive formula for λ_k (and hence β_k) instead of a simple closed form expression. Instead we will choose a simpler λ_k that yields essentially the same bound.

Specifically, suppose that we set $\lambda_k = (k+2)/2$. First, note that from (14) this yields

$$\beta_{k+1} = \frac{\frac{k+2}{2} - 1}{\frac{k+1}{2}} = \frac{k}{k+3},$$

which coincides with the rule for setting β_k given in (5). Next, note that we can write

$$\sum_{i=0}^{k-1} \lambda_i^2 \delta_{i+1} - (\lambda_i^2 - \lambda_i)\delta_i = (\lambda_0 - \lambda_0^2)\delta_0 + \lambda_{k-1}^2 \delta_k + \sum_{i=1}^{k-1} (\lambda_{i-1}^2 - \lambda_i^2 + \lambda_i)\delta_i.$$

Plugging in $\lambda_i = (i+2)/2$ yields

$$\begin{aligned}\sum_{i=0}^{k-1} \lambda_i^2 \delta_{i+1} - (\lambda_i^2 - \lambda_i)\delta_i &= \left(\frac{k+1}{2}\right)^2 \delta_k + \frac{1}{4}\sum_{i=0}^{k-1} \delta_i \\ &\geq \left(\frac{k+1}{2}\right)^2 \delta_k,\end{aligned}$$

where the inequality follows since $\delta_i = f(\boldsymbol{x}_i) - f(\boldsymbol{x}^\star) \geq 0$. Combining this lower bound with (15) yields

$$\left(\frac{k+1}{2}\right)^2 \delta_k \leq \frac{L}{2}\|\boldsymbol{x}^\star - \boldsymbol{x}_0\|_2^2$$

or equivalently

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{2L}{(k+1)^2}\|\boldsymbol{x}^\star - \boldsymbol{x}_0\|_2^2,$$

which is exactly the $O(1/k^2)$ convergence rate we wanted.

Technical Details: Analysis of the heavy ball method

We will analyze the heavy ball method for the special case of a quadratic function:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$$

We will assume that the eigenvalues of $\boldsymbol{Q}$ are in $[\mu, L]$, and so $f(\boldsymbol{x})$ is both L-smooth and μ-strongly convex.

Gradient descent revisited

We will warm up for our analysis on the heavy ball method by quickly revisiting standard gradient descent. In the quadratic case, there is an easy argument that

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2 &\leq \frac{L-\mu}{L+\mu}\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2 \\ &= \frac{\kappa - 1}{\kappa + 1}\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2,\end{aligned}$$

where $\kappa = L/\mu$ is the condition number of $\boldsymbol{Q}$.

Since $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$ and $\nabla f(\boldsymbol{x}^\star) = \boldsymbol{0}$, we have

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2 &= \|\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) - \boldsymbol{x}^\star\|_2 \\ &= \|\boldsymbol{x}_k - \boldsymbol{x}^\star - \alpha_k \left(\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}^\star)\right)\|_2 \\ &= \|(\mathbf{I} - \alpha_k \boldsymbol{Q})(\boldsymbol{x}_k - \boldsymbol{x}^\star)\|_2 \\ &\leq \|\mathbf{I} - \alpha_k \boldsymbol{Q}\| \cdot \|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2\end{aligned}$$

Since we have a bound on the eigenvalues of $\boldsymbol{Q}$, we know that the maximum eigenvalue of the symmetric matrix $\mathbf{I} - \alpha_k \boldsymbol{Q}$ is no more than

$$\|\mathbf{I} - \alpha_k \boldsymbol{Q}\| \;\leq\; \max\left(|1 - \alpha_k \mu|, |1 - \alpha_k L|\right).$$

If we take $\alpha_k = 2/(L + \mu)$, we obtain

$$\|\mathbf{I} - \alpha_k \boldsymbol{Q}\| \;\leq\; \frac{L - \mu}{L + \mu} = \frac{\kappa - 1}{\kappa + 1},$$

and so

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2 \leq \left(\frac{\kappa - 1}{\kappa + 1}\right) \|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2,$$

and by induction on k

$$\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2 \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^k \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2.$$

Heavy ball

For the heavy ball method, we have a similar analysis[3] that ends in a better result. Recall the heavy ball iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \beta_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) - \alpha_k \nabla f(\boldsymbol{x}_k),$$

We will derive a bound on how quickly $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|^2 + \|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2$ goes to zero for fixed values of $\alpha_k = \alpha,\ \beta_k = \beta$ which we will choose

[3] These notes are derived from [Wri18].

later. Rewriting the iteration above, we have

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_{k+1} - \boldsymbol{x}^\star \\ \boldsymbol{x}_k - \boldsymbol{x}^\star \end{bmatrix}}_{\boldsymbol{z}_{k+1}} = \begin{bmatrix} \boldsymbol{x}_k + \beta(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) - \boldsymbol{x}^\star \\ \boldsymbol{x}_k - \boldsymbol{x}^\star \end{bmatrix} - \alpha \begin{bmatrix} \nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}^\star) \\ \boldsymbol{0} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{x}_k + \beta(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) - \boldsymbol{x}^\star \\ \boldsymbol{x}_k - \boldsymbol{x}^\star \end{bmatrix} - \alpha \begin{bmatrix} \boldsymbol{Q}\left(\boldsymbol{x}_k - \boldsymbol{x}^\star\right) \\ \boldsymbol{0} \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha\boldsymbol{Q} & -\beta\mathbf{I} \\ \mathbf{I} & \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{T}} \underbrace{\begin{bmatrix} \boldsymbol{x}_k - \boldsymbol{x}^\star \\ \boldsymbol{x}_{k-1} - \boldsymbol{x}^\star \end{bmatrix}}_{\boldsymbol{z}_k},$$

We have $\boldsymbol{z}_k = \boldsymbol{T}^k \boldsymbol{z}_0$, and so

$$\|\boldsymbol{z}_k\|_2 \ \leq \ \|\boldsymbol{T}^k\| \cdot \|\boldsymbol{z}_0\|_2,$$

so we want to bound the spectral norm (largest singular value) of $\boldsymbol{T}^k_k$.

We are now analyzing the rate of convergence (to zero) of a linear dynamical system. We know that the eigenvalues of $\boldsymbol{T}^k$ are the eigenvalues of $\boldsymbol{T}$ raised to the kth power. The only complicating factor is that $\boldsymbol{T}$ is not symmetric, and so the eigenvalues and singular values are not the same thing. We reconcile this using the *spectral radius*

$$\rho(\boldsymbol{T}) = \text{maximum magnitude of eigenvalues of } \boldsymbol{T}.$$

Two key results from linear algebra and dynamical systems are that $\rho(\boldsymbol{T}) \leq \|\boldsymbol{T}\|$ and

$$\rho(\boldsymbol{T}) = \lim_{k\to\infty} \|\boldsymbol{T}^k\|^{1/k}.$$

That is, for any given $\delta > 0$, there exists an n such that

$$\|\boldsymbol{T}^k\|^{1/k} \leq \rho(\boldsymbol{T}) + \delta,$$

for all $k \geq n$. Thus if we define the constant

$$C = \max_{0 \leq k \leq n} \frac{\|\boldsymbol{T}^k\|}{(\rho(\boldsymbol{T}) + \delta)^k},$$

we will have

$$\|\boldsymbol{T}^k\| \leq C \left(\rho(\boldsymbol{T}) + \delta\right)^k. \tag{16}$$

We are left with the task of bounding $\rho(\boldsymbol{T}) < 1$ and choosing an appropriate δ. (Note that if $\boldsymbol{T}$ were symmetric, we would simply have $\rho(\boldsymbol{T}) = \|\boldsymbol{T}\|$ and $\|\boldsymbol{T}^k\| = \|\boldsymbol{T}\|^k = \rho(\boldsymbol{T})^k$.)

We can get a start on this by taking an eigenvalue decomposition of the symmetric positive definite matrix $\boldsymbol{Q} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}$. Since $\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}} = \mathbf{I}$, we can write

$$\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha\boldsymbol{Q} & -\beta\mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{V} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{V} \end{bmatrix} \begin{bmatrix} (1+\beta)\mathbf{I} - \alpha\boldsymbol{\Lambda} & -\beta\mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{V}^{\mathrm{T}} \end{bmatrix}.$$

Since $\begin{bmatrix} \boldsymbol{V} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{V} \end{bmatrix}$ is orthonormal, its application on the right of a matrix and its transpose (inverse) on the left does not change the eigenvalues, and so we can study the spectral radius of

$$\boldsymbol{T}' = \begin{bmatrix} (1+\beta)\mathbf{I} - \alpha\boldsymbol{\Lambda} & -\beta\mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}.$$

Notice that this a $2N \times 2N$ matrix divided into 4 blocks, each of which is an $N \times N$ diagonal matrix. As such, there is also a permutation matrix $\boldsymbol{P}$ that we can apply on both the rows and columns to make this a block diagonal matrix (with 2×2 blocks along the diagonal):

$$\boldsymbol{P}\boldsymbol{T}'\boldsymbol{P}^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{T}'_1 & & \\ & \cdots & \\ & & \boldsymbol{T}'_N \end{bmatrix}, \quad \boldsymbol{T}'_n = \begin{bmatrix} 1+\beta-\alpha\lambda_n & -\beta \\ 1 & 0 \end{bmatrix}.$$

Since again the application of a matrix and its inverse on either side does not change the eigenvalues, we can compute the spectral radius of the matrix on the right. Since it is block diagonal, we know the spectral radius is the maximum of the individual spectral radii of the blocks. That is, we now have

$$\rho(\boldsymbol{T}) = \max_{1 \leq n \leq N} \rho(\boldsymbol{T}'_n).$$

Since it is a 2×2 matrix, we can compute the eigenvalues of $\boldsymbol{T}'_n$ exactly. We know that γ is an eigenvalue of $\boldsymbol{T}'_n$ if $\det(\boldsymbol{T}'_n - \gamma \mathbf{I}) = 0$, i.e. if

$$\gamma^2 - (1 + \beta - \alpha\lambda_n)\gamma + \beta = 0,$$

which means the eigenvalues are

$$(\gamma_1, \gamma_2) = \frac{1}{2}\left(1 + \beta - \alpha\lambda_n \pm \sqrt{(1 + \beta - \alpha\lambda_n)^2 - 4\beta}\right).$$

If we choose β so that the eigenvalues are complex,

$$4\beta > (1 + \beta - \alpha\lambda_n)^2 \tag{17}$$

then we have

$$(\gamma_1, \gamma_2) = \frac{1}{2}\left(1 + \beta - \alpha\lambda_n \pm j\sqrt{4\beta - (1 + \beta - \alpha\lambda_n)^2}\right),$$

and $|\gamma_1| = |\gamma_2| = \beta$, and hence $\rho(\boldsymbol{T}'_n) = \beta$. Using that fact that $\mu \leq \lambda_n \leq L$, we can ensure (17) holds when

$$\beta = \min(|1 - \sqrt{\alpha\mu}|^2, |1 - \sqrt{\alpha L}|^2).$$

We can now choose α so that these two terms are equal,

$$\alpha = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2} \quad \Rightarrow \quad 1 - \sqrt{\alpha\mu} = -(1 - \sqrt{\alpha L}) = \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}},$$

and so

$$\beta = \left(\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}\right)^2 = \left(1 - \frac{2}{\sqrt{\kappa}+1}\right)^2.$$

Taking $\delta = 1/(\sqrt{\kappa}+1)$ in (16) above and using $\beta^2 \leq \beta$, we have

$$\|\boldsymbol{z}_k\|_2 \;\leq\; C\left(1 - \frac{1}{\sqrt{\kappa}+1}\right)^k \|\boldsymbol{z}_0\|_2.$$

This means we are guaranteed that $\|\boldsymbol{z}_k\|_2 \leq \epsilon$ when

$$\begin{aligned} k &\geq (\sqrt{\kappa}+1)\log(C\epsilon_0/\epsilon), \quad \epsilon_0 = \|\boldsymbol{z}_0\|_2, \\ &\gtrsim \sqrt{\kappa}\log(\epsilon_0/\epsilon). \end{aligned}$$

References

[Nes83] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Proc. USSR Acad. Sci.*, 269:543–547, 1983.

[Nes04] Y. Nesterov. *Introductory Lectures on Convex Optimization*. Springer Science+Business Media, 2004.

[Pol64] B. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[She94] J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. 1994.

[Wri18] S. Wright. Gradient methods for optimization. Slides, Madison Summer School, July 2018.

Newton's Method

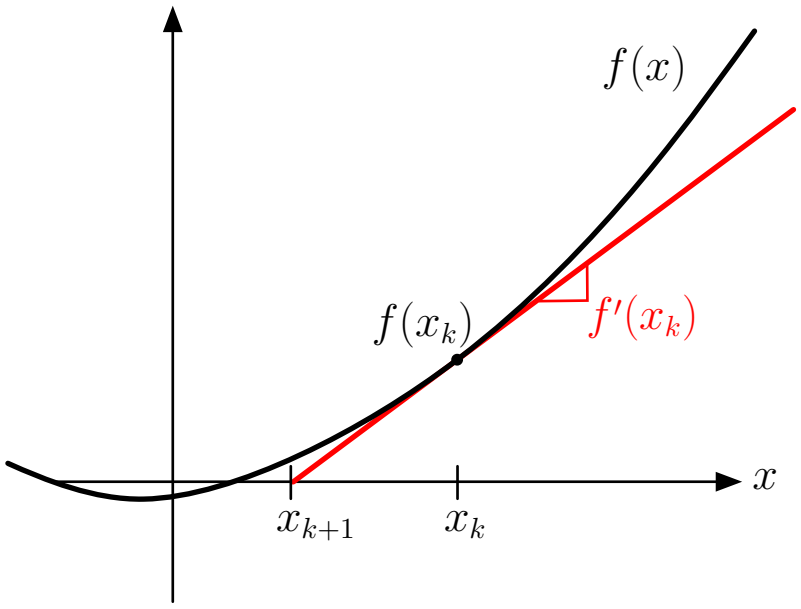
Newton's method is a classical technique for finding the root of a general differentiable function $f(x) : \mathbb{R} \to \mathbb{R}$. That is, we want to find an $x \in \mathbb{R}$ such that

$$f(x) = 0.$$

As you probably learned in high school, one technique for doing this is to start at some guess x_0, and then follow the iteration

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

This update results from taking a simple linear approximation at each step:



Of course, there can be many roots, and which one we converge to will depend on what we choose for x_0. It is also very much possible that the iterations do not converge for certain (or even almost all) initial values x_0.

However there is a classical convergence theory that says that once we are close enough to a particular root x_0, we will have

$$\underbrace{|x_0 - x_{k+1}|}_{\epsilon_{k+1}} \leq C \cdot \underbrace{(x_0 - x_k)^2}_{\epsilon_k^2},$$

where the constant C depends on the ratio between the first and second derivatives in the interval[1] around the root x_0:

$$C = \sup_{x \in \mathcal{I}} \frac{|f''(x)|}{2|f'(x)|}.$$

The take-away here is that close to the solution, Newton's methods exhibits *quadratic convergence*: the error at the next iteration is proportional to the square of the error at the last iteration. Since we are concerned with ϵ_k small, $\epsilon_k \ll 1$, this means that under the right conditions, the error goes down in dramatic fashion from iteration to iteration.

Notice that applying the technique requires that f is differentiable, but the convergence guarantee depends on f be twice (continuously) differentiable.

When $f(x)$ is convex, twice differentiable, and has a minimizer, we can find a minimizer by applying Newton's method to the derivative. We start at some initial guess x_0, and then take

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{1}$$

[1] There are various technical conditions that f must obey on $\mathcal{I}$ for this result to hold, including the second derivative being continuous and the first derivative not being equal to zero. Also, the condition "close enough" is characterized by looking at ratios of derivatives at the root and on $\mathcal{I}$. The Wikipedia article on this is not bad: https://en.wikipedia.org/wiki/Newton's_method.

Again, if f is three-times continuously differentiable, we converge to the global minimizer quadratically with a constant that depends on

$$C = \sup_{x \in \mathcal{I}} \frac{1}{2} \frac{|f'''(x)|}{|f''(x)|},$$

for an appropriate interval $\mathcal{I}$ around the solution. Again, applying the method relies on us being able to compute first and second derivatives of f, and the analysis relies on f being three-times differentiable.

We can interpret the iteration (1) above in the following way:

1. At x_k, approximate $f(x)$ using the Taylor expansion

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2.$$

2. Find the exact minimizer of this quadratic approximation. Taking the derivative of the expansion above and setting it equal to zero yields the following optimality condition for $\widehat{x}$ to be a minimizer:

$$f''(x_k) \cdot (\widehat{x} - x_k) = -f'(x_k).$$

   This is just a re-arrangement of the iteration (1).

3. Take $x_{k+1} = \widehat{x}$.

This last interpretation extends naturally to the case where $f(\boldsymbol{x})$ is a function of many variables, $f : \mathbb{R}^N \to \mathbb{R}$. We know that if f is convex and twice differentiable, we have a minimizer $\boldsymbol{x}^\star$ when $\nabla f(\boldsymbol{x}^\star) = \mathbf{0}$. Newton's method to find such a minimizer proceeds as above. We start with an initial guess $\boldsymbol{x}_0$, and use the following iteration:

1. Take a Taylor approximation around $f(\boldsymbol{x}_k)$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \boldsymbol{g} \rangle + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{H} (\boldsymbol{x} - \boldsymbol{x}_k)$$

where

$$\boldsymbol{g} = \nabla f(\boldsymbol{x}_k) = N \times 1 \text{ gradient vector at } \boldsymbol{x}_k$$
$$\boldsymbol{H} = \nabla^2 f(\boldsymbol{x}_k) = N \times N \text{ Hessian matrix at } \boldsymbol{x}_k.$$

2. Find the exact minimizer $\widehat{\boldsymbol{x}}$ to this approximation. This gives us the problem

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{H} (\boldsymbol{x} - \boldsymbol{x}_k).$$

Since $\boldsymbol{H} \in \mathcal{S}_+^N$ (since we are assuming f is convex), we know that the conditions for $\widehat{\boldsymbol{x}}$ being a minimizer[2] are

$$\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}_k) = -\boldsymbol{g}.$$

If $\boldsymbol{H}$ is invertible (i.e., $\boldsymbol{H} \in \mathcal{S}_{++}^N$), then we have a unique minimizer and

$$\widehat{\boldsymbol{x}} = \boldsymbol{x}_k - \boldsymbol{H}^{-1}\boldsymbol{g}.$$

3. Take $\boldsymbol{x}_{k+1} = \widehat{\boldsymbol{x}}$.

This procedure is often referred to as a *pure Newton step*, as it does not involve the selection of a step size. In practice, however, it is often beneficial to choose the step direction as

$$\boldsymbol{d}_k = -\left(\nabla^2 f(\boldsymbol{x}_k)\right)^{-1} \nabla f(\boldsymbol{x}_k),$$

and then choose a step size α_k using a backtracking line search, and then take

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

as before.

[2] Take the gradient of this new expression and set it equal to $\mathbf{0}$.

Convergence of Newton's Method

Suppose that $f(\boldsymbol{x})$ is strongly convex,

$$\mu \mathbf{I} \preceq \nabla^2 f(\boldsymbol{x}) \preceq L\mathbf{I}, \quad \forall \boldsymbol{x} \in \mathbb{R}^N,$$

and that its Hessian is Lipschitz,

$$\|\nabla^2 f(\boldsymbol{x}) - \nabla^2 f(\boldsymbol{y})\| \ \leq \ Q\|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

(The norm on the left-hand side above is the standard operator norm.) We will show that the Newton algorithm coupled with an exact line search[3] provides a solution with precision ϵ:

$$f(\boldsymbol{x}_k) - p^\star \leq \epsilon,$$

provided that the number of iterations satisfies

$$k \ \geq \ C_1 \left(f(\boldsymbol{x}_0) - p^\star\right) + \log_2 \log_2(\epsilon_0/\epsilon),$$

where we can take the constants above to be $C_1 = 2L^2Q^2/\mu^5$ and $\epsilon_0 = 2\mu^3/Q^2$. Qualitatively, this says that Newton's method takes a constant number of iterations to converge to any reasonable precision — we can bound $\log_2 \log_2(\epsilon_0/\epsilon) \leq 6$ (say) for ridiculously small values of ϵ.

To establish this result, we break the analysis into two stages. In the first, the *damped Newton stage*, we are far from the solution (as measured by $\|\nabla f(\boldsymbol{x}_k)\|_2$), but we make constant progress towards the answer. Specifically, we will show that in this stage,

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \ \geq \ 1/C_1.$$

[3] These results are easily extended to backtracking line searches; we are just using an exact line search to make the exposition easier. See [BV04, Sec. 9.5.3] for the analysis with backtracking.

This implies that when we are far from the solution, we reduce the gap $f(\boldsymbol{x}_k) - p^\star$ by at least $1/C_1$ at each iteration. It should be clear, then, that the number of damped Newton steps is no greater than $C_1 \left(f(\boldsymbol{x}_0) - p^\star\right)$.

We will then show that when $\|\nabla f(\boldsymbol{x}_k)\|_2$ is small enough, the gap closes dramatically at every iteration. We call this the *quadratic convergence stage*, as we will be able to show that once the algorithm enters this stage at iteration ℓ, for all $k > \ell$,

$$\|\nabla f(\boldsymbol{x}_k)\|_2 \ \leq \ C_2 \cdot 2^{-2^{k-\ell}},$$

where $C_2 = Q/(2\mu^2)$ is another constant.

Damped phase

We are in this stage when

$$\|\nabla f(\boldsymbol{x}_k)\|_2 \geq \mu^2/Q.$$

We take $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_{\text{exact}} \boldsymbol{d}_k$, where

$$\boldsymbol{d}_k = -\nabla^2 f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k),$$

and α_{exact} is the result of an exact line search[4]:

$$\alpha_{\text{exact}} = \operatorname*{arg\,min}_{0 \leq \alpha \leq 1} f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k).$$

We define the current *Newton decrement* as

$$\lambda_k = \sqrt{\nabla f(\boldsymbol{x}_k)^{\mathrm{T}} (\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k)},$$

[4]For convenience, we are not letting α be larger than 1, just as in a backtracking method.

and note that $\lambda_k^2 = -\nabla f(\boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{d}_k$. Moreover, strong convexity implies that the eigenvalues of $(\nabla^2 f(\boldsymbol{x}_k))^{-1}$ are at least $1/L$ and at most $1/\mu$, yielding the bounds

$$\|\boldsymbol{d}_k\|_2^2 \leq \frac{1}{\mu}\lambda_k^2 \qquad \text{and} \qquad \frac{1}{L}\|\nabla f(\boldsymbol{x}_k)\|_2^2 \leq \lambda_k^2,$$

which we will use below. From the L-smoothness of the gradient of f, we know that for any t we have

$$\begin{aligned} f(\boldsymbol{x}_k + t\boldsymbol{d}_k) &\leq f(\boldsymbol{x}_k) + \langle t\boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k \rangle + \frac{L}{2}\|t\boldsymbol{d}_k\|_2^2 \\ &= f(\boldsymbol{x}_k) - t\lambda_k^2 + \frac{Lt^2}{2}\|\boldsymbol{d}_k\|_2^2 \\ &\leq f(\boldsymbol{x}_k) - t\lambda_k^2 + \frac{Lt^2}{2\mu}\lambda_k^2. \end{aligned}$$

Plugging in $t = \mu/L$ above yields

$$\begin{aligned} f(\boldsymbol{x}_k + \alpha_{\text{exact}}\boldsymbol{d}_k) - f(\boldsymbol{x}_k) &\leq f\left(\boldsymbol{x}_k + \frac{\mu}{L}\boldsymbol{d}_k\right) - f(\boldsymbol{x}_k) \\ &\leq -\frac{\mu}{2L}\lambda_k^2 \\ &\leq -\frac{\mu}{2L^2}\|\nabla f(\boldsymbol{x}_k)\|_2^2 \\ &\leq -\frac{\mu^5}{2Q^2L^2}. \end{aligned}$$

Quadratic convergence

When

$$\|\nabla f(\boldsymbol{x}_k)\|_2 < \mu^2/Q,$$

we start to settle things very quickly. We will assume that in this stage, we choose the step size to be $\alpha_k = 1$. In fact, you can show that under very mild assumptions on the backtracking parameter ($c < 1/3$, to be specific), backtracking will indeed not backtrack at all and return $\alpha_k = 1$ (see [BV04, p. 490]).

We start by pointing out that by construction,

$$\nabla^2 f(\boldsymbol{x}_k)\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k),$$

and so by the fundamental theorem of calculus,

$$\begin{aligned}
\nabla f(\boldsymbol{x}_{k+1}) &= \nabla f(\boldsymbol{x}_k + \boldsymbol{d}_k) - \nabla f(\boldsymbol{x}_k) - \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{d}_k \\
&= \int_0^1 \nabla^2 f(\boldsymbol{x}_k + t\boldsymbol{d}_k)\boldsymbol{d}_k dt - \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{d}_k \\
&= \int_0^1 \left[\nabla^2 f(\boldsymbol{x}_k + t\boldsymbol{d}_k) - \nabla^2 f(\boldsymbol{x}_k)\right] \boldsymbol{d}_k \, dt.
\end{aligned}$$

Thus, we obtain

$$\begin{aligned}
\|\nabla f(\boldsymbol{x}_{k+1})\|_2 &\leq \int_0^1 \|\nabla^2 f(\boldsymbol{x}_k + t\boldsymbol{d}_k) - \nabla^2 f(\boldsymbol{x}_k)\| \cdot \|\boldsymbol{d}_k\|_2 \, dt \\
&\leq \int_0^1 tQ\|\boldsymbol{d}_k\|_2^2 \, dt \\
&= \frac{Q}{2}\|[\nabla^2 f(\boldsymbol{x}_k)]^{-1}\nabla f(\boldsymbol{x}_k)\|_2^2 \\
&\leq \frac{Q}{2\mu^2}\|\nabla f(\boldsymbol{x}_k)\|_2^2,
\end{aligned}$$

where the second inequality follows from the Lipschitz assumption on the Hessian and the last inequality follows from the fact that the maximum eigenvalue of $(\nabla^2 f(\boldsymbol{x}_k))^{-2}$ is less than $1/\mu^2$. Thus we have

$$\frac{Q}{2\mu^2}\|\nabla f(\boldsymbol{x}_{k+1})\|_2 \;\leq\; \left(\frac{Q}{2\mu^2}\|\nabla f(\boldsymbol{x}_k)\|_2\right)^2 \;\leq\; \left(\frac{1}{2}\right)^2,$$

where the last inequality follows since $\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \mu^2/Q$. That is, at every iteration, we are **squaring** the error (which is less than $1/2$). If we entered this stage at iteration ℓ, this means

$$\frac{Q}{2\mu^2}\|\nabla f(\boldsymbol{x}_k)\|_2 \;\leq\; \left(\frac{Q}{2\mu^2}\|\nabla f(\boldsymbol{x}^{(\ell)})\|_2\right)^{2^{k-\ell}} \;\leq\; \left(\frac{1}{2}\right)^{2^{k-\ell}}.$$

Then using the strong convexity of f,

$$f(\boldsymbol{x}_k) - p^\star \;\leq\; \frac{1}{2\mu}\|\nabla f(\boldsymbol{x}_k)\|_2^2 \;\leq\; \frac{2\mu^3}{Q^2}\left(\frac{1}{2}\right)^{2^{k-\ell+1}}.$$

The right hand side above is less than ϵ when

$$k - \ell + 1 \geq \log_2 \log_2(\epsilon_0/\epsilon), \quad \epsilon_0 = 2m^3/L^2,$$

so we spend no more than $\log_2 \log_2(\epsilon_0/\epsilon)$ iterations in this phase.

Note that

$$\epsilon = 10^{-20}\epsilon_0 \quad \Rightarrow \quad \log_2 \log_2(\epsilon_0/\epsilon) = 6.0539.$$

Convergence criteria: the Newton decrement

We know that at the minimia of a smooth convex functional we will have $\nabla f(\boldsymbol{x}) = \mathbf{0}$. So a natural test for convergence is to measure how far away $\nabla f(\boldsymbol{x})$ is from $\mathbf{0}$; that is, we say we are converged when the norm of $\nabla f(\boldsymbol{x})$ is below some threshold (call it ϵ):

$$\text{stop when } \|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon.$$

Which norm should we use?

The natural instinct here is to go with the standard Euclidean (ℓ_2) norm, stopping when

$$\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \epsilon,$$

and in fact, this quantity played a key role in our analysis above. But there is something that is unsatisfying about using the Euclidean norm, and this problem also extends to the way we approached the analysis in the previous section. An interesting feature of Newton's method is that it is *affine invariant*; if we simply change the coordinates, the iterates change accordingly. For example, let $\boldsymbol{T}$ be a $N \times N$ invertible matrix, and set $\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{T}\boldsymbol{x})$. Suppose we run Newton's method to try to find a minima of f starting at $\boldsymbol{x}_0$ and computing iterates $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$. Then we run Newton's method on $\tilde{f}$ starting at $\boldsymbol{T}^{-1}\boldsymbol{x}_0$ and compute iterates $\widetilde{\boldsymbol{x}}_1, \widetilde{\boldsymbol{x}}_2, \ldots$. This second set of iterates will follow the same progression as the first under transformation by $\boldsymbol{T}$:

$$\tilde{\boldsymbol{x}}_k = \boldsymbol{T}^{-1}\boldsymbol{x}_k, \quad k = 1, 2, \ldots$$

The problem, then, with the the Euclidean norm of the gradient is that it is not affinely invariant:

$$\|\nabla \tilde{f}(\boldsymbol{x})\|_2 \neq \|\nabla f(\boldsymbol{T}\boldsymbol{x})\|_2 \quad \text{for general } \boldsymbol{T}.$$

(Apply the chain rule.)

A criteria that is affinely invariant is the Newton decrement:

$$\lambda(\boldsymbol{x}) = \sqrt{\boldsymbol{g}^{\mathrm{T}} \boldsymbol{H}^{-1} \boldsymbol{g}}, \quad \boldsymbol{g} = \nabla f(\boldsymbol{x}), \quad \boldsymbol{H} = \nabla^2 f(\boldsymbol{x}).$$

(Again, you can work this out with a little effort by applying the chain rule.) These are various ways you can interpret this: one is as size of the gradient in the norm induced by $\boldsymbol{H}^{-1}$:

$$\lambda(\boldsymbol{x}) = \|\nabla f(x)\|_{\boldsymbol{H}^{-1}}.$$

Of course, the norm itself depends on the point $\boldsymbol{x}$. You can also think of it as the directional derivative in the direction we are taking a Newton step; if $\boldsymbol{d} = -(\nabla^2 f(\boldsymbol{x}))^{-1} \nabla f(\boldsymbol{x})$, then

$$\langle \boldsymbol{d}, \nabla f(\boldsymbol{x}) \rangle = -\lambda(\boldsymbol{x})^2.$$

At any rate, the convergence criteria for Newton's method is usually whether $\lambda(\boldsymbol{x}_k)$ is below some threshold.

Self-concordant functions

There is an alternative analysis of Newton's method that is more satisfying in that it gives an affinely invariant bound, and it does not depend on the constants μ, L, Q that are usually unknown. The analysis holds for functions that are self-concordant, a term that we define below.

Definition. We say that a convex function of one variable $f : \mathbb{R} \to \mathbb{R}$ is *self-concordant* if

$$|f'''(x)| \leq 2f''(x)^{3/2}, \quad \text{for all} \quad x \in \operatorname{dom} f.$$

We say that a convex function of multiple variables $f : \mathbb{R}^N \to \mathbb{R}$ is self-concordant if

$$g(t) = f(\boldsymbol{x} + t\boldsymbol{v}) \text{ is self-concordant for all } \boldsymbol{x} \in \operatorname{dom} f, \ \boldsymbol{v} \in \mathbb{R}^N.$$

We should note that the constant 2 that appears in front of the $f''(x)$ above is somewhat arbitrary — if there is any uniform bound on the ratio of $|f'''(x)|$ to $f''(x)^{3/2}$, then f can be made self-concordant simply by re-scaling.

We mention a few important examples (see [BV04, Chapter 9.6] for many more).

- Since the third derivative of all linear and quadratic functionals is zero, they are self-concordant.
- $f(x) = -\log(x)$ is self-concordant
- $f(\boldsymbol{X}) = -\log\det \boldsymbol{X}$ for $\boldsymbol{X} \in S^N_{++}$ is self-concordant
- Self-concordance is preserved under composition with an affine

transformation, so for example

$$f(\boldsymbol{x}) = -\sum_{m=1}^{M} \log(b_m - \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x}) \quad \text{on } \{\boldsymbol{x} : \boldsymbol{a}_m^{\mathrm{T}} \leq b_m,\ m = 1, \ldots, M\}$$

is self-concordant. Functions of the above form will play a major role when we talk about log-barrier methods for contrained optimization.

Using a line of argumentation not too different than in the classical analysis in the last section, we have the following result for the convergence of Newton's method (again, see [BV04, Chapter 9] for the details):

If $f(\boldsymbol{x}) : \mathbb{R}^N \to \mathbb{R}$ is self-concordant, then Newton iterations starting from $\boldsymbol{x}_0$ coupled with standard backtracking line search will have

$$f(\boldsymbol{x}_k) - p^\star \leq \epsilon$$

when

$$k \geq C\epsilon_0 + \log_2 \log_2(1/\epsilon), \quad \epsilon_0 = f(\boldsymbol{x}_0) - p^\star.$$

The constant C above depends only on the backtracking parameters.

You may more fully appreciate this result when we talk about log barrier techniques a little later.

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

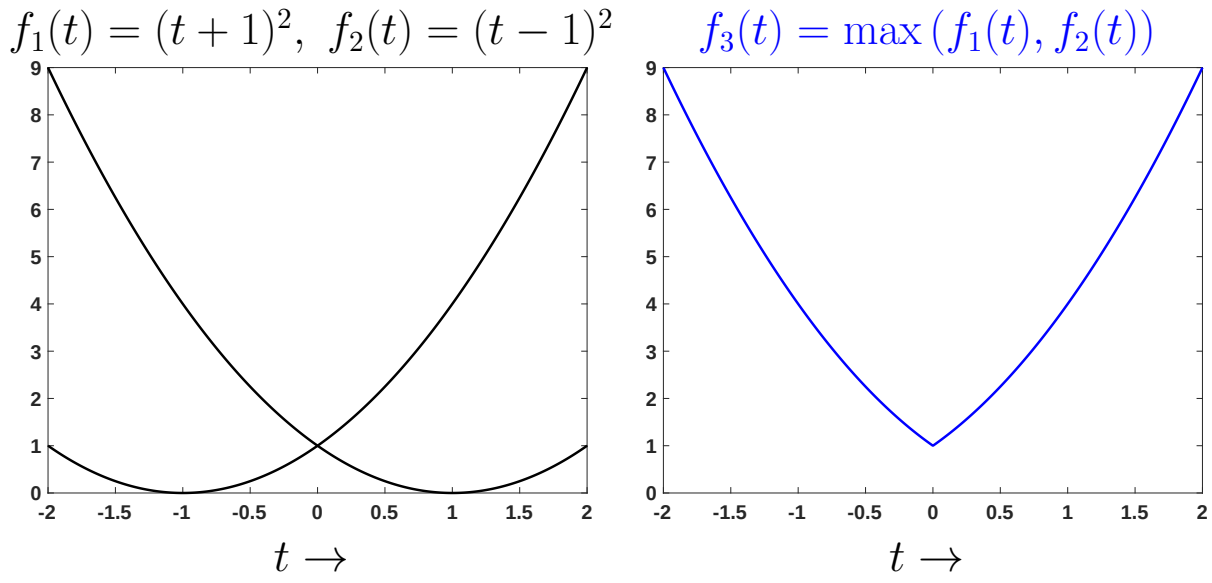
Nonsmooth optimization

Most of the theory and algorithms that we have explored for convex optimization have assumed that the functions involved are differentiable – that is, smooth.

This is not always the case in interesting applications. In fact, nonsmooth functions can arise quite naturally in applications. We already have looked at optimization programs involving the hinge loss $\max(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b, 0)$, the ℓ_1 norm, the ℓ_∞ norm, and the nuclear norm — none of these is differentiable. As another example, suppose $f_1, \ldots, f_Q$ are all perfectly smooth convex functions. Then the pointwise maximum

$$f(\boldsymbol{x}) = \max_{1 \le q \le Q} \; f_q(\boldsymbol{x})$$

is in general not smooth.

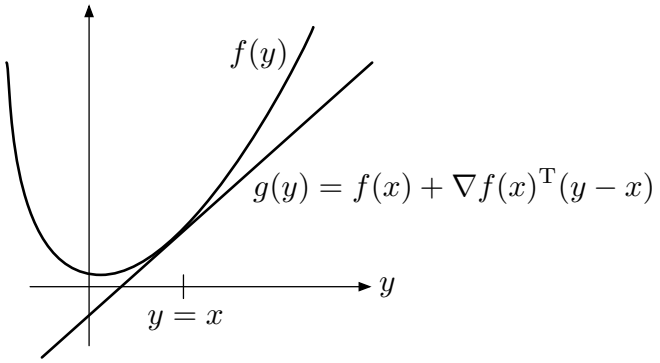


Fortunately, the theory for nonsmooth optimization is not too different than for smooth optimization. We really just need one new concept: that of a subgradient.

Subgradients

If you look back through the notes so far, you will see that the vast majority of the time we use the gradient of a convex function, it is in the context of the inequality

$$f(\boldsymbol{y}) \;\geq\; f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x}) \rangle, \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom} f.$$



This is a very special property of convex functions, and it led to all kinds of beautiful results.

When a convex f is not differentiable at a point $\boldsymbol{x}$, we can more or less reproduce the entire theory using subgradients. A **subgradient** of f at $\boldsymbol{x}$ is a vector $\boldsymbol{g}$ such that

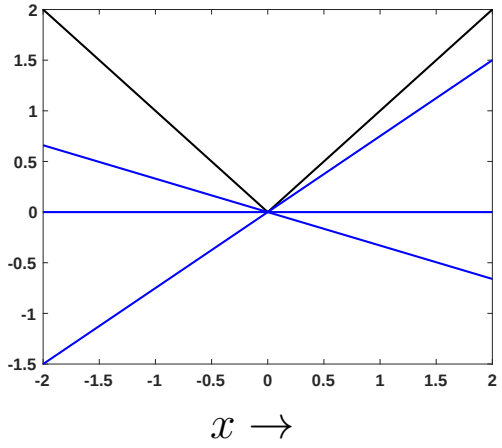
$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \boldsymbol{g} \rangle, \quad \text{for all } \boldsymbol{y} \in \operatorname{dom} f.$$

Unlike gradients for smooth functions, there can be more than one subgradient of a nonsmooth function at a point. We call the collection of subgradients the **subdifferential** at $\boldsymbol{x}$:

$$\partial f(\boldsymbol{x}) = \{\boldsymbol{g} \;:\; f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \boldsymbol{g} \rangle, \quad \text{for all } \boldsymbol{y} \in \operatorname{dom} f\}.$$

Example:

$$f(x) = |x|, \quad \partial f(x) = \begin{cases} -1, & x < 0 \\ [-1, 1], & x = 0 \\ 1, & x > 0. \end{cases}$$



black: $f(x) = |x|$
blue: $f(0) + g(x - 0)$ for a few $g \in \partial f(0)$

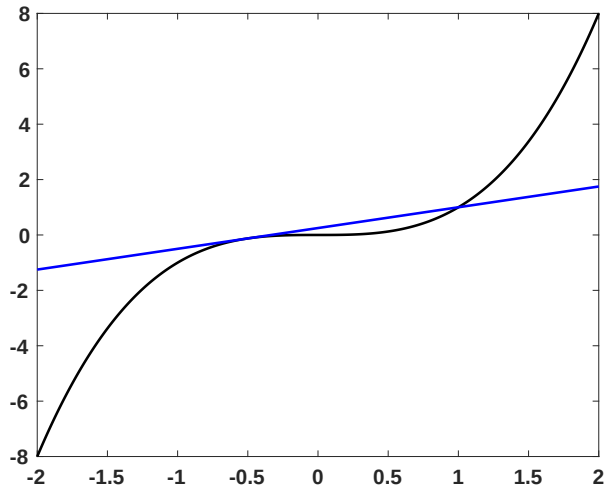
Facts for subdifferentials of convex functions:

1. If f is convex and differentiable at $\boldsymbol{x}$, then the subdifferential contains exactly one vector: the gradient,

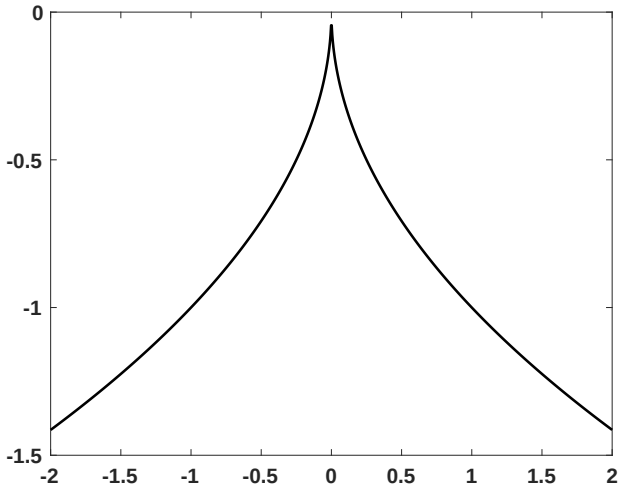
$$\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}.$$

2. If f is convex on $\operatorname{dom} f$, then the subdifferential is non-empty and bounded at all $\boldsymbol{x}$ in the interior of $\operatorname{dom} f$.

For non-convex f, these two points do not hold in general. The gradient at a point is not necessarily a subgradient:



and there can also be points where neither the gradient nor subgradient exist, e.g. $f(x) = -\sqrt{|x|}$ for $x \in \mathbb{R}$



Example: The ℓ_1 norm

Consider the function

$$f(\boldsymbol{x}) = \|\boldsymbol{x}\|_1.$$

The ℓ_1 norm is not differentiable at any $\boldsymbol{x}$ that has at least one coordinate equal to zero. We will see that optimization problems involving the ℓ_1 norm very often have solutions that are sparse, meaning that they have many zeros. This is a big problem – the nonsmoothness is kicking in at exactly the points we are interested in.

What does the subdifferential $\partial\|\boldsymbol{x}\|_1$ look like in such a case? To see, recall that by definition, if a vector $\boldsymbol{u} \in \partial\|\boldsymbol{x}\|_1$, at the point $\boldsymbol{x}$, then we must have

$$\|\boldsymbol{y}\|_1 \geq \|\boldsymbol{x}\|_1 + \langle \boldsymbol{y} - \boldsymbol{x}, \boldsymbol{u} \rangle \tag{1}$$

for all $\boldsymbol{y} \in \mathbb{R}^N$. To understand what this means in terms of $\boldsymbol{x}$, it is useful to introduce the notation $\Gamma(\boldsymbol{x})$ to denote the set of indexes where $\boldsymbol{x}$ is non-zero:

$$\Gamma(\boldsymbol{x}) = \{n \ : \ x_n \neq 0\}.$$

Using this, we can re-write the right-hand side of (1) as

$$\begin{aligned}
\|\boldsymbol{x}\|_1 + \langle \boldsymbol{y} - \boldsymbol{x}, \boldsymbol{u} \rangle &= \sum_{n=1}^{N} |x_n| + \sum_{n=1}^{N} u_n(y_n - x_n) \\
&= \sum_{n \in \Gamma} |x_n| - u_n x_n + \sum_{n=1}^{N} u_n y_n.
\end{aligned}$$

Note that if

$$u_n = \operatorname{sign}(x_n) = \begin{cases} 1 & \text{if } x_n \geq 0, \\ -1 & \text{if } x_n < 0, \end{cases}$$

then $u_n x_n = |x_n|$. Thus, if $u_n = \text{sign}(x_n)$ for all $n \in \Gamma$, we have

$$\sum_{n \in \Gamma} |x_n| - u_n x_n = \sum_{n \in \Gamma} |x_n| - |x_n| = 0.$$

Thus, if we set $u_n = \text{sign}(x_n)$ for all $n \in \Gamma$, then (1) reduces to

$$\|\boldsymbol{y}\|_1 \geq \langle \boldsymbol{y}, \boldsymbol{u} \rangle.$$

As long as $|u_n| \leq 1$ for all n, then this will hold. Hence, if a vector $\boldsymbol{u}$ satisfies

$$\begin{aligned} u_n &= \text{sign}(x_n) && \text{if } n \in \Gamma, \\ |u_n| &\leq 1 && \text{if } n \notin \Gamma, \end{aligned}$$

then $\boldsymbol{u} \in \partial \|\boldsymbol{x}\|_1$. It is not hard to show that for any $\boldsymbol{u}$ that violates these conditions, we can construct a $\boldsymbol{y}$ such that (1) is violated, and thus this is a complete description of all vectors in $\boldsymbol{u} \in \partial \|\boldsymbol{x}\|_1$.

Example: The ℓ_2 norm

While the function $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} = \|\boldsymbol{x}\|_2^2$ is the prototypical differentiable ($\nabla f(\boldsymbol{x}) = \boldsymbol{x}$), smooth, and strongly convex function ($\nabla^2 f(\boldsymbol{x}) = \mathbf{I}$), the function $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2$ is not as nice; it is not strongly convex, and it is not differentiable at $\boldsymbol{x} = \mathbf{0}$ (to appreciate this latter point, consider that a 1D slice of the function $s(t) = \|t\boldsymbol{v}\|_2 = |t|\|\boldsymbol{v}\|_2$ looks like the absolute value function as function of t).

For $\boldsymbol{x} \neq \mathbf{0}$, an easy calculation[1] shows that

$$\nabla \|\boldsymbol{x}\|_2 = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}.$$

At $\boldsymbol{x} = \mathbf{0}$, we know that $\boldsymbol{u} \in \partial\|\boldsymbol{x}\|_2$ if

$$\|\boldsymbol{y}\|_2 \geq \|\mathbf{0}\|_2 + \langle \boldsymbol{y} - \mathbf{0}, \boldsymbol{u} \rangle = \langle \boldsymbol{y}, \boldsymbol{u} \rangle \quad \text{for all} \quad \boldsymbol{y} \in \mathbb{R}^N. \tag{2}$$

We can find $\boldsymbol{u}$ that meet these conditions using the Cauchy-Schwarz inequality. Note that

$$\langle \boldsymbol{y}, \boldsymbol{u} \rangle \leq \|\boldsymbol{y}\|_2 \|\boldsymbol{u}\|_2,$$

so (2) will hold when $\|\boldsymbol{u}\|_2 \leq 1$. On the other hand, if $\|\boldsymbol{u}\|_2 > 1$, then for $\boldsymbol{y} = \boldsymbol{u}$, we have

$$\langle \boldsymbol{y}, \boldsymbol{u} \rangle = \|\boldsymbol{y}\|_2^2 > \|\boldsymbol{y}\|_2,$$

and (2) does not hold. Therefore

$$\partial\|\boldsymbol{x}\|_2 = \begin{cases} \{\boldsymbol{u} \ : \ \|\boldsymbol{u}\|_2 \leq 1\}, & \boldsymbol{x} = \mathbf{0} \\ \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}, & \boldsymbol{x} \neq \mathbf{0}. \end{cases}$$

[1] Use the fact that $\frac{d}{dx}\sqrt{x^2 + a} = x/\sqrt{x^2 + a}$.

General norms at $\boldsymbol{x} = \mathbf{0}$

Norms in general are not differentiable at $\boldsymbol{x} = \mathbf{0}$, again because they look like an absolute value function along a line: $s(t) = \|t\boldsymbol{v}\| = |t| \cdot \|\boldsymbol{v}\|$ for any valid norm $\|\cdot\|$. We can generalize the result for the ℓ_2 norm at $\boldsymbol{x} = \mathbf{0}$ using the concept of a **dual norm**.

The dual norm $\|\cdot\|_*$ of a norm $\|\cdot\|$ is

$$\|\boldsymbol{y}\|_* = \max_{\|\boldsymbol{x}\| \leq 1} \langle \boldsymbol{x}, \boldsymbol{y} \rangle.$$

Since sublevel sets of norms in $\mathbb{R}^N$ are compact, we know that this maximum is achieved, and it is an easy exercise to show that $\|\cdot\|_*$ is a valid norm. You can also verify the following easy facts at home

- the dual of $\|\cdot\|_2$ is again $\|\cdot\|_2$
- the dual of $\|\cdot\|_1$ is $\|\cdot\|_\infty$
- the dual of $\|\cdot\|_\infty$ is $\|\cdot\|_1$

It is also a fact (for norms on $\mathbb{R}^N$) that the dual of $\|\cdot\|_*$ is the original norm $\|\cdot\|$, i.e. $\|\boldsymbol{x}\|_{**} = \|\boldsymbol{x}\|$. We also have the generalized Cauchy-Schwarz inequality

$$|\langle \boldsymbol{x}, \boldsymbol{y} \rangle| \ \leq \ \|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|_*.$$

We can use these facts with an argument similar to the ℓ_2 case above to compute the subdifferential of any norm at $\mathbf{0}$ as

$$\partial \|\mathbf{0}\| = \{\boldsymbol{u} \ : \ \|\boldsymbol{u}\|_* \leq 1\}.$$

Properties of subdifferentials

Here are some properties of the subdifferential that we will state without proof (but are easy to prove). Below, we assume that all functions are well-defined on all of $\mathbb{R}^N$.

Summation: If $f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$, then

$$\partial f(\boldsymbol{x}) = \partial f_1(\boldsymbol{x}) + \partial f_2(\boldsymbol{x}).$$

That is, the set of all subgradients (at $\boldsymbol{x}$) of f is the set of vectors that can be written as a sum of a vector from $\partial f_1(\boldsymbol{x})$ plus a vector from $\partial f_2(\boldsymbol{x})$.

Chain rule for affine transformations: If $h(\boldsymbol{x}) = f(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b})$, then

$$\partial h(\boldsymbol{x}) = \boldsymbol{A}^{\mathrm{T}} \partial f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}).$$

That is, we compute the subgradients of f at the point $\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}$, then map them through $\boldsymbol{A}^{\mathrm{T}}$.

Max of functions: If $f(\boldsymbol{x}) = \max\{f_1(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x})\}$, then

$$\partial f(\boldsymbol{x}) = \operatorname{conv}\left(\bigcup_{m \in \Gamma(\boldsymbol{x})} \partial f_i(\boldsymbol{x}) \right),$$

where $\Gamma(\boldsymbol{x}) = \{m \ : \ f_m(\boldsymbol{x}) = f(\boldsymbol{x})\}$, and conv takes the convex hull:

$$\operatorname{conv}(\mathcal{X}) = \left\{ \sum_{p=1}^{P} \lambda_p \boldsymbol{x}_p, \ \boldsymbol{x}_p \in \mathcal{X}, \lambda_p \geq 0, \sum_{p=1}^{P} \lambda_p = 1, \ \forall P \right\}$$

Exercise: Compute $\partial f(\boldsymbol{x})$ for $f(\boldsymbol{x}) = \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_1$.

Answer: Set $\Gamma(\boldsymbol{x}) = \{m \ : \ \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} = y_m\}$, where $\boldsymbol{a}_m^{\mathrm{T}}$ is the mth row of $\boldsymbol{A}$. Then $\partial f(\boldsymbol{x})$ is the set of vectors that can be written as

$$\boldsymbol{u} = \sum_{m \notin \Gamma(\boldsymbol{x})} \mathrm{sgn}(\boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} - y_m)\boldsymbol{a}_m + \sum_{m \in \Gamma(\boldsymbol{x})} \beta_m \boldsymbol{a}_m$$

for any β_m with $|\beta_m| \leq 1$.

Exercise: Compute $\partial f(x)$ for $f(x) = \max(x, 0)$.

Answer:

$$\partial f(x) = \begin{cases} 0, & x < 0, \\ [0, 1], & x = 0, \\ 1, & x > 0. \end{cases}$$

Exercise: Compute $\partial f(x)$ for $f(x) = \max((x+1)^2, (x-1)^2)$.

Answer:

$$\partial f(x) = \begin{cases} 2(x-1) & x < 0, \\ [-2, 2], & x = 0, \\ 2(x+1), & x > 0. \end{cases}$$

Exercise: Compute $\partial f(\boldsymbol{x})$ for $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_\infty$.

Optimality conditions for unconstrained optimization

With the right definition in place, it is very easy to re-derive the central mathematical results in this course for general[2] convex functions.

Let $f(\boldsymbol{x})$ be a general convex function. Then $\boldsymbol{x}^\star$ is a solution to the unconstrained problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x})$$

if and only if

$$\mathbf{0} \in \partial f(\boldsymbol{x}^\star).$$

The proof of this statement is so easy you could do it in your sleep. Suppose $\mathbf{0} \in \partial f(\boldsymbol{x}^\star)$. Then

$$\begin{aligned} f(\boldsymbol{y}) &\geq f(\boldsymbol{x}^\star) + \langle \boldsymbol{y} - \boldsymbol{x}^\star, \mathbf{0} \rangle \\ &= f(\boldsymbol{x}^\star) \end{aligned}$$

for all $\boldsymbol{y} \in \operatorname{dom} f$. Thus $\boldsymbol{x}^\star$ is optimal. Likewise, if $f(\boldsymbol{y}) \geq f(\boldsymbol{x}^\star)$ for all $\boldsymbol{y} \in \operatorname{dom} f$, then of course it must also be true that $f(\boldsymbol{y}) \geq f(\boldsymbol{x}^\star) + \langle \boldsymbol{y} - \boldsymbol{x}, \mathbf{0} \rangle$ for all $\boldsymbol{y}$, and so $\mathbf{0} \in \partial f(\boldsymbol{x}^\star)$.

Example: The LASSO

Consider the ℓ_1 regularized least-squares problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \tau\|\boldsymbol{x}\|_1.$$

[2] Meaning not necessarily differentiable.

We can quickly translate the general result $\mathbf{0} \in \partial f(\boldsymbol{x}^\star)$ into a useful set of optimality conditions. We need to compute the subdifferential of $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \tau\|\boldsymbol{x}\|_1$. The first term is smooth, so the subdifferential just contains the gradient:

$$\partial f(\boldsymbol{x}) = \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) + \tau\partial\|\boldsymbol{x}\|_1.$$

As shown above $\partial\|\boldsymbol{x}\|_1$ is the set of all vectors $\boldsymbol{u}$ such that

$$\begin{aligned} u_n &= \operatorname{sign}(x_n) && \text{if } x_n \neq 0, \\ |u_n| &\leq 1 && \text{if } x_n = 0. \end{aligned}$$

Thus the optimality condition

$$\mathbf{0} \in \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}^\star - \boldsymbol{y}) + \tau\partial\|\boldsymbol{x}^\star\|_1,$$

means that $\boldsymbol{x}^\star$ is optimal if and only if

$$\begin{aligned} \boldsymbol{a}_n^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}^\star) &= \tau \operatorname{sign} x_n^\star && \text{if } x_n^\star \neq 0, \\ |\boldsymbol{a}_n^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}^\star)| &\leq \tau && \text{if } x_n^\star = 0. \end{aligned}$$

where here $\boldsymbol{a}_n$ is the n^{th} column of $\boldsymbol{A}$.

Note that this doesn't quite give us a closed form expression for $\boldsymbol{x}^\star$ (except when $\boldsymbol{A}$ is an orthonormal matrix), but it is useful both algorithmically (for checking if a candidate $\boldsymbol{x}$ is a solution) and theoretically (for understanding and analyzing the properties of the solution to this optimization problem.)

The subgradient method

The subgradient method is the non-smooth version of gradient descent. The basic algorithm is straightforward, consisting of the iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{d}_k, \tag{3}$$

where $\boldsymbol{d}_k$ is *any* subgradient at $\boldsymbol{x}_k$, i.e., $\boldsymbol{d}_k \in \partial f(\boldsymbol{x}_k)$. Of course, there could be many choices for $\boldsymbol{d}_k$ at every step, and the progress you make at that iteration could very dramatically with this choice. Making this determination, though, is often very difficult, and whether or not it can even be done it very problem dependent. Thus the analytical results for the subgradient method just assume we have any subgradient at a particular step.

With the right choice of step sizes $\{\alpha_k\}$, some simple analysis (which we will get to in a minute) shows that the subgradient method converges. The convergence rate, though, is very slow. This is also evidenced in most practical applications of this method: it can take many iterations on even a medium-sized problem to arrive at a solution that is even close to optimal.

Here is what we know about this algorithm for solving the general unconstrained program

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{x}). \tag{4}$$

We will look at one particular case here; for more detailed results see [Nes04, Chapter 3]. Along with f being convex, we will assume that it has at least one minimizer. The results also assume that f is Lipschitz:

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \;\leq\; L\|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

Note that here we are assuming that f is Lipschitz, not that f has Lipschitz gradients (since the gradient does not even necessarily exist). A direct consequence of f being Lipschitz is that the norms of the subgradients are bounded:

$$\|\boldsymbol{d}\|_2 \leq L, \quad \text{for all } \boldsymbol{d} \in \partial f(\boldsymbol{x}), \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^N. \tag{5}$$

The results below used pre-determined step sizes. Thus the iteration (3) does not necessarily decrease the functional $f(\boldsymbol{x})$ at every step. We will keep track of the best value we have up to the current iteration with

$$f_k^{\text{best}} = \min \left\{ f(\boldsymbol{x}_i), \quad 0 \leq i < k \right\}.$$

We will let $\boldsymbol{x}^\star$ be any solution to (4) and set $f^\star = f(\boldsymbol{x}^\star)$.

Our analytical results stem from a careful look at what happens during a single iteration. Note that

$$\begin{aligned} \|\boldsymbol{x}_{i+1} - \boldsymbol{x}^\star\|_2^2 &= \|\boldsymbol{x}_i - \alpha_i \boldsymbol{d}_i - \boldsymbol{x}^\star\|_2^2 \\ &= \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 - 2\alpha_i \langle \boldsymbol{x}_i - \boldsymbol{x}^\star, \boldsymbol{d}_i \rangle + \alpha_i^2 \|\boldsymbol{d}_i\|_2^2 \\ &\leq \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 - 2\alpha_i (f(\boldsymbol{x}_i) - f^\star) + \alpha_i^2 \|\boldsymbol{d}_i\|_2^2, \end{aligned}$$

where the inequality follows from the definition of a subgradient:

$$f^\star \geq f(\boldsymbol{x}_i) + \langle \boldsymbol{x}^\star - \boldsymbol{x}_i, \boldsymbol{d}_i \rangle.$$

Rearranging the bound above we have

$$2\alpha_i \left(f(\boldsymbol{x}_i) - f^\star \right) \leq \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{i+1} - \boldsymbol{x}^\star\|_2^2 + \alpha_i^2 \|\boldsymbol{d}_i\|_2^2,$$

and so of course

$$2\alpha_i \left(f_i^{\text{best}} - f^\star \right) \leq \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 + \alpha_i^2 \|\boldsymbol{d}_i\|_2^2.$$

Since f_i^{best} is monotonically decreasing, at iteration k we have

$$2\alpha_i \left(f_k^{\text{best}} - f^\star\right) \leq \|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{i+1} - \boldsymbol{x}^\star\|_2^2 + \alpha_i^2 \|\boldsymbol{d}_i\|_2^2,$$

for all $i \leq k$. To understand what has happened after k iterations, we sum both sides of the expression above from $i = 0$ to $i = k - 1$. Notice that the two error terms on the right hand side give us the telescoping sum:

$$\begin{aligned}\sum_{i=0}^{k-1} \left(\|\boldsymbol{x}_i - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{i+1} - \boldsymbol{x}^\star\|_2^2\right) &= \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 \\ &\leq \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2\end{aligned}$$

and so

$$f_k^{\text{best}} - f^\star \leq \frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2 + \sum_{i=0}^{k-1} \alpha_i^2 \|\boldsymbol{d}_i\|_2^2}{2\sum_{i=0}^{k-1} \alpha_i} \tag{6}$$

We can now specialize this result to general step-size strategies.

Fixed step size. Suppose that $\alpha_k = \alpha > 0$ for all k. Then (6) becomes

$$f_k^{\text{best}} - f^\star \leq \frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2}{2k\alpha} + \frac{L^2\alpha}{2},$$

where we have also used the Lipschitz property (5). Note that in this case, no matter how small we choose α, **the subgradient algorithm is not guaranteed to converge**. This is, in fact, standard in practice as well. The problem is that, unlike gradients for smooth functions, the subgradients do not have to vanish as we approach the solution. Even at the solution, there can be subgradients that are large.

Fixed step length. A similar result holds if we always move the same amount, taking

$$\alpha_k = s/\|\boldsymbol{d}_k\|_2.$$

This means that

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2 = s.$$

Of course, with this strategy it is self-evident that it will never converge, since we move some fixed amount at every step. We can bound the suboptimality at step k as

$$f_k^{\text{best}} - f^\star \leq \frac{L\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2}{2ks} + \frac{Ls}{2},$$

which is not necessarily worse than the fixed step size result. In fact, notice that even though you are moving some fixed amount, you will never move too far from an optimal point.

Decreasing step size. The results above suggest that we might want to decrease the step size as k increases, so we can get rid of this constant offset term. To make the terms in (6) work out, we let $\alpha_k \to 0$, but not too fast. Specifically, we choose a sequence $\{\alpha_k\}$ such that

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \text{and} \quad \frac{\sum_{i=0}^{k-1} \alpha_i^2}{\sum_{i=0}^{k-1} \alpha_i} \to 0.$$

Looking at (6) above, we can see that under these conditions $f_k^{\text{best}} \to f^\star$. It is an exercise (but a nontrivial one) to show that it is enough to choose $\{\alpha_k\}$ such that

$$\alpha_k \to 0 \text{ as } k \to \infty, \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k = \infty. \tag{7}$$

To get an idea of the tradeoffs involved here, suppose that $\alpha_k = \alpha/(k+1)$. Then for large k, we have the approximations

$$\sum_{i=0}^{k-1} \alpha_i \sim \alpha \log k, \quad \text{and} \quad \sum_{i=0}^{k-1} \alpha_i^2 \sim \text{Const} = \alpha^2 \pi^2/6$$

that are good as upper and lower bounds to within constants. In this case, the convergence result (6) becomes

$$f_k^{\text{best}} - f^\star \lesssim \frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2}{\alpha \log k} + \text{Const} \cdot \frac{\alpha L^2}{\log k}.$$

So the convergence is extraordinarily slow – *logarithmic* in k.

You can get much better rates than this (but still not great) by decreasing the stepsize more slowly. Consider now $\alpha_k = \alpha/\sqrt{k+1}$. Then for large k

$$\sum_{i=0}^{k-1} \alpha_i \sim (\alpha+1)\sqrt{k}, \quad \text{and} \quad \sum_{i=0}^{k-1} \alpha_i^2 \sim \alpha^2 \log k,$$

and so

$$f_k^{\text{best}} - f^\star \lesssim \frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2}{(\alpha+1)\sqrt{k}} + \text{Const} \cdot \frac{\alpha L^2 \log k}{\sqrt{k}}.$$

This is something like $O(1/\sqrt{k})$ convergence. This means that if we want to guarantee $f_k^{\text{best}} - f^\star \leq \epsilon$, we need $k = O(1/\epsilon^2)$ iterations.

In [Nes04, Chapter 3], it is shown that there is no better rate of convergence than $O(1/\sqrt{k})$ that holds uniformly across all problems.

Example. Consider the "ℓ_1 approximation problem"

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1.$$

We have already looked at the subdifferential of $\|\boldsymbol{x}\|_1$. Specifically, we showed that $\boldsymbol{u}$ is a subgradient of $\|\boldsymbol{x}\|_1$ at $\boldsymbol{x}$ if it satisfies

$$\begin{aligned} u_n &= \text{sign}(x_n) \quad \text{if } x_n \neq 0, \\ |u_n| &\leq 1 \qquad\quad \text{if } x_n = 0. \end{aligned}$$

In the exercise above, we also derived the subdifferential for $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1$. We quickly re-derive it here using "guess and check". First consider a vector $\boldsymbol{z}$ that satisfies

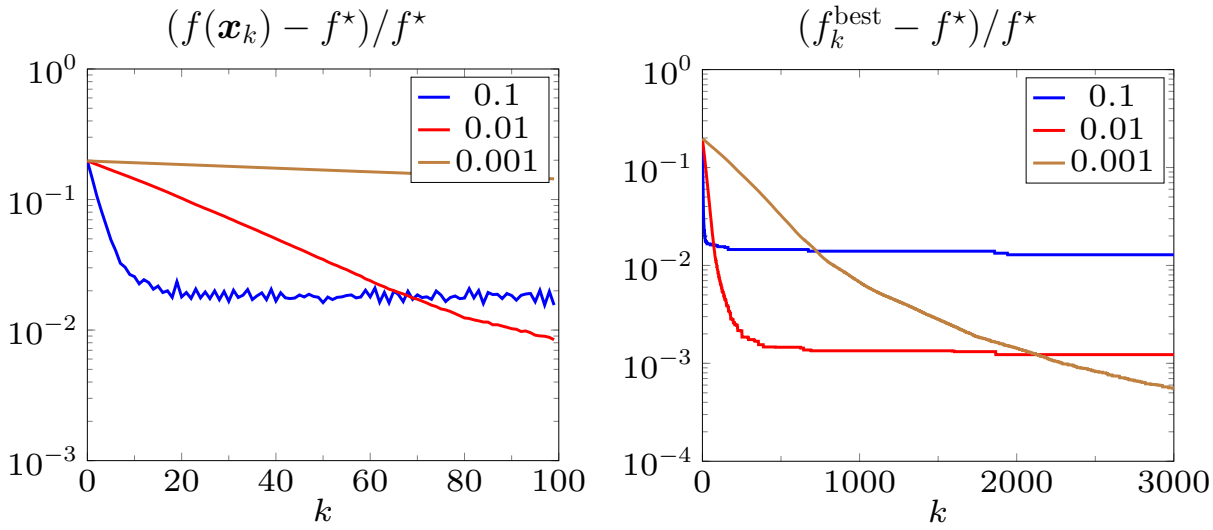
$$\begin{aligned} z_m &= \text{sign}(\boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} - b_m) \quad \text{if } \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} - b_m \neq 0, \\ |z_m| &\leq 1 \qquad\qquad\qquad\quad \text{if } \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x} - b_m = 0. \end{aligned}$$

Now consider the vector $\boldsymbol{u} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{z}$. Note that

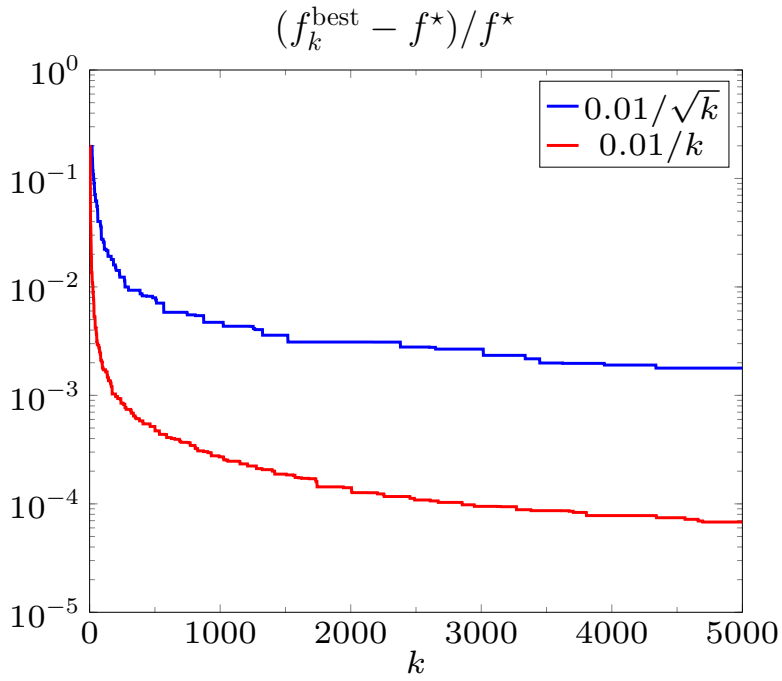
$$\begin{aligned} \boldsymbol{u}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) &= \boldsymbol{z}^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{y} - \boldsymbol{x}) \\ &= \boldsymbol{z}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b} + \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \\ &= \boldsymbol{z}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}) - \boldsymbol{z}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\ &= \boldsymbol{z}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}) - \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \\ &\leq \|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}\|_1 - \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1. \end{aligned}$$

Rearranging this shows that $\boldsymbol{u}$ is a subgradient of $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1$. Using this we can construct a subgradient at each step $\boldsymbol{x}_k$.

Below we illustrate the performance of this approach for a randomly generated example with $\boldsymbol{A} \in \mathbb{R}^{500\times 100}$ and $\boldsymbol{b} \in \mathbb{R}^{1000}$. For three different sizes of fixed step length, $s = 0.1, 0.01, 0.001$, we make quick progress at the beginning, but then saturate, just as the theory predicts:



Here is a run using two different decreasing step size strategies: $\alpha_k = .01/\sqrt{k}$ and $\alpha_k = .01/k$.



As you can see, even though the theoretical worst case bound makes a stepsize of $\sim 1/\sqrt{k}$ look better, in this particular case, a stepsize $\sim 1/k$ actually performs better.

Qualitatively, the takeaways for the subgradient method are:

1. It is a natural extension of the gradient descent formulation
2. In general, it does not converge for fixed stepsizes.
3. If the stepsizes decrease, you can guarantee convergence.
4. Theoretical convergence rates are slow.
5. Convergence rates in practice are also very slow, but depend a lot on the particular example.

References

[Nes04] Y. Nesterov. *Introductory Lectures on Convex Optimization*. Springer Science+Business Media, 2004.

Proximal algorithms

The subgradient algorithm is one generalization of gradient descent. It is simple, but the convergence is typically very slow (and it does not even converge in general for a fixed step size).

One way to deal with this is to add a smooth *regularization* term. Specifically, it is easy to show that if $\boldsymbol{x}^\star$ is a minimizer of $f(\boldsymbol{x})$, then it is also the minimizer of

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{x}) + \delta\|\boldsymbol{x}-\boldsymbol{x}^\star\|_2^2,$$

where $\delta > 0$. While the resulting optimization problem is still nonsmooth, it is now strongly convex, and we know that strongly convex functions are generally much easier to minimize. The "only" challenge is that it requires us to already know the solution $\boldsymbol{x}^\star$, which would seem to limit the practical applicability of this idea.

We can turn this into an actual algorithm by adopting an iterative approach. The **proximal algorithm** or **proximal point method** uses the following iteration:

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min}\left(f(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{x}_k\|_2^2\right). \tag{1}$$

As noted above, when f is convex, $f(\boldsymbol{x}) + \delta\|\boldsymbol{x}-\boldsymbol{z}\|_2^2$ is strictly convex for all $\delta > 0$ and $\boldsymbol{z}\in\mathbb{R}^N$, so the mapping from $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1}$ is well-defined. We will sometimes use the "prox operator" to denote this mapping:

$$\text{prox}_{\alpha_k f}(\boldsymbol{z}) = \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min}\left(f(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2\right).$$

It can be shown (and in fact we give a proof later in these notes) that the iterations above do find a minimizer of convex f for an appropriate choice of "step sizes" α_k.

At this point, you would be forgiven for having doubts about what we are really doing here. We have taken an optimization problem and turned it into... a sequence of *many* optimization problems. However, these problems can sometimes be far easier to solve that the original problem. One way to think about the additional $\frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{x}_k\|$ term is as a *regularizer* that makes each subproblem computationally easier to solve, and whose influence naturally disappears as we approach the solution, even for a fixed "step size" $\alpha_k = \alpha$.

A very nice a detailed review of proximal algorithms can be found in [PB14].

Implicit gradient descent ("backward Euler")

The proximal point method can also be interpreted as a variation on gradient descent. To see this, let us return for a moment to the differential equations for the "gradient flow" of f:

$$\boldsymbol{x}'(t) = -\nabla f(\boldsymbol{x}(t)), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0. \tag{2}$$

The equilibrium points for this system are the $\boldsymbol{x}$ such that $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$, which are precisely the minimizers for $f(\boldsymbol{x})$.

As we first discussed in the context of momentum-based methods, we can interpret gradient descent as a first-order numerical method for tracing the path from $\boldsymbol{x}_0$ to a solution $\boldsymbol{x}^\star$. This comes from discretizing the derivative on the right using a forward finite difference:

$$\frac{\boldsymbol{x}(t+h) - \boldsymbol{x}(t)}{h} \approx -\nabla f(\boldsymbol{x}(t)) \quad \text{for small } h.$$

Thus the gradient descent iterations

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h\nabla f(\boldsymbol{x}_k)$$

approximate the solution at equispaced times spaced h seconds apart — the step size in gradient descent can be interpreted as the time scale to which we are approximating the derivative. This is known as the *forward Euler method* for discretizing (2).

But now suppose we used a *backward difference* to approximate the derivative:

$$\frac{\boldsymbol{x}(t) - \boldsymbol{x}(t-h)}{h} \approx -\nabla f(\boldsymbol{x}(t)) \quad \text{for small } h.$$

Now the iterates must obey

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h\nabla f(\boldsymbol{x}_{k+1}).$$

This is an equally valid technique for discretizing (2) known as the *backward Euler method*. However, computing the iterates is not as straightforward – we can't just compute the gradient at the current point, we have to find the next point by finding an $\boldsymbol{x}_{k+1}$ that obeys the equation above.

This is exactly what the proximal operator does. If f is differentiable, then

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(f(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\ &\Updownarrow \\ \mathbf{0} &= \nabla f(\boldsymbol{x}_{k+1}) + \frac{1}{\alpha_k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k). \end{aligned} \tag{3}$$

So the proximal point method can be interpreted as a backward Euler discretization for gradient flow.

Note that we assumed the differentiability of f above purely for illustration; we can compute the prox operator whether or not f has a gradient.

Example: Least squares

Suppose we want to solve the standard least-squares 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, we know that the solution is given by $\widehat{\boldsymbol{x}}_{\text{ls}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$. However, we also know that when $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is not well-conditioned, this inverse can be unstable to compute, and iterative descent methods (gradient descent and conjugate gradients) can take many iterations to converge.

Consider the proximal point iteration (with fixed $\alpha_k = \alpha$) for solving this problem:

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min} \left(\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \frac{1}{2\alpha}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right).$$

Here we have the closed form solution

$$\begin{aligned} \boldsymbol{x}_{k+1} &= (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \delta\mathbf{I})^{-1}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \delta\boldsymbol{x}_k), \quad \delta = \frac{1}{\alpha} \\ &= \boldsymbol{x}_k + (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \delta\mathbf{I})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_k). \end{aligned}$$

Now each step is equivalent to solving a least-squares problem, but this problem can be made well-conditioned by choosing δ (i.e., α)

appropriately. The iterations above will converge to $\widehat{\boldsymbol{x}}_{\text{ls}}$ for any value of α; as we decrease α (increase δ), the number of iterations to get within a certain accuracy of $\widehat{\boldsymbol{x}}_{\text{ls}}$ increases, but the least-squares problems involved are all very well conditioned. For α very small, we are back at gradient descent (with step size α).

This is actually a well-known technique in numerical linear algebra called *iterative refinement*.

Proximal gradient algorithms

Recall the core update equation for the proximal point method:

$$\boldsymbol{x}_{k+1} = \text{prox}_{\alpha_k f}(\boldsymbol{x}_k) = \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \left(f(\boldsymbol{x}) + \frac{1}{2\alpha} \|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right).$$

Suppose that we did not wish to fully solve this problem at each iteration. If f is differentiable, we could approximate this update by replacing $f(\boldsymbol{x})$ with its linear approximation $f(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k) \rangle$. This would yield the update

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \left(f(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k) \rangle + \frac{1}{2\alpha_k} \|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\
&= \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \left(\frac{\alpha_k}{2} \|\nabla f(\boldsymbol{x}_k)\|_2^2 + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k) \rangle + \frac{1}{2\alpha_k} \|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\
&= \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \left(\frac{1}{2\alpha_k} \|\boldsymbol{x} - \boldsymbol{x}_k + \alpha_k \nabla f(\boldsymbol{x}_k)\|_2^2 \right) \\
&= \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k).
\end{aligned}$$

Thus, taking a linear approximation of f, the proximal method simply reduces to standard gradient descent. (Note that the first equality

above comes from the fact that the presence/absence of $f(\boldsymbol{x}_k)$ and $\|\nabla f(\boldsymbol{x}_k)\|_2^2$ does not affect what the minimizer is, as $\boldsymbol{x}_k$ is fixed.)

Where this starts getting interesting is when we encounter optimization problems where the objective function can be broken into the sum two parts, i.e.,

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x}),$$

where both g and h are convex, but g is smooth (differentiable) and h is a non-smooth function for which there is a fast proximal operator. Such optimization problems quite a bit more often than you might expect.

The **proximal gradient** algorithm is the result of applying the proximal point method to minimize the approximation of f where we take a linear approximation to the smooth component g. Using the same argument as above, this results in the update rule[1]

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(g(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla g(\boldsymbol{x}_k)\rangle + h(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\
&= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(h(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k + \alpha_k \nabla g(\boldsymbol{x}_k)\|_2^2 \right) \\
&= \text{prox}_{\alpha_k h}\left(\boldsymbol{x}_k - \alpha_k \nabla g(\boldsymbol{x}_k)\right).
\end{aligned}$$

This is also called *forward-backward splitting*, with the "forward" referring to the gradient step, and the "backward" to the proximal step. (The prox step is still making progress, just like the gradient step; the forward and backward refer to the interpretations of gradient descent and the proximal algorithm as forward and backward Euler discretizations, respectively.)

[1] Again, the second line comes from removing $g(\boldsymbol{x}_k)$ and adding a multiple of $\|\nabla g(\boldsymbol{x}_k)\|_2^2$ and then completing the square.

Example: The LASSO

Recall our friend the LASSO:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\tau\|\boldsymbol{x}\|_1.$$

We take

$$g(\boldsymbol{x})=\frac{1}{2}\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2, \quad \text{so} \quad \nabla g(\boldsymbol{x})=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}),$$

and

$$h(\boldsymbol{x})=\tau\|\boldsymbol{x}\|_1.$$

The prox operator for the ℓ_1 norm is:

$$\begin{aligned}\text{prox}_{\alpha h}(\boldsymbol{z}) &= \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min}\left(\tau\|\boldsymbol{x}\|_1+\frac{1}{2\alpha}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2\right)\\ &= T_{\tau\alpha}(\boldsymbol{z}),\end{aligned}$$

where $T_{\tau\alpha}$ is the soft-thresholding operator

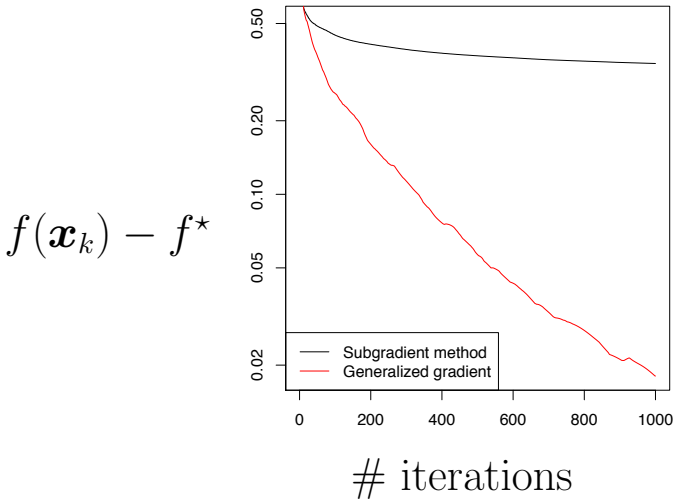
$$\left(T_{\tau\alpha}(\boldsymbol{z})\right)_i=\begin{cases}z_i-\tau\alpha, & z_i\geq\tau\alpha,\\ 0, & |z_i|\leq\tau\alpha,\\ z_i+\tau\alpha, & z_i\leq-\tau\alpha.\end{cases}$$

Hence, the gradient step requires an application of $\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}$, and the proximal step simply requires a soft-thresholding operation. The iteration looks like

$$\boldsymbol{x}_{k+1}=T_{\tau\alpha_k}\left(\boldsymbol{x}_k+\alpha_k\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}_k)\right).$$

This is also called the *iterative soft thresholding algorithm*, or ISTA.

Here is a comparison[2] of a typical run for ISTA versus the subgradient method. ISTA absolutely crushes the subgradient method.



[2]This is taken from the lecture notes of Geoff Gordon and Ryan Tibshirani; "generalized gradient" in the legend means ISTA.

Convergence of the proximal gradient method

The convergence analysis of the proximal gradient method is extremely similar to what we did for gradient descent. In fact, gradient descent is a special case of the proximal gradient method (when $h(\boldsymbol{x}) = 0$), and our analysis will recover the same result. We will assume that g is L-smooth, but we will make no assumptions on h aside from convexity. As before, we will use a fixed "step size", $\alpha_k = 1/L$ for all k. We will $\boldsymbol{x}^\star$ denote any minimizer of f.

The general structure of the argument is as follows:

1. Using the L-smoothness of g as well as the first-order characterization of convexity, we can establish that

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \boldsymbol{d}_k \rangle - \frac{1}{2L}\|\boldsymbol{d}_k\|_2^2 \tag{4}$$

   for all $\boldsymbol{z} \in \mathbb{R}^N$ where $\boldsymbol{d}_k := L(\boldsymbol{x}_k - \boldsymbol{x}_{k+1})$.

2. From (4) we can conclude, by setting $\boldsymbol{z} = \boldsymbol{x}_k$, that

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \frac{1}{2L}\|\boldsymbol{d}_k\|_2^2 \leq f(\boldsymbol{x}_k),$$

   and thus $f(\boldsymbol{x}_k)$ is non-increasing at every step.

3. From (4) we can also conclude, by setting $\boldsymbol{z} = \boldsymbol{x}^\star$, that

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}^\star) + \langle \boldsymbol{x}_k - \boldsymbol{x}^\star, \boldsymbol{d}_k \rangle - \frac{1}{2L}\|\boldsymbol{d}_k\|_2^2.$$

   By exactly the same argument as we have seen in the analysis of both gradient descent and Nesterov's method, we can show that this bound is equivalent to

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^\star) \leq \frac{L}{2}\left(\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2^2\right).$$

4. This yields a telescopic sum, and hence by an identical argument to that used in analyzing gradient descent, we arrive at the bound

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{L}{2k}\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Thus, the proximal gradient algorithm exhibits the same convergence rate as gradient descent: $O(1/k)$. This is remarkable when considering that it holds for *any* h. This result is in fact a kind of "master result" for the convergence rate of many different algorithms:

- gradient descent (take $h(\boldsymbol{x}) = 0$),
- the proximal point method (take $g(\boldsymbol{x}) = 0$),
- the proximal gradient method.

The work above gives a unified analysis for all three of these, showing that they all exhibit $O(1/k)$ convergence.

Note that the only novelty in the analysis above compared to that of gradient descent is the derivation of (4). To establish this inequality, we first note that

$$\begin{aligned} f(\boldsymbol{x}_{k+1}) &= g(\boldsymbol{x}_{k+1}) + h(\boldsymbol{x}_{k+1}) \\ &\leq g(\boldsymbol{x}_k) - \frac{1}{L}\langle \boldsymbol{d}_k, \nabla g(\boldsymbol{x}_k)\rangle + \frac{1}{2L}\|\boldsymbol{d}_k\|_2^2 + h(\boldsymbol{x}_{k+1}), \end{aligned} \tag{5}$$

where the inequality follows directly from the definition of L-smoothness. We now use two facts to get an upper bound on this expression. First, note that from the first-order characterization of convexity,

$$g(\boldsymbol{z}) \geq g(\boldsymbol{x}_k) + \langle \boldsymbol{z} - \boldsymbol{x}_k, \nabla g(\boldsymbol{x}_k)\rangle. \tag{6}$$

Second, since

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \text{prox}_{h/M}\left(\boldsymbol{x}_k - \frac{1}{L}\nabla g(\boldsymbol{x}_k)\right) \\ &= \arg\min_{\boldsymbol{x}}\left(h(\boldsymbol{x}) + \frac{L}{2}\left\|\boldsymbol{x} - \boldsymbol{x}_k + \frac{1}{L}\nabla g(\boldsymbol{x}_k)\right\|_2^2\right),\end{aligned}$$

we know

$$\mathbf{0} \in \partial h(\boldsymbol{x}_{k+1}) - \boldsymbol{d}_k + \nabla g(\boldsymbol{x}_k) \quad \Rightarrow \quad \boldsymbol{d}_k - \nabla g(\boldsymbol{x}_k) \in \partial h(\boldsymbol{x}_{k+1}).$$

Thus

$$h(\boldsymbol{z}) \geq h(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{z} - \boldsymbol{x}_{k+1}, \boldsymbol{d}_k - \nabla g(\boldsymbol{x}_k)\rangle. \tag{7}$$

We combine (6) and (7) back into (5) to obtain

$$\begin{aligned}f(\boldsymbol{x}_{k+1}) &\leq g(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \nabla g(\boldsymbol{x}_k)\rangle - \frac{1}{L}\langle \boldsymbol{d}_k, \nabla g(\boldsymbol{x}_k)\rangle + \frac{1}{2L}\|\boldsymbol{d}_k\|_2^2 \\ &\quad + h(\boldsymbol{z}) - \left\langle \boldsymbol{z} - \boldsymbol{x}_k + \frac{1}{L}\boldsymbol{d}_k, \boldsymbol{d}_k - \nabla g(\boldsymbol{x}_k)\right\rangle \\ &= f(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \boldsymbol{d}_k\rangle + \frac{1}{L}\left(\frac{L}{2L} - 1\right)\|\boldsymbol{d}_k\|_2^2 \\ &\leq f(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \boldsymbol{d}_k\rangle - \frac{1}{2L}\|\boldsymbol{d}_k\|_2^2,\end{aligned}$$

which establishes (4).

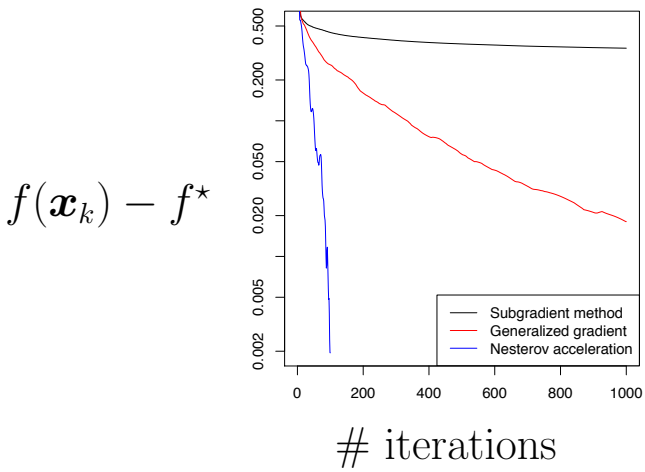
Accelerated proximal gradient

We can accelerate the proximal gradient method in exactly the same way we accelerated gradient descent – in fact, the Nesterov's method for gradient descent is simply a special case as that for the proximal gradient algorithm. The accelerated iteration is

$$\begin{aligned} \boldsymbol{p}_k &= \frac{k-1}{k+2}\left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right) \\ \boldsymbol{x}_{k+1} &= \operatorname{prox}_{\alpha_k h}\left(\boldsymbol{x}_k + \boldsymbol{p}_k - \alpha_k \nabla g(\boldsymbol{x}_k + \boldsymbol{p}_k)\right). \end{aligned}$$

Again, the computations here are in general no more involved than for the non-accelerated version, but the number of iterations can be significantly lower. We will not prove it here (see [BT09] for an analysis), but adding in the momentum term results in convergence rate of $O(1/k^2)$ using a similar argument as before.

The numerical performance can also be dramatically better. Here are typical runs[3] for the LASSO, which compares the standard proximal gradient method (ISTA) to its accelerated version (FISTA):



[3] Again, this example comes from Gordon and Tibshirani; as before "generalized gradient" means ISTA, and "Nesterov acceleration" means FISTA.

References

[BT09] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2(1):183–202, 2009.

[PB14] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2014.

III. Constrained Convex Optimization

Optimality conditions for constrained optimization

When we are solving an unconstrained optimization problem, the goal is clear: we want to find a point where the gradient vanishes. All of the algorithms we looked at over the last few lectures were in service of this condition. Once we add constraints, the optimality conditions are more complicated, and involve relationships between the gradient of the functional we are minimizing along with the gradients of the constraints — these are the so-called Karush-Kuhn-Tucker (KKT) conditions.

We will build up to the KKT conditions slowly. We will first derive a general (and very easy to prove) *geometric* necessary and sufficient condition for $\boldsymbol{x}^\star$ to be a minimizer of a constrained optimization program. We will then show how this simple result immediately yields the KKT conditions for certain kinds of constraints. In the next set of notes, we will derive the KKT conditions, show that they are always sufficient, and discuss conditions under which they are also necessary.

To keep things simpler, in our initial discussion of constrained optimization, we will restrict our focus to *smooth* optimization problems. As before, most of what we have to say can be extended to the non-smooth case by simply replacing gradients with subgradients, but we will assume that our objective function (and eventually, our constraints) are differentiable for the time being.

We start by considering the general constrained problem

$$\underset{\boldsymbol{x} \in \mathcal{C}}{\text{minimize}} \, f(\boldsymbol{x})$$

where $\mathcal{C}$ is a closed, convex set, and f is again a convex function.

We have the following fundamental result:

Let f be a differentiable convex function, and $\mathcal{C}$ be a closed convex set. Then $\boldsymbol{x}^\star$ is a minimizer of

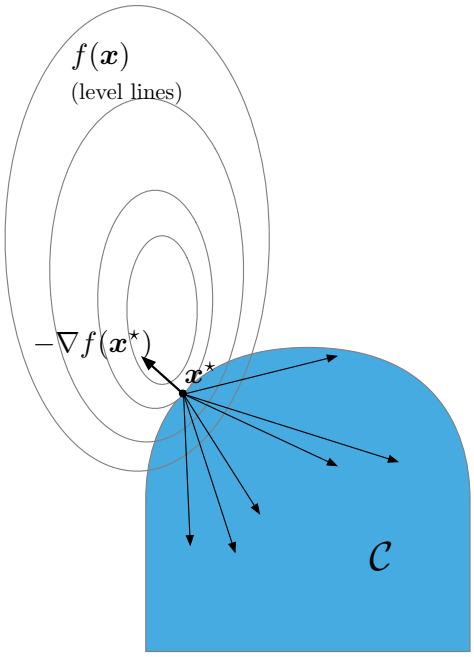
$$\underset{\boldsymbol{x}\in\mathcal{C}}{\text{minimize}}\, f(\boldsymbol{x})$$

if and only if $\boldsymbol{x}^\star \in \mathcal{C}$ and

$$\langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \;\geq\; 0$$

for all $\boldsymbol{y} \in \mathcal{C}$.

This result is geometrically intuitive; it is saying that every vector from $\boldsymbol{x}^\star$ to another point $\boldsymbol{y}$ in $\mathcal{C}$ must make an **obtuse** angle with $-\nabla f(\boldsymbol{x}^\star)$. That is, there cannot be any descent directions from $\boldsymbol{x}^\star$ that lead to another point in $\mathcal{C}$. Here is a picture:



To prove this, we first argue that $\langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0$ for all $\boldsymbol{y} \in \mathcal{C}$ implies that $\boldsymbol{x}^\star$ is optimal. Since f is convex, for any $\boldsymbol{y} \in \mathcal{C}$

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}^\star) + \langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle,$$

and so

$$f(\boldsymbol{y}) - f(\boldsymbol{x}^\star) \geq \langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \ \geq \ 0,$$

Since this holds for every $\boldsymbol{y} \in \mathcal{C}$, $\boldsymbol{x}^\star$ is a minimizer.

Now suppose that $\boldsymbol{x}^\star$ is a minimizer. If there were a $\boldsymbol{y} \in \mathcal{C}$ such that $\langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle < 0$, then $\boldsymbol{d} = \boldsymbol{y} - \boldsymbol{x}^\star$ would be a descent direction, and there would exist a $0 < t < 1$ such that

$$f(\boldsymbol{x}^\star + t(\boldsymbol{y} - \boldsymbol{x}^\star)) < f(\boldsymbol{x}^\star).$$

Since $\mathcal{C}$ is convex and $\boldsymbol{x}^\star, \boldsymbol{y} \in \mathcal{C}$, we know $\boldsymbol{x}^\star + t(\boldsymbol{y} - \boldsymbol{x}^\star) \in \mathcal{C}$. But this contradicts the assertion that $\boldsymbol{x}^\star$ is a minimizer, and so no such $\boldsymbol{y}$ can exist.

Examples

The abstract geometrical result in the previous section will eventually lead us to the Karush-Kuhn-Tucker (KKT) conditions. But we will build up to this by looking at what it tells us in several important (and prevalent) cases.

We assume throughout this section that f is convex, differentiable, and defined on all of $\mathbb{R}^N$.

Linear constraints

Consider a convex optimization problem with linear[1] constraints,

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

where $\boldsymbol{A}$ is $M \times N$ and $\boldsymbol{b} \in \mathbb{R}^M$. At a solution $\boldsymbol{x}^\star$, we have

$$\langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \;\; \geq \;\; 0,$$

for all $\boldsymbol{y}$ such that $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$. Since $\boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{b}$ as well, this is equivalent to

$$\langle \boldsymbol{h}, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0, \quad \text{for all} \;\; \boldsymbol{h} \in \text{Null}(\boldsymbol{A}).$$

Since $\boldsymbol{h} \in \text{Null}(\boldsymbol{A}) \Leftrightarrow -\boldsymbol{h} \in \text{Null}(\boldsymbol{A})$, we must have

$$\langle \boldsymbol{h}, \nabla f(\boldsymbol{x}^\star) \rangle = 0, \quad \text{for all} \;\; \boldsymbol{h} \in \text{Null}(\boldsymbol{A}),$$

i.e. the gradient is **orthogonal** to the null space of $\boldsymbol{A}$. This means that it is in the row space,

$$\nabla f(\boldsymbol{x}^\star) \in \text{Row}(\boldsymbol{A}) = \text{Col}(\boldsymbol{A}^{\text{T}}),$$

and so there is a $\boldsymbol{\nu} \in \mathbb{R}^M$ such that

$$\nabla f(\boldsymbol{x}^\star) + \boldsymbol{A}^{\text{T}}\boldsymbol{\nu} = \boldsymbol{0}.$$

[1] We really should be saying *affine constraints*, but "linear constraints" is typical nomenclature for this type of problem.

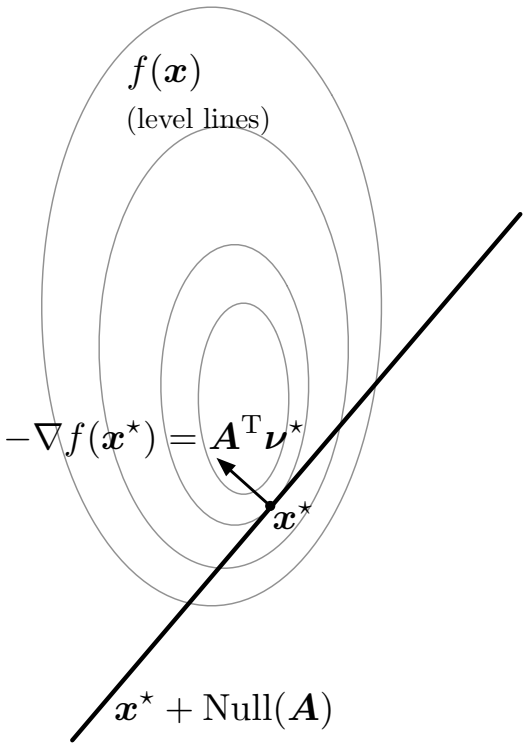
Summary:

$\boldsymbol{x}^\star$ is a solution to

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

if and only if

1. $\boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{b}$, and
2. there exists a $\boldsymbol{\nu}^\star \in \mathbb{R}^M$ such that $\nabla f(\boldsymbol{x}^\star) + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}^\star = \boldsymbol{0}$.



Non-negativity constraints

Now consider the convex program

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{x} \geq \mathbf{0}.$$

At a solution $\boldsymbol{x}^\star$, we will have

$$\langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0, \quad \text{for all} \;\; \boldsymbol{y} \in \mathbb{R}^N_+. \tag{1}$$

Since both $\mathbf{0} \in \mathbb{R}^N_+$ and $2\boldsymbol{x}^\star \in \mathbb{R}^N_+$, this means

$$\langle \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle = 0, \tag{2}$$

and so

$$\langle \boldsymbol{y}, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0, \quad \text{for all} \;\; \boldsymbol{y} \in \mathbb{R}^N_+,$$

meaning that the gradient has only non-negative values as well,

$$\nabla f(\boldsymbol{x}^\star) \geq \mathbf{0}. \tag{3}$$

The conditions (2) and (3) are sufficient as well, as together they imply (1).

Note that condition (3) is the same as saying there exists a $\boldsymbol{\lambda}^\star \geq \mathbf{0}$ such that

$$\nabla f(\boldsymbol{x}^\star) - \boldsymbol{\lambda}^\star = \mathbf{0}.$$

We can also see that (2) and (3), along with the fact that $\boldsymbol{x}^\star \in \mathbb{R}^N_+$, mean that $\nabla f(\boldsymbol{x}^\star)$ and $\boldsymbol{x}^\star$ can only be non-zero at different indices:

$$\begin{aligned} [\nabla f(\boldsymbol{x}^\star)]_n > 0 \Rightarrow \;\; & x_n = 0, \\ x_n > 0 \Rightarrow \;\; & [\nabla f(\boldsymbol{x}^\star)]_n = 0. \end{aligned}$$

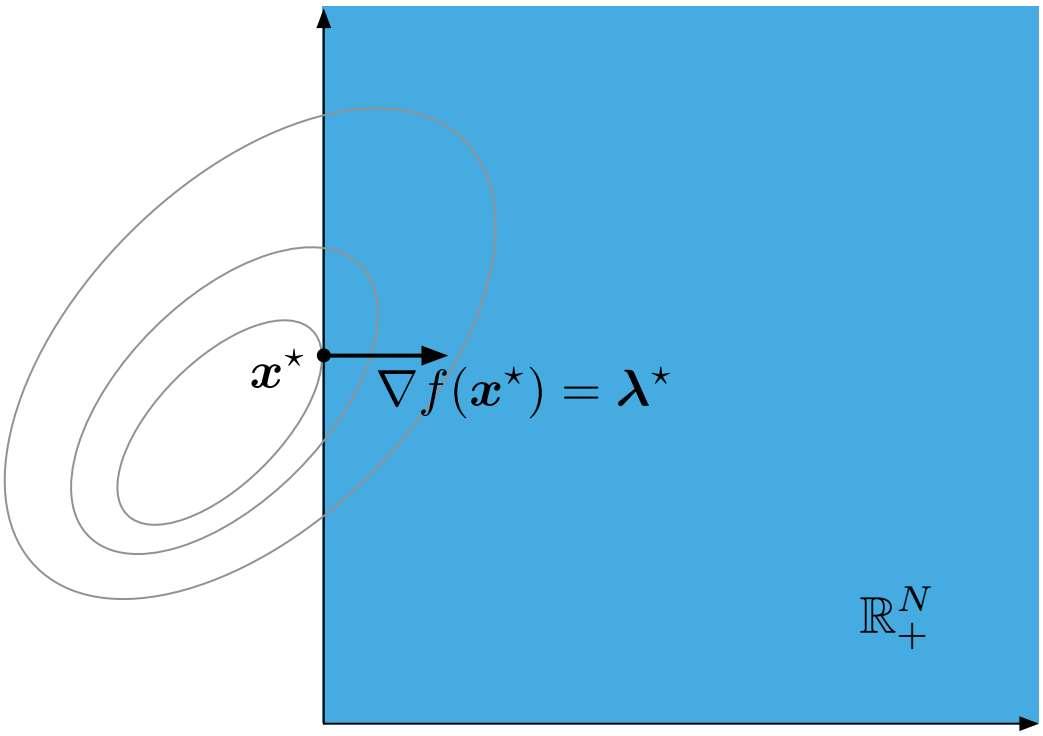
Summary:

$\boldsymbol{x}^\star$ is a solution to

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{x} \geq \mathbf{0},$$

if and only if

1. $\boldsymbol{x}^\star \geq \mathbf{0}$,
   and there exists a $\boldsymbol{\lambda}^\star \in \mathbb{R}^N$ such that
2. $\boldsymbol{\lambda}^\star \geq \mathbf{0}$, and
3. $\lambda_n x_n = 0$ for all $n = 1, \ldots, N$, and
4. $\nabla f(\boldsymbol{x}^\star) - \boldsymbol{\lambda}^\star = \mathbf{0}$.



A single convex inequality constraint

Now consider the convex program

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad g(\boldsymbol{x}) \leq 0,$$

where g is also a differentiable convex function. We will argue that in this case, the optimality conditions for $\boldsymbol{x}^\star$,

$$g(\boldsymbol{x}^\star) \leq 0, \quad \text{and} \quad \langle \boldsymbol{y} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0, \;\; \text{for all } \boldsymbol{y} \text{ with } g(\boldsymbol{y}) \leq 0,$$

are equivalent to one of these two conditions holding,

1. $g(\boldsymbol{x}^\star) < 0$ and $\nabla f(\boldsymbol{x}^\star) = \mathbf{0}$, or
2. $g(\boldsymbol{x}^\star) = 0$ and the gradients of g and f are negatively aligned:

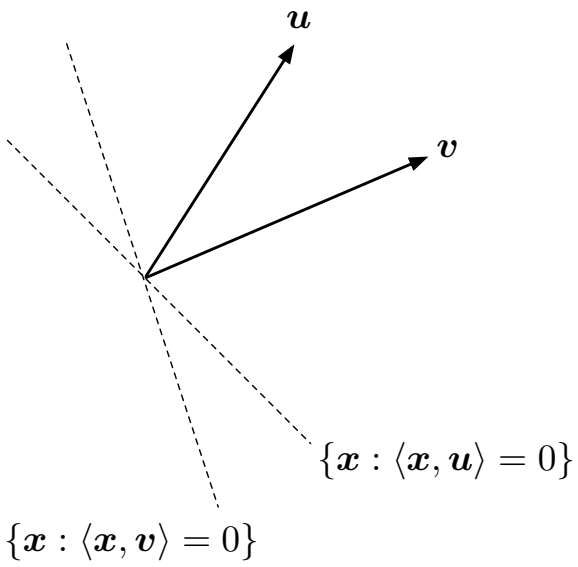
$$\nabla g(\boldsymbol{x}^\star) = -\lambda \nabla f(\boldsymbol{x}^\star), \quad \text{for some} \;\; \lambda > 0.$$

Establishing this relies on the following geometric fact:[2]

> Let $\boldsymbol{u}, \boldsymbol{v}$ be vectors in $\mathbb{R}^N$. If no $\boldsymbol{d}$ exists such that
>
> $$\langle \boldsymbol{d}, \boldsymbol{u} \rangle < 0, \quad \text{and} \quad \langle \boldsymbol{d}, \boldsymbol{v} \rangle < 0 \quad \text{simultaneously}, \tag{4}$$
>
> then $\boldsymbol{u}$ and $\boldsymbol{v}$ are negatively aligned,
>
> $$\boldsymbol{u} = -\lambda \boldsymbol{v}, \quad \text{for some} \;\; \lambda > 0. \tag{5}$$
>
> The converse also holds, as if (5) is true, there is no way (4) can be true.

[2] This is a special case of the famous Gordan Theorem.

The argument for this is simple. The sets $\{\boldsymbol{x} : \langle \boldsymbol{x}, \boldsymbol{u} \rangle < 0\}$ and $\{\boldsymbol{x} : \langle \boldsymbol{x}, \boldsymbol{v} \rangle < 0\}$ are open half spaces, and these half spaces are disjoint if and only if (5) holds.



Suppose that $\boldsymbol{x}^\star$, with $g(\boldsymbol{x}^\star) \leq 0$, is a minimizer. We know that $\nabla g(\boldsymbol{x}^\star)$ and $\nabla f(\boldsymbol{x}^\star)$ must be negatively aligned, as otherwise our geometric fact dictates that there is a $\boldsymbol{d}$ that is a descent direction for both g and f, meaning there is a $0 < t < 1$ such that

$$\begin{aligned} f(\boldsymbol{x}^\star + t\boldsymbol{d}) &< f(\boldsymbol{x}^\star), \text{ and} \\ g(\boldsymbol{x}^\star + t\boldsymbol{d}) &< g(\boldsymbol{x}^\star) \leq 0. \end{aligned}$$

This would mean that there is a feasible point at which f is smaller than it is at $\boldsymbol{x}^\star$, directly contradicting the assertion that $\boldsymbol{x}^\star$ is a minimizer. Thus no such $\boldsymbol{d}$ can exist.

Suppose now that there is an $\boldsymbol{x}^\star$ such that $g(\boldsymbol{x}^\star) = 0$ and a $\lambda > 0$ so that $\nabla g(\boldsymbol{x}^\star) = -\lambda \nabla f(\boldsymbol{x}^\star)$. Let $\boldsymbol{x}$ be any other feasible point; $g(\boldsymbol{x}) \leq 0$. Then, by the convexity of g,

$$g(\boldsymbol{x}^\star + \theta(\boldsymbol{x} - \boldsymbol{x}^\star)) \leq 0, \quad \text{for all } 0 \leq \theta \leq 1.$$

Since the above is true for *all* θ in this range, we know that $\boldsymbol{x} - \boldsymbol{x}^\star$ cannot be an ascent direction for g from $\boldsymbol{x}^\star$. Thus

$$\langle \boldsymbol{x} - \boldsymbol{x}^\star, \nabla g(\boldsymbol{x}^\star) \rangle \leq 0.$$

Since $\nabla g(\boldsymbol{x}^\star) = -\lambda \nabla f(\boldsymbol{x}^\star)$, we now know

$$\langle \boldsymbol{x} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0.$$

Then by the convexity of f,

$$\begin{aligned} f(\boldsymbol{x}) &\geq f(\boldsymbol{x}^\star) + \langle \boldsymbol{x} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \\ &\geq f(\boldsymbol{x}^\star), \end{aligned}$$

and so $\boldsymbol{x}^\star$ is a minimizer.

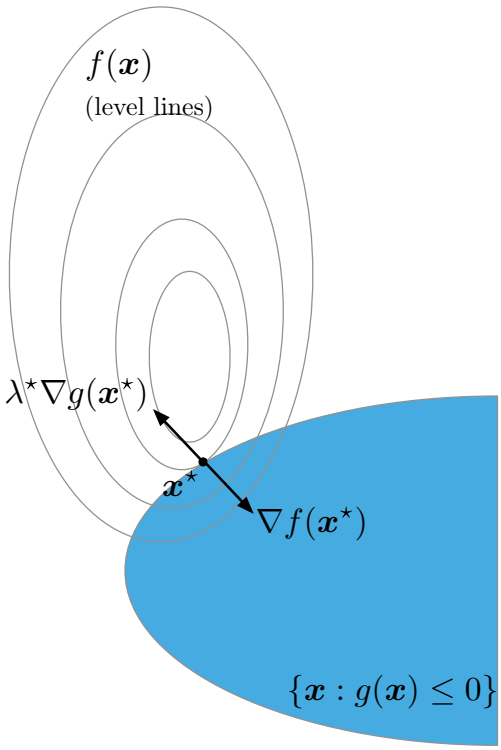
We can collect all of this into the following summary:

$\boldsymbol{x}^\star$ is a solution to

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad g(\boldsymbol{x}) \leq 0,$$

if and only if

1. $g(\boldsymbol{x}^\star) \leq 0$,
   and there exists a $\lambda^\star \in \mathbb{R}$ such that
2. $\lambda^\star \geq 0$, and
3. $\lambda^\star \, g(\boldsymbol{x}^\star) = 0$, and
4. $\nabla f(\boldsymbol{x}^\star) + \lambda^\star \nabla g(\boldsymbol{x}^\star) = \boldsymbol{0}$.


$f(\boldsymbol{x})$
(level lines)
$\lambda^\star \nabla g(\boldsymbol{x}^\star)$
$\boldsymbol{x}^\star$
$\nabla f(\boldsymbol{x}^\star)$
$\{\boldsymbol{x} : g(\boldsymbol{x}) \leq 0\}$

The Karush-Kuhn-Tucker (KKT) conditions

In this section, we will give a set of sufficient (and at most times necessary) conditions for a $\boldsymbol{x}^\star$ to be the solution of a given convex optimization problem. These are called the Karush-Kuhn-Tucker (KKT) conditions, and they play a fundamental role in both the theory and practice of convex optimization. We have derived these conditions (and have shown that they we both necessary and sufficient) in some special cases in the previous notes

We will start here by considering a general convex program with **inequality** constraints only. This is just to make the exposition easier — after we have this established, we will show how to include equality constraints (which must always be affine in convex programming). A great source for the material in this section is [Lau13, Chap. 10].

Everywhere in this section, the functions $f(\boldsymbol{x}), g_1(\boldsymbol{x}), \ldots, g_M(\boldsymbol{x})$, $g_m : \mathbb{R}^N \to \mathbb{R}$, are convex and differentiable.

KKT (inequality only)
The KKT conditions for the convex program

$$\underset{\boldsymbol{x}}{\text{minimize}} \ f(\boldsymbol{x}) \quad \text{subject to} \quad \begin{aligned} g_1(\boldsymbol{x}) &\leq 0 \\ g_2(\boldsymbol{x}) &\leq 0 \\ &\vdots \\ g_M(\boldsymbol{x}) &\leq 0 \end{aligned} \tag{1}$$

in $\boldsymbol{x} \in \mathbb{R}^N$ and $\boldsymbol{\lambda} \in \mathbb{R}^M$ are

$$g_m(\boldsymbol{x}) \leq 0, \quad m = 1, \ldots, M, \tag{K1}$$
$$\boldsymbol{\lambda} \geq \mathbf{0}, \tag{K2}$$
$$\lambda_m g_m(\boldsymbol{x}) = 0, \quad m = 1, \ldots, M, \tag{K3}$$
$$\nabla f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m \nabla g_m(\boldsymbol{x}) = \mathbf{0}, \tag{K4}$$

We start by establishing that these are sufficient conditions for a minimizer.

If the KKT conditions hold for $\boldsymbol{x}^\star$ and some $\lambda^\star \in \mathbb{R}^M$, then $\boldsymbol{x}^\star$ is a solution to the program (1).

Below, we denote the feasible set as

$$\mathcal{C} = \{\boldsymbol{x} \in \mathbb{R}^N \ : \ g_m(\boldsymbol{x}) \leq 0, \ m = 1, \ldots, M\}.$$

It should be clear that the convexity of the g_m implies the convexity[1]

[1]The g_m are convex functions, so their sublevel sets are convex sets, and $\mathcal{C}$

of $\mathcal{C}$. The sufficiency proof simply relies on the convexity of $\mathcal{C}$, the convexity of f, and the concept of a descent/ascent direction (see the previous notes).

Suppose $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star$ obey the KKT conditions. The first thing to note is that if

$$\lambda_1 = \lambda_2 = \cdots = \lambda_M = 0,$$

then (K4) implies that

$$\nabla f(\boldsymbol{x}^\star) = \mathbf{0},$$

and hence $\boldsymbol{x}^\star$ is a global min, as by the convexity of f,

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}^\star) + \langle \boldsymbol{x} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle = f(\boldsymbol{x}^\star),$$

for all $\boldsymbol{x} \in \mathcal{C}$.

Now suppose that $R > 0$ entries of $\boldsymbol{\lambda}^\star$ are positive — without loss of generality, we will take these to be the first R,

$$\lambda_1^\star > 0, \quad \lambda_2^\star > 0, \quad \cdots, \quad \lambda_R^\star > 0, \quad \lambda_{R+1}^\star = 0, \quad \cdots \quad, \lambda_M^\star = 0.$$

We can rewrite (K4) as

$$\nabla f(\boldsymbol{x}^\star) + \lambda_1^\star \nabla g_1(\boldsymbol{x}^\star) + \cdots + \lambda_R^\star \nabla g_R(\boldsymbol{x}^\star) = \mathbf{0}, \tag{2}$$

and note that by (K3),

$$g_1(\boldsymbol{x}^\star) = 0, \ldots, g_R(\boldsymbol{x}^\star) = 0.$$

Consider any $\boldsymbol{x} \in \mathcal{C}$, $\boldsymbol{x} \neq \boldsymbol{x}^\star$. As $\mathcal{C}$ is convex, every point in between $\boldsymbol{x}^\star$ and $\boldsymbol{x}$ must also be in $\mathcal{C}$, meaning

$$g_m(\boldsymbol{x}^\star + \theta(\boldsymbol{x} - \boldsymbol{x}^\star)) \leq 0 = g_m(\boldsymbol{x}^\star), \quad m = 1, \ldots, R,$$

is an intersection of sublevel sets.

for all $0 \leq \theta \leq 1$. This means that $\boldsymbol{x} - \boldsymbol{x}^\star$ cannot be an ascent direction, and so

$$\langle \boldsymbol{x} - \boldsymbol{x}^\star, \nabla g_m(\boldsymbol{x}^\star) \rangle \leq 0, \quad m = 1, \ldots, R.$$

It is now clear that

$$\langle \boldsymbol{x} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \geq 0,$$

as otherwise there is no way (2) can hold with positive λ_m. Along with the convexity of f, this means that

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}^\star) + \langle \boldsymbol{x} - \boldsymbol{x}^\star, \nabla f(\boldsymbol{x}^\star) \rangle \geq f(\boldsymbol{x}^\star).$$

Since this holds for all $\boldsymbol{x} \in \mathcal{C}$, $\boldsymbol{x}^\star$ is a minimizer.

Necessity

To establish the necessity of the KKT conditions, we need one piece of mathematical technology that we have not been exposed to yet. The *Farkas lemma* is a fundamental result in convex analysis; we will prove it in the Technical Details section.

Farkas Lemma:
Let $\boldsymbol{A}$ be an $M \times N$ matrix and $\boldsymbol{b} \in \mathbb{R}^M$. The exactly one of the following two things is true:

1. there exists $\boldsymbol{x} \geq \boldsymbol{0}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$;
2. there exists $\boldsymbol{\lambda} \in \mathbb{R}^M$ such that

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} \leq \boldsymbol{0}, \quad \text{and} \quad \langle \boldsymbol{b}, \boldsymbol{\lambda} \rangle > 0.$$

With this in place, we can give two different situations under which KKT is necessary. These are by no means the only situations for which this is true, but these two cover a high percentage of the cases encountered in practice.

Suppose $\boldsymbol{x}^\star$ is a solution to a convex program with affine inequality constraints:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}.$$

Then there exists a $\boldsymbol{\lambda}^\star$ such that $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star$ obey the KKT conditions.

In this case, the constraint functions have the form

$$g_m(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{a}_m \rangle - b_m, \quad \text{and so} \quad \nabla g_m(\boldsymbol{x}) = \boldsymbol{a}_m,$$

where $\boldsymbol{a}_m^{\mathrm{T}}$ is the mth row of $\boldsymbol{A}$. Since $\boldsymbol{x}^\star$ is feasible, K1 must hold. If none of the constraints are "active", meaning $g_m(\boldsymbol{x}^\star) < 0$ for $m = 1, \ldots, M$ (and so $\boldsymbol{x}^\star$ lies in the interior of $\mathcal{C}$), then it must be that $\nabla f(\boldsymbol{x}^\star) = \boldsymbol{0}$, and K2–K4 hold with $\boldsymbol{\lambda} = \boldsymbol{0}$.

Suppose that there are R active constraints at $\boldsymbol{x}^\star$; without loss of generality, we will take these to be the first R:

$$\begin{aligned} g_1(\boldsymbol{x}^\star) = 0 \;,\; g_2(\boldsymbol{x}^\star) = 0 \;,\; \ldots \;,\; g_R(\boldsymbol{x}^\star) = 0, \\ g_{R+1}(\boldsymbol{x}^\star) < 0 \;,\; \ldots,\; g_M(\boldsymbol{x}^\star) < 0. \end{aligned}$$

We start by taking $\lambda_{R+1} = \lambda_{R+2} = \cdots = \lambda_M = 0$, which means K3 will hold. Suppose that there were no $\boldsymbol{\lambda} \geq \boldsymbol{0}$ such that

$$\nabla f(\boldsymbol{x}^\star) + \lambda_1 \nabla g_1(\boldsymbol{x}^\star) + \cdots + \lambda_R \nabla g_R(\boldsymbol{x}^\star) = \boldsymbol{0}. \tag{3}$$

With $\boldsymbol{A}' : R \times N$ consisting of the first R rows of $\boldsymbol{A}$, and $\boldsymbol{b}' \in \mathbb{R}^R$ as the first R entries in $\boldsymbol{b}$, this means that there is no $\boldsymbol{\lambda}' \in \mathbb{R}^R$ such that

$$\boldsymbol{A}'^{\mathrm{T}}\boldsymbol{\lambda}' = -\nabla f(\boldsymbol{x}^\star), \quad \boldsymbol{\lambda}' \geq \boldsymbol{0}.$$

By the Farkas lemma, this means that there is a $\boldsymbol{d} \in \mathbb{R}^N$ such that

$$\boldsymbol{A}'\boldsymbol{d} \leq \boldsymbol{0}, \quad \langle \boldsymbol{d}, -\nabla f(\boldsymbol{x}^\star) \rangle > 0,$$

which means, since $\nabla g_m(\boldsymbol{x}) = \boldsymbol{a}_m$,

$$\begin{aligned} \langle \boldsymbol{d}, \nabla f(\boldsymbol{x}^\star) \rangle &< 0 \\ \langle \boldsymbol{d}, \nabla g_1(\boldsymbol{x}^\star) \rangle &\leq 0 \\ &\vdots \\ \langle \boldsymbol{d}, \nabla g_R(\boldsymbol{x}^\star) \rangle &\leq 0. \end{aligned}$$

This means that $\boldsymbol{d}$ is a descent direction for f, and is not an ascent direction for $g_1, \ldots, g_R$. Because the constraint functionals are affine, if $\langle \boldsymbol{d}, \nabla g_m(\boldsymbol{x}^\star) \rangle = 0$ above, then $g_m(\boldsymbol{x}^\star + t\boldsymbol{d}) = g_m(\boldsymbol{x}^\star)$ — this means that moving in the direction $\boldsymbol{d}$ will not increase $g_1, \ldots, g_m$. Since the last $M - R$ constraints are not active, we can move at least a small amount in any direction so that they stay that way. This means that there exists a $t > 0$ such that

$$f(\boldsymbol{x}^\star + t\boldsymbol{d}) < f(\boldsymbol{x}^\star),$$

but also maintains feasibility:

$$g_m(\boldsymbol{x}^\star + t\boldsymbol{d}) \leq 0, \quad m = 1, \ldots, M.$$

This directly contradicts the assertion that $\boldsymbol{x}^\star$ is optimal, and so $\lambda_1, \ldots, \lambda_R \geq 0$ must exist such that (3) holds.

For general convex inequality constraints, there are various other scenarios under which the KKT conditions are necessary; these are

called **constraint qualifications.** We have already seen that polygonal (affine) constraints qualify. Another set of constraint qualifications are *Slater's condition*:

Slater's condition: There exists at least one strictly feasible point; a $\boldsymbol{x}$ such that none of the constraints are active:

$$g_1(\boldsymbol{x}) < 0 \ , \ g_2(\boldsymbol{x}) < 0 \ , \ \cdots \ , \ g_M(\boldsymbol{x}) < 0.$$

Suppose that Slater's condition holds for $g_1, \ldots, g_M$, and let $\boldsymbol{x}^\star$ be a solution to

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ f(\boldsymbol{x}) \quad \text{subject to} \quad g_m \leq 0, \quad m = 1, \ldots, M.$$

Then there exists a $\boldsymbol{\lambda}^\star$ such that $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star$ obey the KKT conditions.

This is proved in much the same way as in the affine inequality case. Suppose that $\boldsymbol{x}^\star$ is a solution, and that

$$\begin{aligned} g_1(\boldsymbol{x}^\star) = 0 \ , \ g_2(\boldsymbol{x}^\star) = 0 \ , \ \ldots \ , \ g_R(\boldsymbol{x}^\star) = 0, \\ g_{R+1}(\boldsymbol{x}^\star) < 0 \ , \ \ldots, \ g_M(\boldsymbol{x}^\star) < 0. \end{aligned}$$

We take $\lambda_{R+1} = \cdots = \lambda_M = 0$, and show that if there is not $\lambda_1, \ldots, \lambda_R \geq 0$ such that

$$\nabla f(\boldsymbol{x}^\star) + \sum_{m=1}^{R} \lambda_m \nabla g_m(\boldsymbol{x}^\star) = \boldsymbol{0}, \tag{4}$$

then there is a another feasible point with a smaller value of f.

By the Farkas lemma, if there does not exist a $\lambda_1, \ldots, \lambda_R \geq 0$ such that (4) holds, then there must be a $\boldsymbol{u} \in \mathbb{R}^N$ such that

$$\begin{aligned} \langle \boldsymbol{u}, \nabla f(\boldsymbol{x}^\star) \rangle &< 0 \\ \langle \boldsymbol{u}, \nabla g_1(\boldsymbol{x}^\star) \rangle &\leq 0 \\ &\vdots \\ \langle \boldsymbol{u}, \nabla g_R(\boldsymbol{x}^\star) \rangle &\leq 0. \end{aligned}$$

Now let $\boldsymbol{z}$ be a strictly feasible point, $g_m(\boldsymbol{z}) < 0$ for all m. We know that

$$0 > g_m(\boldsymbol{z}) \geq g_m(\boldsymbol{x}^\star) + \langle \boldsymbol{z} - \boldsymbol{x}^\star, \nabla g_m(\boldsymbol{x}^\star) \rangle \quad \Rightarrow \quad \langle \boldsymbol{z} - \boldsymbol{x}^\star, \nabla g_m(\boldsymbol{x}^\star) \rangle < 0,$$

for $m = 1, \ldots, R$, since then $g_m(\boldsymbol{x}^\star) = 0$. So $\boldsymbol{u}$ is a descent direction for f_0, and $\boldsymbol{z} - \boldsymbol{x}^\star$ is a descent direction for all all of the constraint functions $g_m,\ m = 1, \ldots, R$ that are active.

We consider a convex combination of these two vectors

$$\boldsymbol{d}_\theta = (1 - \theta)\boldsymbol{u} + \theta(\boldsymbol{z} - \boldsymbol{x}^\star).$$

We know that $\langle \boldsymbol{d}_\theta, \nabla g_m(\boldsymbol{x}^\star) \rangle < 0$ for all $0 < \theta \leq 1,\ m = 1, \ldots, R$. We also know that there is a θ small enough so that $\boldsymbol{d}_\theta$ is a descent direction for f_0; there exists $0 < \epsilon_0 < 1$ such that

$$\langle \boldsymbol{d}_{\epsilon_0}, \nabla f(\boldsymbol{x}^\star) \rangle < 0.$$

Finally, we also know that we can move a small enough amount in any direction and keep constraints $g_{R+1}, \ldots, g_M$ inactive. Thus there is a $t > 0$ such that

$$f(\boldsymbol{x}^\star + t\boldsymbol{d}_{\epsilon_0}) < f(\boldsymbol{x}^\star), \quad g_m(\boldsymbol{x}^\star + t\boldsymbol{d}_{\epsilon_0}) \leq 0, \quad m = 1, \ldots, M,$$

which directly contradicts the assertion that $\boldsymbol{x}^\star$ is optimal.

It should be clear from the two arguments above that Slater's condition can be refined — we only need a point which obeys $g_m(\boldsymbol{z}) < 0$ for the g_m which are not affine. We now state this formally:

Suppose that $g_1, \ldots, g_{M'}$ are affine functionals, and $g_{M'+1}, \ldots, g_M$ are convex functional which are not affine. Suppose that Slater's condition holds for $g_{M'+1}, \ldots, g_M$, and let $\boldsymbol{x}^\star$ be a solution to

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad g_m(\boldsymbol{x}) \leq 0, \quad m = 1, \ldots, M.$$

Then there exists a $\boldsymbol{\lambda}^\star$ such that $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star$ obey the KKT conditions.

The above statement lets us extend the KKT conditions to optimization problems with linear equality constraints, which we now state.

KKT (with equality constraints)
The KKT conditions for the optimization program

$$\underset{\boldsymbol{x}}{\text{minimize}} \ f(\boldsymbol{x}) \quad \text{subject to} \quad \begin{aligned} g_m(\boldsymbol{x}) &\leq 0, \quad m = 1, \ldots, M \\ h_p(\boldsymbol{x}) &= 0, \quad p = 1, \ldots, P \end{aligned} \tag{5}$$

in $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{\lambda} \in \mathbb{R}^M$, and $\boldsymbol{\nu} \in \mathbb{R}^P$ are

$$\begin{aligned} g_m(\boldsymbol{x}) &\leq 0, \quad m = 1, \ldots, M, && \text{(K1)} \\ h_p(\boldsymbol{x}) &= 0, \quad p = 1, \ldots, P \\ \boldsymbol{\lambda} &\geq \mathbf{0}, && \text{(K2)} \\ \lambda_m g_m(\boldsymbol{x}) &= 0, \quad m = 1, \ldots, M, && \text{(K3)} \\ \nabla f_0(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m \nabla g_m(\boldsymbol{x}) + \sum_{p=1}^{P} \nu_p \nabla h_p(\boldsymbol{x}) &= \mathbf{0}, && \text{(K4)} \end{aligned}$$

We call the $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ above **Lagrange multipliers**. Notice that $\boldsymbol{\lambda}$ is constrained to be positive, while $\boldsymbol{\nu}$ can be arbitrary. Also, if the h_p are affine, which they have to be for the program above to be convex, then we can write the equality constraints

$$h_p(\boldsymbol{x}) = 0, \quad p = 1, \ldots, P \quad \text{as} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

for some $\boldsymbol{A} : P \times N$ and $\boldsymbol{b} \in \mathbb{R}^P$. Also, we can rewrite (K4) as

$$\nabla f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m \nabla g_m(\boldsymbol{x}) + \boldsymbol{A}^{\mathrm{T}} \boldsymbol{\nu} = \mathbf{0}.$$

If the g_m are convex and the h_p affine, then the KKT conditions are sufficient for $\boldsymbol{x}^\star$ to be the solution to the convex program (5).

If Slater's condition holds for the non-affine g_m, then they are also necessary. Almost nothing changes in the proofs above — we could simply separate an equality constraint of the form $\langle \boldsymbol{x}, \boldsymbol{a} \rangle = b$ into $\langle \boldsymbol{x}, \boldsymbol{a} \rangle - b \leq 0$ and $\langle \boldsymbol{x}, -\boldsymbol{a} \rangle + b \leq 0$. Then we can recombine the result, taking $\nu = \lambda_1 - \lambda_2$, where λ_1 is the Lagrange multiplier for $\langle \boldsymbol{x}, \boldsymbol{a} \rangle - b$ and λ_2 is the same for $\langle \boldsymbol{x}, -\boldsymbol{a} \rangle + b$.

Technical Details: Proof of the Farkas Lemma

We prove the Farkas Lemma: if $\boldsymbol{A}$ is an $M \times N$ matrix and $\boldsymbol{b} \in \mathbb{R}^M$ is a given vector, then exactly one of the following two things is true:

1. there exists $\boldsymbol{x} \geq \boldsymbol{0}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$;
2. there exists $\boldsymbol{v} \in \mathbb{R}^M$ such that

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v} \leq \boldsymbol{0}, \quad \text{and} \quad \langle \boldsymbol{b}, \boldsymbol{v} \rangle > 0.$$

It is clear that if the first condition holds, the second cannot, as $\langle \boldsymbol{b}, \boldsymbol{v} \rangle = \langle \boldsymbol{x}, \boldsymbol{A}^{\mathrm{T}}\boldsymbol{v} \rangle$ for any $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, and $\langle \boldsymbol{x}, \boldsymbol{A}^{\mathrm{T}}\boldsymbol{v} \rangle \leq 0$ for any $\boldsymbol{x} \geq \boldsymbol{0}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v} \leq \boldsymbol{0}$.

It is more difficult to argue that if the first condition does not hold, the second must. This ends up being a direct result of the separating hyperplane theorem. Let $\mathcal{C}(\boldsymbol{A})$ be the (convex) cone generated by the columns $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N$ of $\boldsymbol{A}$:

$$\mathcal{C}(\boldsymbol{A}) = \left\{ \boldsymbol{v} \in \mathbb{R}^M \ : \ \boldsymbol{v} = \sum_{n=1}^{N} \theta_n \boldsymbol{a}_n, \ \ \theta_n \geq 0, \ n = 1, \ldots, N \right\}.$$

Then 1 above is clearly equivalent to $\boldsymbol{b} \in \mathcal{C}(\boldsymbol{A})$. Since $\mathcal{C}(\boldsymbol{A})$ is closed and convex, and $\boldsymbol{b}$ is a single point, we know that if $\boldsymbol{b} \notin \mathcal{C}(\boldsymbol{A})$, then $\mathcal{C}(\boldsymbol{A})$ and $\boldsymbol{b}$ are strongly separated by a hyperplane. That is, if $\boldsymbol{b} \notin \mathcal{C}(\boldsymbol{A})$ implies that there exists a $\boldsymbol{v} \in \mathbb{R}^M$ such that

$$\boldsymbol{v}^{\mathrm{T}}\boldsymbol{b} > \boldsymbol{v}^{\mathrm{T}}\boldsymbol{\lambda} \quad \text{for all} \quad \boldsymbol{\lambda} \in \mathcal{C}(\boldsymbol{A}),$$

which is the same as saying

$$\boldsymbol{v}^{\mathrm{T}}\boldsymbol{b} > \sup_{\boldsymbol{\lambda} \in \mathcal{C}(\boldsymbol{A})} \boldsymbol{v}^{\mathrm{T}}\boldsymbol{\lambda} = \sup_{\boldsymbol{x} \geq \boldsymbol{0}} \boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}.$$

We know that $\mathbf{0} \in \mathcal{C}(\boldsymbol{A})$, so we must have $\boldsymbol{v}^{\mathrm{T}}\boldsymbol{b} > 0$. The above equation also gives a finite upper bound (namely whatever the actual value of $\boldsymbol{v}^{\mathrm{T}}\boldsymbol{b}$ is) on the function $\boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$ for all $\boldsymbol{x} \geq \mathbf{0}$. But this means that $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v} \leq \mathbf{0}$, as otherwise we would have the following contradiction. If there were some index n such that $(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v})[n] = \epsilon > 0$, then with $\boldsymbol{e}_m \geq \mathbf{0}$ as the unit vector

$$\boldsymbol{e}_n[k] = \begin{cases} 1, & k = n, \\ 0, & k \neq n \end{cases},$$

we have

$$\sup_{\boldsymbol{x} \geq \mathbf{0}} \boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \geq \sup_{\alpha \geq 0} \boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}(\alpha \boldsymbol{e}_n) = \sup_{\alpha \geq 0} \alpha\epsilon = \infty,$$

which contradicts the existence of this upper bound.

References

[Lau13] N. Lauritzen. *Undergraduate Convexity*. World Scientific, 2013.

Lagrange duality

In the previous lecture, we derived the KKT conditions for minimizing a convex function under convex inequality constraints and/or affine equality constraints. This involved introducing additional variables $\boldsymbol{\nu}$ and $\boldsymbol{\lambda}$. Here we will provide an alternative perspective on these problems and provide a bit more intuition as to how to interpret these additional variables.

The Lagrangian

We again consider an optimization program of the form

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; & f(\boldsymbol{x}) \\ \text{subject to} \;\; & g_m(\boldsymbol{x}) \leq 0, \quad m = 1, \ldots, M \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{1}$$

We will focus on the case where the objective function f and the inequality constraints g_m are convex, and the equality constraints are affine (note that for equality constraints, convexity is equivalent to being affine). However, in general much of what we have to say applies to arbitrary (nonconvex) problems as well so we will be clear when we are or are not assuming convexity. We will take the domain of all of the g_m to be all of $\mathbb{R}^N$ below; this just simplifies the exposition, we can easily replace this with the intersections of the dom g_m. We will also assume that the feasible set

$$\mathcal{C} = \{\boldsymbol{x} \;:\; g_m(\boldsymbol{x}) \leq 0 \; m = 1, \ldots, M, \; \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$$

is non-empty and a subset $\mathbb{R}^N$.

The **Lagrangian** takes the constraints in the program above and integrates them into the objective function. Specifically, the Lagrangian associated with (1) is

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m g_m(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}).$$

For reasons that will become clearer below, the $\boldsymbol{x}$ above are referred to as **primal variables**, and the $\boldsymbol{\lambda}, \boldsymbol{\nu}$ as either **dual variables** or **Lagrange multipliers**.

The Lagrangian allows us to transform the *constrained* optimization problem in (1) into an *unconstrained* one. Specifically, suppose for the moment that we are interested in a problem of the form in (1) but without equality constraints. Consider the problem given by

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m g_m(\boldsymbol{x}). \tag{2}$$

To get some intuition, suppose that we set the $\lambda_1, \ldots, \lambda_M$ to be very large (positive) numbers. In this case, violating any of the constraints (allowing $g_m(\boldsymbol{x}) > 0$) will result in a very large penalty being added to the objective function, so that by setting the corresponding λ_m to be large we will eventually guarantee that the resulting solution will satisfy the desired constraints.

The problem here is that large values of λ_m not only avoid the setting where $g_m(\boldsymbol{x}) > 0$, but actually encourages $g_m(\boldsymbol{x}) \ll 0$ (since we can potentially benefit by not just satisfying the constraints but by exceeding them by a large margin).

This raises a natural question: can we set $\boldsymbol{\lambda}$ so that the solution to the unconstrained problem (2) is the same as the constrained problem (1)? Here we will provide an answer in the case where the objective function f and the constraints $g_1, \ldots, g_M$ are both convex and differentiable.

Suppose that $\boldsymbol{x}^\star$ is a solution to the constrained problem (1). If we want $\boldsymbol{x}^\star$ to be a solution to (2), then a necessary and sufficient condition is

$$\nabla \mathcal{L}(\boldsymbol{x}^\star, \boldsymbol{\lambda}) = \nabla f(\boldsymbol{x}^\star) + \sum_{m=1}^{M} \lambda_m \nabla g_m(\boldsymbol{x}^\star) = \mathbf{0}. \tag{3}$$

At this point you might want to compare (3) with condition (K4) from the second two examples from the previous lecture. (Hint: they are the same!)

If we knew $\boldsymbol{x}^\star$ already, finding a $\boldsymbol{\lambda}$ that would make the unconstrained and constrained problems equivalent (meaning that they both have the same solution $\boldsymbol{x}^\star$) would just amount to finding a $\boldsymbol{\lambda}$ such that (3) holds. Unfortunately, this might not seem to be particularly useful since $\boldsymbol{x}^\star$ is what we are trying to find to begin with.

To see how we might compute a $\boldsymbol{\lambda}$ that makes the unconstrained and constrained problems equivalent, we will need to begin our first exploration of one of the deepest and most important ideas of optimization: **duality**.

The Lagrange dual function

We can think of the unconstrained optimization problem (2) as actually representing a family of different optimization problems (depending on $\boldsymbol{\lambda}$). For any fixed $\boldsymbol{\lambda}$, imagine solving (2) and computing the minimal value of the objective function – we can think of this as actually defining a function that maps $\boldsymbol{\lambda} \in \mathbb{R}^M$ to $\mathbb{R}$. Specifically, returning to the case where we have both inequality and equality constraints, the **Lagrange dual function** $d(\boldsymbol{\lambda}, \boldsymbol{\nu})$ is the minimum[1] of the Lagrangian over all $\boldsymbol{x} \in \mathbb{R}^N$:

$$d(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\boldsymbol{x} \in \mathbb{R}^N} \left(f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m g_m(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \right).$$

Note that since the dual is the pointwise infimum of a family of affine functions in $\boldsymbol{\lambda}, \boldsymbol{\nu}$, the Lagrange dual function is **always concave**, regardless of whether or not f, g_m, and equality constraints are convex. While we will not stress this much here, this is a remarkable fact and can be very useful when dealing with nonconvex problems.

A key fact about the dual function is that it can provide a lower bound on the optimal value of the original program. In the discussion below, we assume throughout that $\boldsymbol{\nu}$ and $\boldsymbol{\lambda} \geq 0$ are arbitrary. Our main claim is that if $p^\star = f(\boldsymbol{x}^\star)$ is the optimal value for (1),[2] then we have

$$d(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^\star.$$

This is very easy to show. Specifically, for any feasible point $\boldsymbol{x}'$, we

[1] We are writing inf instead of min here since we in general cannot be sure that the minimum exists. It very well may be that $d(\boldsymbol{\lambda}, \boldsymbol{\nu})$ is $-\infty$.

[2] We use $p^\star$ instead of $f^\star$ to indicate the optimal value of the *primal* problem, which we will soon be opposing to the optimal value of the *dual* problem.

must have $g_m(\boldsymbol{x}') \leq 0$ for all m and also $\boldsymbol{A}\boldsymbol{x}' = \boldsymbol{b}$, and hence

$$\sum_{m=1}^{M} \lambda_m g_m(\boldsymbol{x}') + \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}' - \boldsymbol{b}) \leq 0.$$

From this we have that

$$\mathcal{L}(\boldsymbol{x}', \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f(\boldsymbol{x}'),$$

meaning that

$$d(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \min_{\boldsymbol{x} \in \mathbb{R}^N} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq \mathcal{L}(\boldsymbol{x}', \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f(\boldsymbol{x}').$$

Since this holds for all feasible $\boldsymbol{x}'$, including the minimizer of (1), we have $d(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^\star$.

The (Lagrange) dual problem

Given that $d(\boldsymbol{\lambda}, \boldsymbol{\nu})$ provides a lower bound on $p^\star$, if you wanted to get an idea of what $p^\star$ looks like (for example, to see if you are close to convergence), it is natural to see how large you can make this lower bound. This gives rise to what we call the **(Lagrange) dual problem** of (1):

$$\underset{\boldsymbol{\lambda}, \boldsymbol{\nu}}{\text{maximize}} \;\; d(\boldsymbol{\lambda}, \boldsymbol{\nu}) \quad \text{subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{4}$$

The dual optimal value $d^\star$ is

$$d^\star = \sup_{\boldsymbol{\lambda} \geq \boldsymbol{0}, \boldsymbol{\nu}} \; d(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \sup_{\boldsymbol{\lambda} \geq \boldsymbol{0}, \boldsymbol{\nu}} \; \inf_{\boldsymbol{x} \in \mathbb{R}^N} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}).$$

Since $d(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^\star$, we know that

$$d^\star \leq p^\star.$$

The quantity $p^\star - d^\star$ is called the **duality gap**. If $p^\star = d^\star$, then we say that (1) and (4) exhibit **strong duality**.

We will soon discuss when strong duality holds, but first, why is it important? Suppose that $\boldsymbol{x}^\star$ is a solution to the original constrained problem (1) – which we will call the **primal problem** to distinguish it from the dual problem – and suppose that $(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ is a solution to the dual problem (4). It turns out that if we have strong duality, then $(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ is exactly what we need to make $\boldsymbol{x}^\star$ the solution to the unconstrained problem (2).

To see why, note that if we have strong duality then

$$\begin{aligned} f(\boldsymbol{x}^\star) &= d(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star) \\ &= \inf_{\boldsymbol{x} \in \mathbb{R}^N} \left(f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m^\star g_m(\boldsymbol{x}) + \boldsymbol{\nu}^{\star \mathrm{T}} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \right) \\ &\leq f(\boldsymbol{x}^\star) + \sum_{m=1}^{M} \lambda_m^\star g_m(\boldsymbol{x}^\star) + \boldsymbol{\nu}^{\star \mathrm{T}} (\boldsymbol{A}\boldsymbol{x}^\star - \boldsymbol{b}) \\ &\leq f(\boldsymbol{x}^\star). \end{aligned} \tag{5}$$

where the last inequality follows from the facts that we must have $\lambda_m^\star \geq 0$ and $g_m(\boldsymbol{x}^\star) \leq 0$ and that $\boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{b}$. Looking at this entire chain of inequalities, where the first and last term are both $f(\boldsymbol{x}^\star)$, means that

$$f(\boldsymbol{x}^\star) = \min_{\boldsymbol{x} \in \mathbb{R}^N} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star) = \mathcal{L}(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star).$$

In words, a solution to the primal problem $\boldsymbol{x}^\star$ is also a minimizer of $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$.

Strong duality and Slater's condition

As we have just seen, when we have strong duality there is a very close connection between the solutions of the primal and dual problems. So when can we expect strong duality to hold? For nonconvex problems, we rarely have strong duality, but for convex problems we usually (but not always) do.

Convexity is not quite enough to ensure strong duality, but there are additional conditions that we can require that will ensure that strong duality holds. Perhaps the most commonly encountered such condition is called **Slater's condition**. Informally, Slater's condition simply says that the feasible set has a non-empty interior. More formally, Slater's condition can be expressed as:

Slater's condition: There exists at least one $\bar{\boldsymbol{x}}$ such that for each inequality constraint g_m, either g_m is affine or

$$g_m(\bar{\boldsymbol{x}}) < 0.$$

That is, there is an $\bar{\boldsymbol{x}}$ that is *strictly* feasible for all non-affine constraints.

Nearly all of the optimization problems that we will encounter in this course will satisfy this condition. There are, however, convex problems that do not. As a simple example, let $\boldsymbol{p}_1 = [1, 0]^{\mathrm{T}}$ and $\boldsymbol{p}_2 = [-1, 0]^{\mathrm{T}}$ and consider the constraints

$$g_1(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{p}_1\|_2^2 - 1 \leq 0$$
$$g_2(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{p}_2\|_2^2 - 1 \leq 0.$$

Note that the only $\boldsymbol{x}$ satisfying both constraints is $\boldsymbol{x} = 0$ and there are no *strictly* feasible points.

Certificates of (sub)optimality

One potential application of the above facts is to serve as a way of measuring how far away we are from finding an optimal solution to our optimization problem. To see this recall that any dual feasible[3] $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ gives us a lower bound on $p^\star$, since $d(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^\star$. Thus, if we have a primal feasible $\boldsymbol{x}$, then we know that

$$f(\boldsymbol{x}) - p^\star \;\leq\; f(\boldsymbol{x}) - d(\boldsymbol{\lambda}, \boldsymbol{\nu}).$$

We will refer to $f(\boldsymbol{x}) - d(\boldsymbol{\lambda}, \boldsymbol{\nu})$ as the duality gap for the primal/dual (feasible) variables $\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}$. We know that

$$p^\star \in [d(\boldsymbol{\lambda}, \boldsymbol{\nu}), f(\boldsymbol{x})], \quad \text{and likewise} \quad d^\star \in [d(\boldsymbol{\lambda}, \boldsymbol{\nu}), f(\boldsymbol{x})].$$

If we are ever able to reduce this gap to zero, then we know that $\boldsymbol{x}$ is primal optimal, and $\boldsymbol{\lambda}, \boldsymbol{\nu}$ are dual optimal.

There are certain kinds of "primal-dual" algorithms that produce a series of (feasible) points $\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\nu}_k$ at every iteration. We can then use

$$f(\boldsymbol{x}_k) - d(\boldsymbol{\lambda}_k, \boldsymbol{\nu}_k) \leq \epsilon,$$

as a stopping criteria, and know that our answer would yield an objective value no further than ϵ from optimal.

[3] We simply need $\boldsymbol{\lambda} \geq \mathbf{0}$ for $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ to be dual feasible.

Examples

1. **Inequality LP**. Calculate the dual of

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \langle \boldsymbol{x}, \boldsymbol{c}\rangle \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}.$$

Answer: The Lagrangian is

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) &= \langle \boldsymbol{x}, \boldsymbol{c}\rangle + \sum_{m=1}^{M} \lambda_m \left(\langle \boldsymbol{x}, \boldsymbol{a}_m\rangle - b_m\right) \\ &= \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}.\end{aligned}$$

This is a linear functional in $\boldsymbol{x}$ — it is unbounded below unless

$$\boldsymbol{c} + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{0}.$$

Thus

$$\begin{aligned}d(\boldsymbol{\lambda}) &= \inf_{\boldsymbol{x}} \left(\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}\right) \\ &= \begin{cases} -\langle \boldsymbol{\lambda}, \boldsymbol{b}\rangle, & \boldsymbol{c} + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{0} \\ -\infty, & \text{otherwise.}\end{cases}\end{aligned}$$

So the Lagrange dual program is

$$\underset{\boldsymbol{\lambda}\in\mathbb{R}^M}{\text{maximize}} \ -\langle \boldsymbol{\lambda}, \boldsymbol{b}\rangle \quad \text{subject to} \quad \begin{aligned}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} &= -\boldsymbol{c} \\ \boldsymbol{\lambda} &\geq \boldsymbol{0}.\end{aligned}$$

2. **Least-squares**. Calculate the dual of

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

Check that the duality gap is zero.

Answer: The Lagrangian is

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}.$$

This is quadratic in $\boldsymbol{x}$ and will attain its minimum for

$$\boldsymbol{x} = -\frac{1}{2}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}.$$

Thus

$$\begin{aligned} d(\boldsymbol{\nu}) &= \frac{1}{4}\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\nu} - \frac{1}{2}\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu} \\ &= -\frac{1}{4}\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\nu}, \end{aligned}$$

and the Lagrange dual problem is

$$\underset{\boldsymbol{\nu}\in\mathbb{R}^M}{\text{maximize}} -\frac{1}{4}\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\nu}.$$

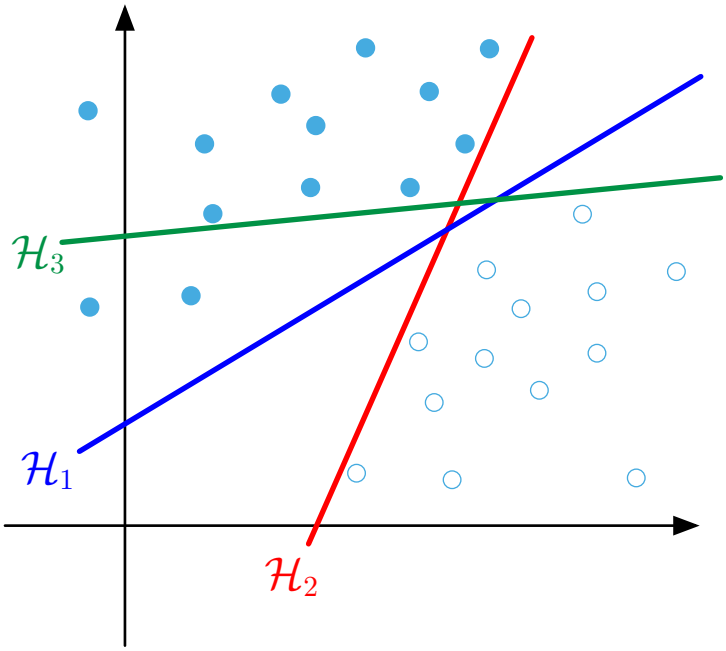
Note that this will be maximized when $-\frac{1}{2}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}^\star = \boldsymbol{b}$, which, when substituted into the dual problem yields

$$-\frac{1}{4}\boldsymbol{\nu}^{\star\mathrm{T}}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}^\star + \frac{1}{2}\boldsymbol{\nu}^{\star\mathrm{T}}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}\star}\boldsymbol{\nu} = \frac{1}{4}\boldsymbol{\nu}^{\star\mathrm{T}}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}^\star = \left\| -\frac{1}{2}\boldsymbol{A}^T\boldsymbol{\nu}^\star \right\|_2^2.$$

Since $\boldsymbol{x}^\star = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}^\star/2$ is primal feasible, we have found a $(\boldsymbol{x}^\star, \boldsymbol{\nu}^\star)$ such that $f(\boldsymbol{x}^\star) = d(\boldsymbol{\nu}^\star)$, so we see first hand that we have strong duality.

Example: Support vector machines

Consider the following fundamental binary classification problem. We are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M \in \mathbb{R}^N$ with class labels $y_1, \ldots, y_M$, where $y_m \in \{-1, +1\}$. We would like to find a hyperplane (i.e., affine functional) which *separates* the classes:



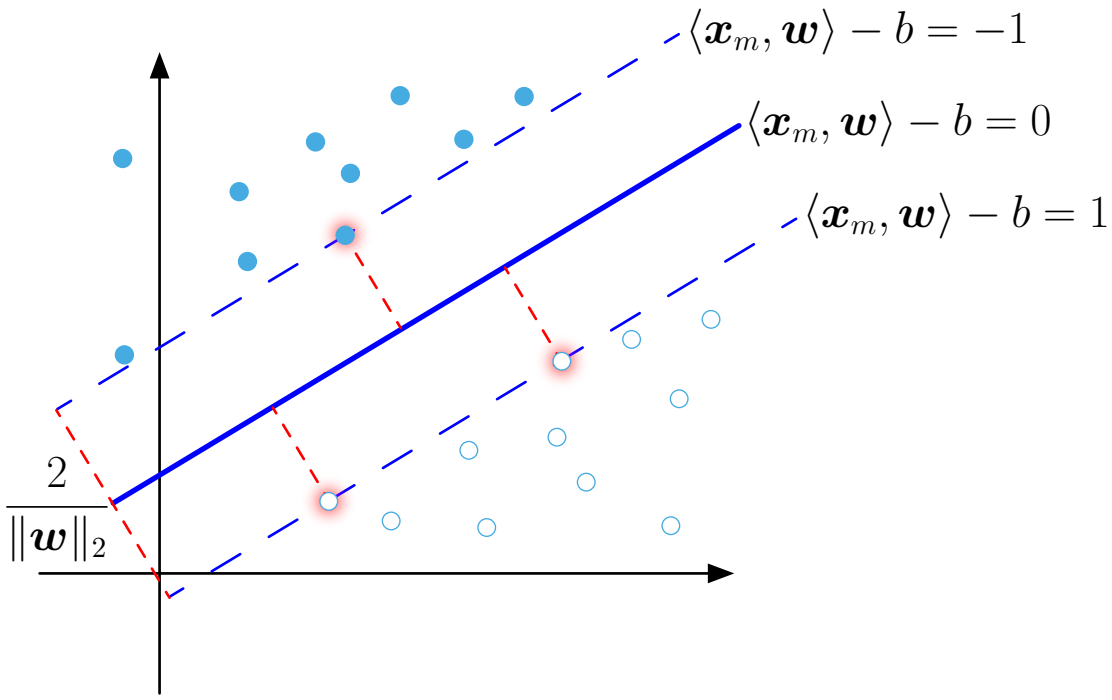
$\mathcal{H}_1$ and $\mathcal{H}_2$ above both separate the classes in $\mathbb{R}^2$, but $\mathcal{H}_3$ does not. While separating the classes is obviously desirable, we still need a good method to choose from among the many hyperplanes that do separate the classes – and some will perform better than others. Support vector machines (SVMs) take the one with *maximum margin*, i.e., we choose the hyperplane that maximizes the distance to the closest point in either class.

To restate this, we want to find a $\boldsymbol{w} \in \mathbb{R}^N$ and $b \in \mathbb{R}$ such that

$$\begin{aligned} \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle - b &\geq 1, \quad \text{when } y_m = 1, \\ \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle - b &\leq -1, \quad \text{when } y_m = -1. \end{aligned}$$

Of course, it is possible that no separating hyperplane exists; in this case, there will be no feasible points in the program above. It is straightforward, though, to modify this discussion to allow "mislabeled" points.

In the formulation above, the distance between the two (parallel) hyperplanes is $2/\|\boldsymbol{w}\|_2$:



Thus maximizing this distance is the same as minimizing $\|\boldsymbol{w}\|_2$.

This leads to the program

$$\begin{aligned} &\underset{\boldsymbol{w}\in\mathbb{R}^N,\ b\in\mathbb{R}}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ &\text{subject to} \ \ y_m(b - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1 \ \leq 0, \quad m = 1, \ldots, M. \end{aligned}$$

This is a linearly constrained quadratic program, and is clearly

convex. The Lagrangian is

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{m=1}^{M} \lambda_m \left[y_m(b - \langle \boldsymbol{x}_m, \boldsymbol{w}\rangle) + 1\right] \\ &= \frac{1}{2}\|\boldsymbol{w}\|_2^2 + b\,\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{1},\end{aligned}$$

where $\boldsymbol{X}$ is the $N \times M$ matrix

$$\boldsymbol{X} = \begin{bmatrix} y_1\boldsymbol{x}_1 & y_2\boldsymbol{x}_2 & \cdots & y_M\boldsymbol{x}_M \end{bmatrix}.$$

The dual function is

$$d(\boldsymbol{\lambda}) = \inf_{\boldsymbol{w}, b} \left(\frac{1}{2}\|\boldsymbol{w}\|_2^2 + b\,\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{1} \right).$$

Since b is unconstrained above, we see that the presence of $b\,\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{y}$ means that the dual will be $-\infty$ unless $\langle \boldsymbol{\lambda}, \boldsymbol{y}\rangle = 0$. Minimizing over $\boldsymbol{w}$, we need the gradient equal to zero,

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \mathbf{0}, \quad \Rightarrow \quad \boldsymbol{w} - \boldsymbol{X}\boldsymbol{\lambda} = \mathbf{0}.$$

This means that we must have $\boldsymbol{w} = \boldsymbol{X}\boldsymbol{\lambda}$, which itself is a very handy fact as it gives us a direct passage from the dual solution to the primal solution. With these substitutions, the dual function is

$$d(\boldsymbol{\lambda}) = \begin{cases} \frac{1}{2}\|\boldsymbol{X}\boldsymbol{\lambda}\|_2^2 - \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\lambda} + \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{1}, & \langle \boldsymbol{\lambda}, \boldsymbol{y}\rangle = 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

Thus, the dual SVM program is then

$$\begin{aligned} \underset{\boldsymbol{\lambda}}{\text{maximize}} \quad & -\frac{1}{2}\|\boldsymbol{X}\boldsymbol{\lambda}\|_2^2 + \sum_{m=1}^{M} \lambda_m \\ \text{subject to} \quad & \langle \boldsymbol{\lambda}, \boldsymbol{y} \rangle = 0, \quad \boldsymbol{\lambda} \geq \mathbf{0}. \end{aligned}$$

Given the solution $\boldsymbol{\lambda}^\star$ to the dual, we can take $\boldsymbol{w}^\star = \boldsymbol{X}\boldsymbol{\lambda}^\star$, and the classifier is

$$\begin{aligned} f(\boldsymbol{x}) &= \langle \boldsymbol{x}, \boldsymbol{w}^\star \rangle - b^\star \\ &= \langle \boldsymbol{x}, \boldsymbol{X}\boldsymbol{\lambda}^\star \rangle - b^\star \\ &= \sum_{m=1}^{M} \lambda_m^\star y_m \langle \boldsymbol{x}, \boldsymbol{x}_m \rangle - b^\star. \end{aligned}$$

Notice that the data $\boldsymbol{x}_m$ appear only through inner products with $\boldsymbol{x}$.

A key realization about the SVM is that the for the dual program, the objective function depends on the data $\boldsymbol{x}_m$ only through inner products, as

$$\|\boldsymbol{X}\boldsymbol{\lambda}\|_2^2 = \sum_{\ell=1}^{M} \sum_{m=1}^{M} y_\ell y_m \langle \boldsymbol{x}_\ell, \boldsymbol{x}_m \rangle.$$

This means that we can replace $\langle \boldsymbol{x}_\ell, \boldsymbol{x}_m \rangle$ with any "positive kernel function" $K(\boldsymbol{x}_\ell, \boldsymbol{x}_m) : \mathbb{R}^N \otimes \mathbb{R}^N \to \mathbb{R}$ – a positive kernel just means that the $M \times M$ matrix $K(\boldsymbol{x}_\ell, \boldsymbol{x}_m)$ is in S_+^M for all choices of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M$.

For example, you might take

$$K(\boldsymbol{x}_\ell, \boldsymbol{x}_m) = (1 + \langle \boldsymbol{x}_\ell, \boldsymbol{x}_m \rangle)^2 = 1 + 2\langle \boldsymbol{x}_\ell, \boldsymbol{x}_m \rangle + \langle \boldsymbol{x}_\ell, \boldsymbol{x}_m \rangle^2.$$

This means we have replaced the inner product of two vectors with the inner product between two vectors which have been mapped into a higher-dimensional space:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_N \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_N^2 \\ \sqrt{2}x_1x_2 \\ \vdots \\ \sqrt{2}x_{N-1}x_N \end{bmatrix}.$$

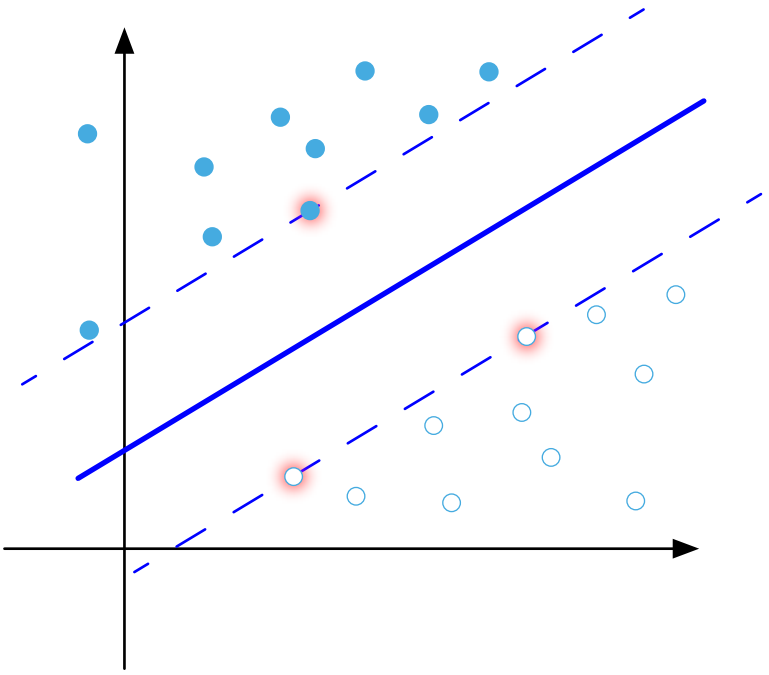
A set of linear constraints on the coordinates on the right, then, corresponds to a second order curve constraint (parabola, ellipse, hyperbola) on the coordinates on the left.

Many kernels are possible. The advantage is that to train and use the classifier, you never have to explicitly move to the higher-dimensional space – you just need to be able to compute $K(\boldsymbol{x}_\ell, \boldsymbol{x}_m)$ for any pair of inputs in $\mathbb{R}^N$. A popular choice of kernel is

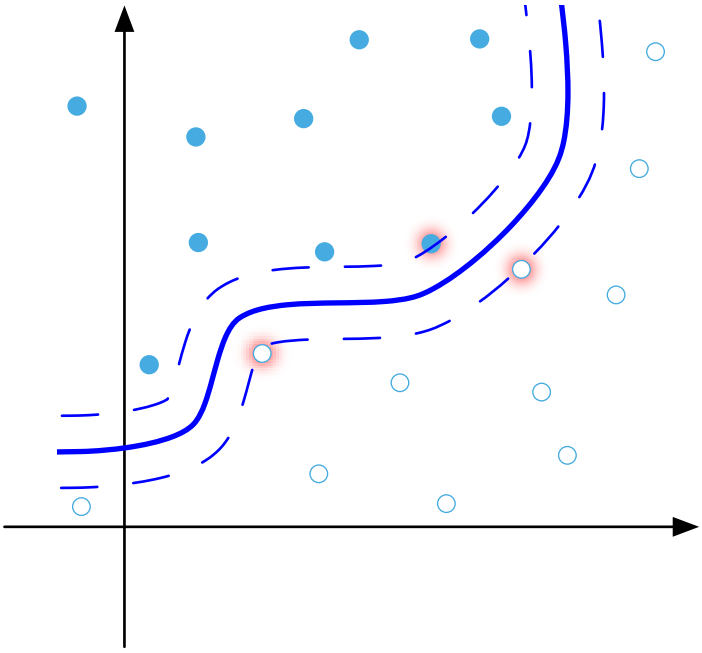
$$K(\boldsymbol{x}_\ell, \boldsymbol{x}_m) = \exp\left(-\gamma \|\boldsymbol{x}_\ell - \boldsymbol{x}_m\|_2^2\right).$$

This is a perfectly valid positive kernel, and it is straightforward to compute it for any pair of inputs. But it corresponds to mapping the $\boldsymbol{x}_m$ into an infinite dimensional space, then finding a separating hyperplane.

Here is an example of a linear classifier in a higher-dimensional space:



that results in a nonlinear classifier in a lower-dimensional space:



A second look at duality

Our first exposure to duality was in the context of constrained optimization: by introducing dual variables (Lagrange multipliers), we can combine the objective function and constraints into a single (Lagrangian) function which we can optimize either by minimizing it with respect to the primal variables or maximizing it with respect to the dual variables.

However, duality is a much broader concept than what we have seen so far, and can even be relevant in unconstrained problems.[1] As an example, suppose we wish to minimize the sum of two functions:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{x}) \tag{1}$$

where f and g are both convex. For simplicity, we will assume that $\operatorname{dom} f = \operatorname{dom} g = \mathbb{R}^N$, although we could easily extend this to the case where the domain is a subset of $\mathbb{R}^N$ by replacing f and/or g with their extension to $\mathbb{R}^N$. This problem is unconstrained, but we can actually represent it as a constrained problem of the form:

$$\underset{\boldsymbol{x},\boldsymbol{z}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x} = \boldsymbol{z}.$$

This formulation involves N affine equality constraints on the primal variables $\boldsymbol{x}$ and $\boldsymbol{z}$ of the form $h_n(\boldsymbol{x}, \boldsymbol{z}) = z_n - x_n = 0$. The Lagrangian function for this problem is

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \langle \boldsymbol{z} - \boldsymbol{x}, \boldsymbol{\nu} \rangle.$$

[1] Moreover, as we will soon see, if you give up on the requirement that your objective function is differentiable, the distinction between constrained and unconstrained problems becomes a bit blurred.

To derive the dual problem, we first must compute the dual function:

$$\begin{aligned} d(\boldsymbol{\nu}) &= \inf_{\boldsymbol{x},\boldsymbol{z}\in\mathbb{R}^N} \; f(\boldsymbol{x}) + g(\boldsymbol{z}) + \langle \boldsymbol{z}-\boldsymbol{x}, \boldsymbol{\nu}\rangle \\ &= \inf_{\boldsymbol{x}\in\mathbb{R}^N} \; f(\boldsymbol{x}) - \langle \boldsymbol{x}, \boldsymbol{\nu}\rangle + \inf_{\boldsymbol{z}\in\mathbb{R}^N} \; g(\boldsymbol{z}) - \langle \boldsymbol{z}, -\boldsymbol{\nu}\rangle \\ &= \underbrace{-\sup_{\boldsymbol{x}\in\mathbb{R}^N} \; \langle \boldsymbol{x}, \boldsymbol{\nu}\rangle - f(\boldsymbol{x})}_{f^*(\boldsymbol{\nu})} \underbrace{- \sup_{\boldsymbol{z}\in\mathbb{R}^N} \; \langle \boldsymbol{z}, -\boldsymbol{\nu}\rangle - g(\boldsymbol{z})}_{g^*(-\boldsymbol{\nu})}, \end{aligned}$$

where

$$f^*(\boldsymbol{\nu}) := \sup_{\boldsymbol{x}\in\mathbb{R}^N} \; \langle \boldsymbol{x}, \boldsymbol{\nu}\rangle - f(\boldsymbol{x})$$

is the **convex conjugate** (or **Fenchel conjugate**) of f. We will try to give a bit more intuition into what the convex conjugate of a function represents below, but first we note that if we can calculate this function, then the resulting dual problem is

$$\underset{\boldsymbol{\nu}\in\mathbb{R}^N}{\text{maximize}} \;\; -f^*(\boldsymbol{\nu}) - g^*(-\boldsymbol{\nu})$$

or equivalently

$$\underset{\boldsymbol{\nu}\in\mathbb{R}^N}{\text{minimize}} \;\; f^*(\boldsymbol{\nu}) + g^*(-\boldsymbol{\nu}). \tag{2}$$

Before we can see this in action on some real optimization problems, however, we first need to understand what the convex conjugate is and, given a function f, how to actually compute f^*.

The convex conjugate

Before going any further, the first thing to say about the convex conjugate is that it is, as its name might suggest, convex. In fact, $f^*(\boldsymbol{\nu})$ is convex, *even if* $f(\boldsymbol{x})$ *is not*. We have seen the argument for this before: $f^*(\boldsymbol{\nu})$ is the pointwise supremum of convex functions since for any fixed $\boldsymbol{x}$, $\langle \boldsymbol{x}, \boldsymbol{\nu} \rangle - f(\boldsymbol{x})$ is a convex (affine) function).

The convex conjugate plays a fundamental role in duality. Before we assumed that $\operatorname{dom} f = \mathbb{R}^N$, but it will sometimes be useful to be explicit about the domain. If we let $\mathcal{D} = \operatorname{dom} f$, then the convex conjugate of f is

$$f^*(\boldsymbol{\nu}) = \sup_{\boldsymbol{x} \in \mathcal{D}} \ \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle - f(\boldsymbol{x}).$$

There are multiple ways to think about the convex conjugate. Perhaps the most natural is simply that $f^*(\boldsymbol{\nu})$ is simply the maximum amount that the linear functional $\langle \boldsymbol{x}, \boldsymbol{\nu} \rangle$ exceeds $f(\boldsymbol{x})$.

Let's consider a particular example in one dimension ($N = 1$). Suppose

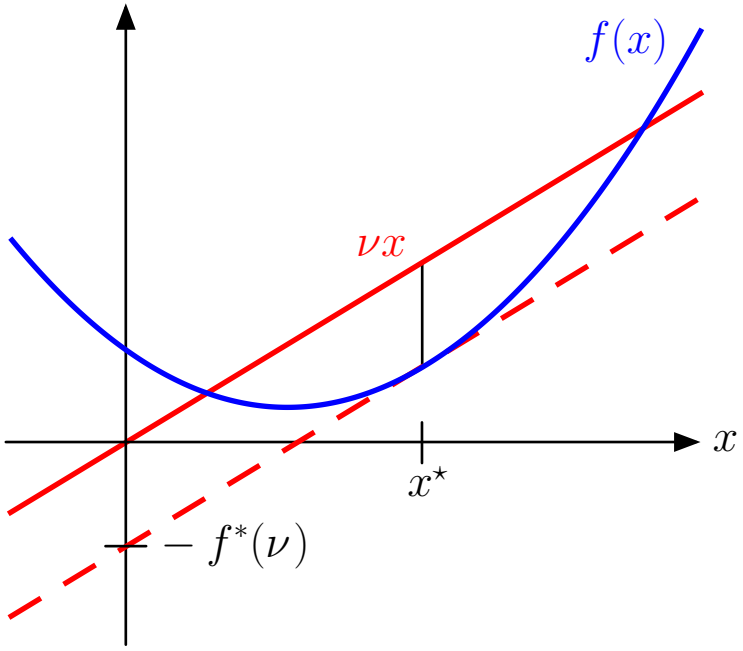
$$f(x) = x^2 - 2x + 2 = (x-1)^2 + 1.$$

We have

$$f^*(\nu) = \sup_{x \in \mathbb{R}} \left(\nu x - x^2 + 2x - 2 \right) = \frac{\nu^2}{4} + \nu - 1.$$

(You can verify the second equality by taking the derivative with respect to x, setting this to zero, and solving for x. This results in $x = \frac{\nu}{2} - 1$, and plugging this in yields the result above.)

Here is an example of what is happening:



Here we illustrate a function $f(x)$ overlayed with νx for an example value of ν. The *vertical* difference between the two functions is maximized at a particular value $x^\star$, and this distance is $f^*(\nu)$.

Another way to think about $f^*(\nu)$ is that it tells us how far we need to shift νx down so that it will be tangent to $f(x)$ (or a subgradient of f at x if f is not differentiable), as illustrated by the dashed line. In the case where f is differentiable, this is easy to see: since f is convex, $-f$ is concave and $\nu x - f(x)$ will be maximized when $\nu - f'(x) = 0$.

Properties

- $f^*(\boldsymbol{\nu})$ is convex (even when $f(\boldsymbol{x})$ is not).
- Fenchel's inequality: For any $\boldsymbol{x}$ and $\boldsymbol{\nu}$ we have

$$f(\boldsymbol{x}) + f^*(\boldsymbol{\nu}) \geq \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle.$$

- For any function $f(\boldsymbol{x})$, we can define the conjugate of $f^*(\boldsymbol{\nu})$ as

$$f^{**}(\boldsymbol{x}) = \sup_{\boldsymbol{\nu} \in \mathcal{D}^*} \ \langle \boldsymbol{\nu}, \boldsymbol{x} \rangle - f^*(\boldsymbol{\nu}),$$

  where $\mathcal{D}^*$ is the domain of f^*. For an arbitrary $f(\boldsymbol{x})$ we have

$$f^{**}(\boldsymbol{x}) \leq f(\boldsymbol{x})$$

- If $f(\boldsymbol{x})$ is convex and has a closed epigraph, then taking the conjugate of $f^*(\boldsymbol{\nu})$ recovers $f(\boldsymbol{x})$:

$$f^{**}(\boldsymbol{x}) = f(\boldsymbol{x}).$$

- If $f(\boldsymbol{x}_1, \boldsymbol{x}_2)$ can be written as the sum of two independent variables:

$$f(\boldsymbol{x}_1, \boldsymbol{x}_2) = f_1(\boldsymbol{x}_1) + f_2(\boldsymbol{x}_2),$$

  then

$$f^*(\boldsymbol{a}_1, \boldsymbol{a}_2) = f_1^*(\boldsymbol{a}_1) + f_2^*(\boldsymbol{a}_2).$$

For more properties, see [BV04, Chapter 3.3].

Examples

The Indicator Function

We define the **indicator function** or **characteristic function** for a convex set $\mathcal{C}$ is given by

$$I_{\mathcal{C}}(\boldsymbol{x}) = \begin{cases} 0, & \text{if } x \in \mathcal{C}, \\ +\infty, & \text{if } x \notin \mathcal{C}. \end{cases}$$

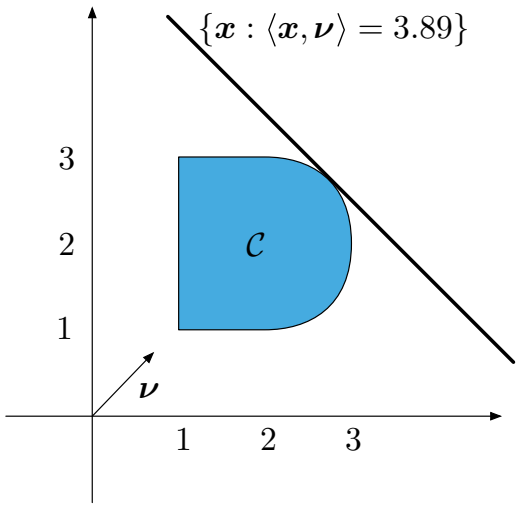
The convex conjugate of the indicator function is

$$\begin{aligned} h_{\mathcal{C}}(\boldsymbol{\nu}) = I_{\mathcal{C}}^*(\boldsymbol{\nu}) &= \sup_{\boldsymbol{x} \in \mathbb{R}^N} \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle - I_{\mathcal{C}}(\boldsymbol{x}) \\ &= \sup_{\boldsymbol{x} \in \mathcal{C}} \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle. \end{aligned}$$

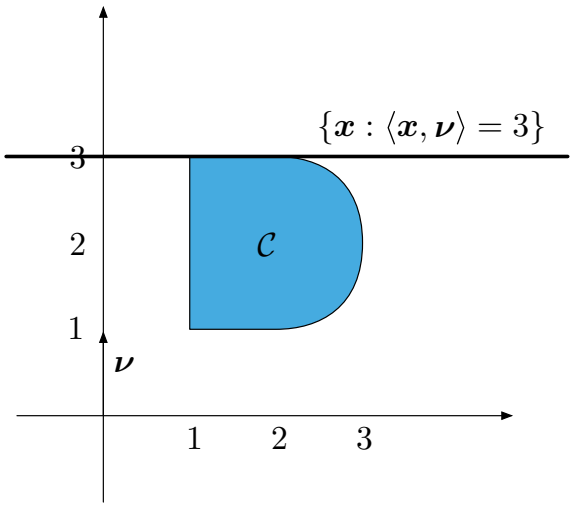
The function $h_{\mathcal{C}}(\boldsymbol{\nu})$ is also called the **support function** of $\mathcal{C}$. The support function defines a vector $\boldsymbol{\nu} \in \mathbb{R}^N$ that defines a linear function on $\mathcal{C}$ and then returns the maximum value of that linear function over $\mathcal{C}$.

Geometrically, this corresponds to taking the half-space $\{\boldsymbol{x} : \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle \leq b\}$ and then determines how large b needs to be to ensure that the half-space contains all of $\mathcal{C}$ (since by definition we will have $\langle \boldsymbol{x}, \boldsymbol{\nu} \rangle \leq h_{\mathcal{C}}(\boldsymbol{\nu})$ for all $\boldsymbol{x} \in \mathcal{C}$).

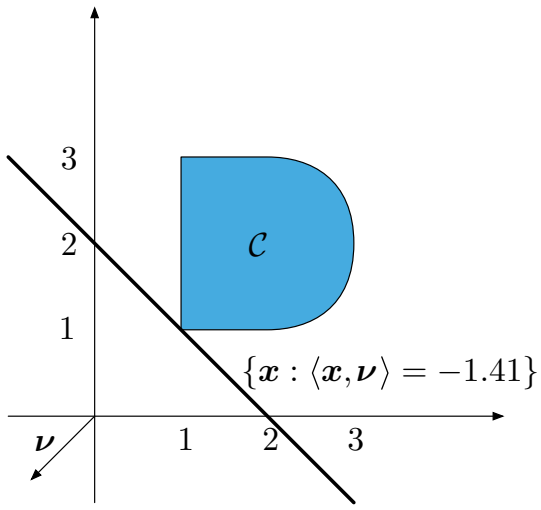
This is illustrated with an example on the following page.



$$\boldsymbol{\nu} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$h_{\mathcal{C}}(\boldsymbol{\nu}) = 3.89$$



$$\boldsymbol{\nu} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$h_{\mathcal{C}}(\boldsymbol{\nu}) = 3$$



$$\boldsymbol{\nu} = -\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$h_{\mathcal{C}}(\boldsymbol{\nu}) = -1.41$$

Norms

As we have already seen on many occasions, norms frequently arise in optimization problems as either objective functions or constraints, and so their convex conjugates are important to be able to compute and work with. Before discussing the general case, let's look at what happens for $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2$ to get an idea of what we will need to do. In this case we have

$$\begin{aligned} f^*(\boldsymbol{\nu}) &= \sup_{\boldsymbol{x} \in \mathbb{R}^N} \ \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle - \|\boldsymbol{x}\|_2 \\ &= \sup_{\boldsymbol{x} \in \mathbb{R}^N} \ \|\boldsymbol{x}\|_2 \|\boldsymbol{\nu}\|_2 - \|\boldsymbol{x}\|_2 \\ &= \sup_{\boldsymbol{x} \in \mathbb{R}^N} \ \|\boldsymbol{x}\|_2 (\|\boldsymbol{\nu}\|_2 - 1), \end{aligned}$$

where the second inequality above follows from Cauchy-Schwarz (which can be made to hold with equality). There are two cases to consider here. If $\|\boldsymbol{\nu}\|_2 \leq 1$, then we are trying to maximize a non-positive quantity, which we can do simply by setting $\boldsymbol{x} = \boldsymbol{0}$, resulting in $f^*(\boldsymbol{\nu}) = 0$. However, if $\|\boldsymbol{\nu}\|_2 > 1$ then $\|\boldsymbol{x}\|_2(\|\boldsymbol{\nu}\|_2 - 1)$ can be made arbitrarily large. Thus

$$f^*(\boldsymbol{\nu}) = \begin{cases} 0, & \text{if } \|\boldsymbol{\nu}\|_2 \leq 1, \\ +\infty, & \text{if } \|\boldsymbol{\nu}\|_2 > 1. \end{cases}$$

In other words, $f^*(\boldsymbol{\nu})$ is the indicator function for the unit ball corresponding to $\|\cdot\|_2$.

Now suppose that $f(\boldsymbol{x}) = \|\boldsymbol{x}\|$ where $\|\cdot\|$ denotes an arbitrary norm. To extend the argument above, we will need to re-introduce a the dual norm (which we first encountered in our discussion of subdifferentials).

Recall that for a norm $\|\cdot\|$ the **dual norm** is defined as

$$\|\boldsymbol{\nu}\|_* = \sup_{\boldsymbol{x}:\|\boldsymbol{x}\|\leq 1} \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle.$$

Note that this is the support function (the convex conjugate of the indicator function) of the unit ball corresponding to $\|\cdot\|$. Moreover, as a direct consequence of the definition of the dual norm we have

$$\langle \boldsymbol{x}, \boldsymbol{\nu} \rangle \leq \|\boldsymbol{x}\| \|\boldsymbol{\nu}\|_*.$$

This is exactly what we need to extend our previous argument.

We again begin with setting $f(\boldsymbol{x}) = \|\boldsymbol{x}\|$ and wish to compute

$$f^*(\boldsymbol{\nu}) = \sup_{\boldsymbol{x}\in\mathbb{R}^N} \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle - \|\boldsymbol{x}\|$$

As before, we can consider two cases. If $\|\boldsymbol{\nu}\|_* \leq 1$ then

$$\langle \boldsymbol{x}, \boldsymbol{\nu} \rangle - \|\boldsymbol{x}\| \leq \|\boldsymbol{x}\| \|\boldsymbol{\nu}\|_* - \|\boldsymbol{x}\| \leq 0.$$

In this case the largest we can make $f^*(\boldsymbol{\nu})$ is zero (by setting $\boldsymbol{x} = 0$). On the other hand, if $\|\boldsymbol{\nu}\|_* > 1$ then there must exist an $\boldsymbol{x}$ such that $\langle \boldsymbol{x}, \boldsymbol{\nu} \rangle \geq \|\boldsymbol{x}\|$. If we replace this $\boldsymbol{x}$ by a rescaled version $t\boldsymbol{x}$, then we can make

$$\langle t\boldsymbol{x}, \boldsymbol{\nu} \rangle - \|t\boldsymbol{x}\| = t(\langle \boldsymbol{x}, \boldsymbol{\nu} \rangle - \|\boldsymbol{x}\|)$$

arbitrarily large. Thus we have

$$f^*(\boldsymbol{\nu}) = \begin{cases} 0, & \text{if } \|\boldsymbol{\nu}\|_* \leq 1, \\ +\infty, & \text{if } \|\boldsymbol{\nu}\|_* > 1, \end{cases}$$

i.e., $f^*(\boldsymbol{\nu})$ is the indicator function for the unit ball corresponding to the dual norm $\|\cdot\|_*$.

For lots of other examples, see [BV04, Chapter 3.3].

Fenchel duality

We began these notes by showing that if we consider the unconstrained problem

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{x}) \tag{3}$$

where f and g are both convex, we can derive the equivalent dual problem

$$\underset{\boldsymbol{\nu}}{\text{maximize}} \; -f^*(\boldsymbol{\nu}) - g^*(-\boldsymbol{\nu}). \tag{4}$$

Recall from our first discussion of Lagrange duality that the dual problem provides a lower bound for the primal problem, or in the language of the problems above, we have

$$\inf_{\boldsymbol{x}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{x}) \geq \sup_{\boldsymbol{\nu}} -f^*(\boldsymbol{\nu}) - g^*(-\boldsymbol{\nu}).$$

Moreover, under certain conditions we have *strong duality*. In this setting, strong duality implies that the above inequality will hold with equality, i.e.,

$$\inf_{\boldsymbol{x}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{x}) = \sup_{\boldsymbol{\nu}} -f^*(\boldsymbol{\nu}) - g^*(-\boldsymbol{\nu}). \tag{5}$$

Fenchel's Duality Theorem tells us that under certain regularity assumptions on f and g, we have strong duality and (5) holds.[2] Specifically, if $\mathcal{D} = \operatorname{dom} f$ and $\mathcal{C}$ denotes the set of $\boldsymbol{x} \in \mathbb{R}^N$ where g is finite and continuous, then (5) holds whenever there exists an $\bar{\boldsymbol{x}} \in \operatorname{relint}(\mathcal{D} \cap \mathcal{C})$.

[2] In our discussion here as well as when reviewing Lagrangian duality, we have assumed that $\inf_{\boldsymbol{x}} f(\boldsymbol{x}) + g(\boldsymbol{x})$ is finite (so that these quantities are even defined). If the primal (or dual) is not bounded, then there is no solution to the optimization problem and strong duality will not hold.

Note that if $g(\boldsymbol{x})$ is the indicator for a convex set $\mathcal{C}$, then this is equivalent to a constrained optimization problem, and the conditions above are equivalent to the assumption that there is a strictly feasible point in $\operatorname{dom} f$, i.e., Slater's condition. In this case we can also write (5) more cleaning if we define a new function $h'_{\mathcal{C}}(\boldsymbol{\nu})$ which is related to the support function of $\mathcal{C}$, just with an infimum instead of a supremum:

$$h'_{\mathcal{C}}(\boldsymbol{\nu}) = \inf_{\boldsymbol{x}\in\mathcal{C}} \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle = -\sup_{\boldsymbol{x}\in\mathcal{C}} \langle \boldsymbol{x}, -\boldsymbol{\nu} \rangle = -h_{\mathcal{C}}(-\boldsymbol{\nu}).$$

With this notation we can re-write (5) for the case of standard constrained optimization as

$$\inf_{\boldsymbol{x}\in\mathcal{C}} f(\boldsymbol{x}) = \sup_{\boldsymbol{\nu}} h'_{\mathcal{C}}(\boldsymbol{\nu}) - f^*(\boldsymbol{\nu}). \tag{6}$$

We will not do so here, but from this point you can actually show that if our constraints match the form that is assumed in our discussion of Lagrangian duality, then the right-hand side of (6) exactly corresponds to the Lagrangian dual problem. In this sense Lagrangian duality is just a special case of Fenchel duality.

Super-Easy Example

Before we look at serious applications of Fenchel duality, let's look at a very simple example just to get a feel for the computations involved. We will compute

$$\inf_{x\in[3,5]} x^3.$$

Of course, we know the answer already: it is 27, as the function above achieves its minimum value at $\widehat{x} = 3$. But let's verify the Fenchel duality theorem for this case.

We will take $f(x) = x^3$ which is convex over the non-negative reals, so we take $\mathcal{D} = \{x : x \geq 0\}$. The constraint set is the interval $\mathcal{C} = [3, 5]$. First we compute

$$h'_{\mathcal{C}}(\nu) = \inf_{x \in [3,5]} \nu x = \begin{cases} 3\nu, & \nu \geq 0, \\ 5\nu, & \nu < 0. \end{cases}$$

The conjugate of f is

$$f^*(\nu) = \sup_{x \geq 0} (\nu x - x^3).$$

For fixed $\nu \geq 0$, this expression is maximized at $x^\star = \sqrt{\nu/3}$; for $\nu < 0$ it is maximized at $x^\star = 0$. Thus

$$f^*(\nu) = \begin{cases} \frac{2}{3}\sqrt{\frac{\nu^3}{3}}, & \nu \geq 0, \\ 0, & \nu < 0. \end{cases}$$

Thus

$$\max_{\nu \in \mathbb{R}} [h'_{\mathcal{C}}(\nu) - f^*(\nu)] = \max_{\nu \in \mathbb{R}} \begin{cases} 3\nu - \frac{2}{3}\sqrt{\frac{\nu^3}{3}}, & \nu \geq 0, \\ 5\nu, & \nu < 0. \end{cases}$$

It is easy to check that this expression is maximized at $\nu^\star = 27$ (coincidence), and that

$$\left(3\nu - \frac{2}{3}\sqrt{\frac{\nu^3}{3}} \right) \Bigg|_{\nu = 27} = 27.$$

Example: Resource allocation [Lue69]

The "law of diminishing returns" is a fundamental tenet of economics: as we put more and more resources into something, at some point, the incremental gains become less and less. You see this everywhere: what is the difference between spending \$5 on dinner, \$50 on dinner, \$500 on dinner? What are the differences between a \$50 bicycle, a \$500 bicycle, and a \$5000 bicycle?

What this means is that functions $f(x)$ that map resources to return are concave.

Suppose we have D dollars that we would like to allocate to N different activities in such a way that maximizes the return. The return of each activity is a (possibly different) concave function $f_n(x_n)$, where x_n is the amount of money invested. Our optimization problem is

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{maximize}} \;\; f(\boldsymbol{x}) = \sum_{n=1}^{N} f_n(x_n) \quad \text{subject to} \quad \sum_{n=1}^{N} x_n = D \\ \boldsymbol{x} \geq \mathbf{0},$$

or equivalently

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; \tilde{f}(\boldsymbol{x}) = -f(\boldsymbol{x}) \quad \text{subject to} \quad \sum_{n=1}^{N} x_n = D \\ \boldsymbol{x} \geq \mathbf{0},$$

This is a convex optimization problem in N variables, and of course its solution depends on what we actually choose for the return functions $f_n(x_n)$. However, by using Fenchel duality, we can recast this problem as an optimization in a single variable.

Since the natural domain of the f_n is $x \geq 0$, let's take

$$\mathcal{D} = \{\boldsymbol{x} : \boldsymbol{x} \geq \mathbf{0}\}, \quad \mathcal{C} = \{\boldsymbol{x} : x_1 + \cdots + x_N = D\}.$$

We start by computing

$$h'_{\mathcal{C}}(\boldsymbol{\nu}) = \inf_{\boldsymbol{x} \in \mathcal{C}} \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle.$$

Since $\mathcal{C}$ is itself an affine set, $h'_{\mathcal{C}}(\boldsymbol{\nu})$ is unbounded below for almost every $\boldsymbol{\nu}$ we plug in – the exception is if all of the entries of $\boldsymbol{\nu}$ are equal to one another. In this case,

$$\boldsymbol{\nu} = \lambda \mathbf{1}, \quad h'_{\mathcal{C}}(\boldsymbol{\nu}) = D\lambda,$$

where $\mathbf{1}$ is an N-vector of all ones. Thus, we have

$$h'_{\mathcal{C}}(\boldsymbol{\nu}) = \begin{cases} D\lambda, & \boldsymbol{\nu} = \lambda \mathbf{1} \\ -\infty, & \text{otherwise.} \end{cases}$$

Now we compute the conjugate $\tilde{f}^*(\boldsymbol{\nu})$ of $\tilde{f}(\boldsymbol{x}) = -f(\boldsymbol{x})$. Since $\tilde{f}$ is a sum of convex functions of independent variables,

$$\tilde{f}^*(\boldsymbol{\nu}) = \sum_{n=1}^{N} \tilde{f}_n^*(\nu_n),$$

where $\tilde{f}_n^*(\nu_n)$ is the conjugate of a function of a single variable:

$$\tilde{f}_n^*(\nu_n) = \sup_{x \geq 0} \left[\nu_n x - \tilde{f}_n(x) \right].$$

This means we can write the dual as

$$\max_{\boldsymbol{\nu}} \left[h'_{\mathcal{C}}(\boldsymbol{\nu}) - \tilde{f}^*(\boldsymbol{\nu}) \right] \;=\; \max_{\lambda \in \mathbb{R}} \left[D\lambda - \sum_{n=1}^{N} \tilde{f}_n^*(\lambda) \right].$$

That is, the expression to be minimized is a function **of a single variable** λ. All we need to know how to do is evaluate the conjugate functions $\tilde{f}_n^*$.

Example: Norm minimization

Here we look at an example of how we can apply Fenchel duality to provide an alternative characterization of a common constrained optimization program. Consider the "norm minimization" problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \|\boldsymbol{x}\| \quad \text{subject to} \ \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y},$$

where the vector $\boldsymbol{y} \in \mathbb{R}^M$ and matrix $\boldsymbol{A} \in \mathbb{R}^{M \times N}$ are given and we assume that $\text{rank}(\boldsymbol{A})$ is at most N so that we are guaranteed that there is at least one feasible solution. Here the norm in the objective function is an arbitrary norm. We will derive the dual for the general case, which could then be specialized to tell us something about least squares (for the ℓ_2 norm), "Basis Pursuit" (for the ℓ_1 norm), or the result for any choice of norm.

We have already calculated $f^*(\boldsymbol{\nu})$ for the case where $f(\boldsymbol{x}) = \|\boldsymbol{x}\|$. In this case $f^*(\boldsymbol{\nu})$ is the indicator function for the set $\{\boldsymbol{\nu} : \|\boldsymbol{\nu}\|_* \leq 1\}$. If we plug this into (6) we see that the objective function will be $-\infty$ unless $\|\boldsymbol{\nu}\|_* \leq 1$, and thus we can equivalently write the dual problem as

$$\underset{\boldsymbol{\nu}}{\text{maximize}} \ h'_{\mathcal{C}}(\boldsymbol{\nu}) \quad \text{subject to} \ \ \|\boldsymbol{\nu}\|_* \leq 1,$$

where $\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}\}$. This, it remains to calculate $h'_{\mathcal{C}}(\boldsymbol{\nu})$.

We first note that if $\langle \boldsymbol{u}, \boldsymbol{\nu} \rangle \neq 0$ for some $\boldsymbol{u} \in \text{Null}(\boldsymbol{A})$, then $h'_{\mathcal{C}}(\boldsymbol{\nu}) = -\infty$. To see this, note that if $\boldsymbol{u} \in \text{Null}(A)$ then for any $\boldsymbol{x} \in \mathcal{C}$ and $t \in \mathbb{R}$, $\boldsymbol{A}(\boldsymbol{x} + t\boldsymbol{u}) = \boldsymbol{y}$. Thus, $\boldsymbol{x} + t\boldsymbol{u} \in \mathcal{C}$, and $\langle \boldsymbol{x} + t\boldsymbol{u}, \boldsymbol{\nu} \rangle = \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle + t\langle \boldsymbol{u}, \boldsymbol{\nu} \rangle$, which is unbounded since t can be arbitrary.

What remains is to calculate $h'_{\mathcal{C}}(\boldsymbol{\nu})$ for $\boldsymbol{\nu}$ that are orthogonal to Null($\boldsymbol{A}$). Recall that this is equivalent to the assumption that $\boldsymbol{\nu} \in$ Range($\boldsymbol{A}^{\mathrm{T}}$). For any such $\boldsymbol{\nu}$ we can write $\boldsymbol{\nu} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}$ for some $\boldsymbol{w} \in \mathbb{R}^M$, in which case

$$\begin{aligned}\inf_{\boldsymbol{x}\in\mathcal{C}} \ \langle \boldsymbol{x}, \boldsymbol{\nu} \rangle &= \inf_{\boldsymbol{x}\in\mathcal{C}} \ \langle \boldsymbol{x}, \boldsymbol{A}^{\mathrm{T}}\boldsymbol{w} \rangle \\ &= \inf_{\boldsymbol{x}\in\mathcal{C}} \ \langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{w} \rangle \\ &= \inf_{\boldsymbol{x}\in\mathcal{C}} \ \langle \boldsymbol{y}, \boldsymbol{w} \rangle \\ &= \langle \boldsymbol{y}, \boldsymbol{w} \rangle.\end{aligned}$$

Thus, if we replace $\boldsymbol{\nu}$ in our dual problem with $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}$ and optimize over $\boldsymbol{w}$ instead, we arrive at the dual problem

$$\underset{\boldsymbol{w}}{\text{maximize}} \ \langle \boldsymbol{y}, \boldsymbol{w} \rangle \quad \text{subject to} \ \ \|\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}\|_* \leq 1.$$

As an example, if we consider the ℓ_1-norm minimization problem (also known as "Basis Pursuit") where $\|\cdot\| = \|\cdot\|_1$, the dual becomes

$$\underset{\boldsymbol{w}}{\text{maximize}} \ \langle \boldsymbol{y}, \boldsymbol{w} \rangle \quad \text{subject to} \ \ \|\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}\|_\infty \leq 1.$$

Note that this is a standard linear program. This can be a useful observation from a computational perspective, but later in the course we will show how Fenchel duality for this problem can also be used to provide a theoretical characterization of the properties (e.g., sparsity) of the solution $\boldsymbol{x}^\star$ of the primal problem.

Example: Closest point problem

Recall our closest point to a (closed) convex set $\mathcal{C}$ problem

$$\underset{\boldsymbol{x}\in\mathcal{C}}{\text{minimize}}\, \|\boldsymbol{x}-\boldsymbol{x}_0\|_2.$$

We already know that there is a unique primal solution (unique closest point) $\boldsymbol{x}^\star$. From our work above, we know that the dual of this problem is

$$\underset{\boldsymbol{\nu}}{\text{maximize}}\, -h_{\mathcal{C}}(-\boldsymbol{\nu}) - f^*(\boldsymbol{\nu}),$$

where $h_{\mathcal{C}}$ is the support function for $\mathcal{C}$ defined above and

$$\begin{aligned} f^*(\boldsymbol{\nu}) &= \sup_{\boldsymbol{x}\in\mathbb{R}^N} \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x} - \|\boldsymbol{x}-\boldsymbol{x}_0\|_2 \\ &= \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0 + \sup_{\boldsymbol{x}'\in\mathbb{R}^N} \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}' - \|\boldsymbol{x}'\|_2 \\ &= \begin{cases} \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0, & \|\boldsymbol{\nu}\|_2 \le 1 \\ \infty, & \|\boldsymbol{\nu}\|_2 > 1. \end{cases} \end{aligned}$$

Thus the dual is equivalent to

$$\underset{\|\boldsymbol{\nu}\|_2\le 1}{\text{maximize}}\;\; -h_{\mathcal{C}}(\boldsymbol{\nu}) - \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0,$$

which (by maximizing over $-\boldsymbol{\nu}$ in place of $\boldsymbol{\nu}$) is again equivalent to

$$\underset{\|\boldsymbol{\nu}\|_2\le 1}{\text{maximize}}\; d(\boldsymbol{\nu}) := \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0 - h_{\mathcal{C}}(\boldsymbol{\nu}).$$

Note that

$$\begin{aligned} d(\boldsymbol{\nu}) &= \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0 - \sup_{\boldsymbol{x}\in\mathcal{C}} \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x} \\ &= \inf_{\boldsymbol{x}\in\mathcal{C}}\; \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{x}_0-\boldsymbol{x}) \\ &\le \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{x}_0-\boldsymbol{x}^\star) \end{aligned}$$

From the problem statement, we know that we have strong duality, so there exists a $\boldsymbol{\nu}^\star$ with $\|\boldsymbol{\nu}^\star\|_2 \leq 1$ such that

$$d(\boldsymbol{\nu}^\star) = \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2.$$

Since it is also true that for all $\boldsymbol{\nu}$ with $\|\boldsymbol{\nu}\|_2 \leq 1$,

$$d(\boldsymbol{\nu}) \leq \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{x}_0 - \boldsymbol{x}^\star) \leq \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2,$$

by Cauchy-Schwarz we must have

$$\boldsymbol{\nu}^\star = \frac{\boldsymbol{x}_0 - \boldsymbol{x}^\star}{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2}..$$

Let's take a little closer look at the dual function

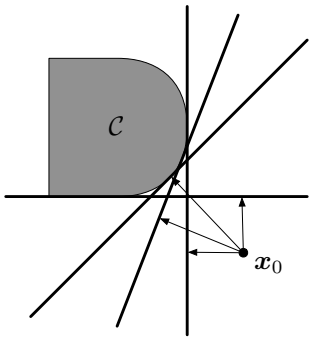
$$d(\boldsymbol{\nu}) = \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0 - h_{\mathcal{C}}(\boldsymbol{\nu}).$$

Given a fixed $\boldsymbol{\nu}$, we know that $(\boldsymbol{\nu}, h_{\mathcal{C}}(\boldsymbol{\nu}))$ define a supporting hyperplane for $\mathcal{C}$, i.e.

$$\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x} - h_{\mathcal{C}}(\boldsymbol{\nu}) \leq 0, \quad \text{for all } \boldsymbol{x} \in \mathcal{C}.$$

When the dual $d(\boldsymbol{\nu}) \geq 0$, i.e. $\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0 - h_{\mathcal{C}}(\boldsymbol{\nu}) \geq 0$, we know that hyperplane *separates* $\boldsymbol{x}_0$ from $\mathcal{C}$. For $\|\boldsymbol{\nu}\|_2 = 1$, we also know that $\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{x}_0 - h_{\mathcal{C}}(\boldsymbol{\nu})$ is the *distance* to the separating hyperplane, and over all separating hyperplanes with normal vector $\boldsymbol{\nu}$, $(\boldsymbol{\nu}, h_{\mathcal{C}}(\boldsymbol{\nu}))$ is the one that is farthest away.

Thus we can interpret the dual program as a search over all hyperplanes that separate $\boldsymbol{x}_0$ and $\mathcal{C}$ and choosing the one that is maximally distant from $\boldsymbol{x}_0$.



References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Lue69] D. G. Luenberger. *Optimization by Vector Space Methods*. Wiley, 1969.

Algorithms for constrained optimization

There are many, many constrained optimization algorithms, each tuned to the particulars of different classes of problems. We will look at the basics that underlie some of the more modern techniques. We will see that the concept of duality both helps us understand how these algorithms work, and gives us a way of determining when we are close to the solution.

We will describe several techniques. Nearly all of these ultimately work by replacing the constrained program with an unconstrained program (or a series of unconstrained programs).

Eliminating equality constraints

The first approach is not so much an algorithm as a "trick" that lets us sometimes avoid even thinking about the constraints. Programs with *linear* equality constraints can always be written as programs without, and if there are no inequality constraints then the new program is unconstrained. To see this, suppose we are solving

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

Let $\boldsymbol{x}_0$ be any point satisfying $\boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{b}$. Then any feasible $\boldsymbol{x}$ can be written as $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{h}$, where $\boldsymbol{h} \in \text{Null}(\boldsymbol{A})$. Note that $\text{Null}(\boldsymbol{A})$ is a linear subspace of dimension $K = N - \text{rank}(\boldsymbol{A})$. If $\boldsymbol{Q}$ is a basis for this space, we can write and $\boldsymbol{h} \in \text{Null}(\boldsymbol{A})$ as $\boldsymbol{h} = \boldsymbol{Q}\boldsymbol{w}$. Using this we can re-write the program above as

$$\underset{\boldsymbol{w}\in\mathbb{R}^K}{\text{minimize}} \;\; f(\boldsymbol{x}_0 + \boldsymbol{Q}\boldsymbol{w}).$$

Sometimes this method can be very helpful, but note that computing $\boldsymbol{x}_0$ and $\boldsymbol{Q}$ could potentially be expensive.

Projected gradient descent

Now suppose that we wish to solve the constrained optimization problem

$$\underset{\boldsymbol{x}\in\mathcal{C}}{\text{minimize}} \;\; f(\boldsymbol{x})$$

where f is a differentiable convex function and $\mathcal{C}$ is a convex set in $\mathbb{R}^N$. Another way to express this problem is as the unconstrained problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + I_{\mathcal{C}}(\boldsymbol{x}), \tag{1}$$

where $I_{\mathcal{C}}$ denotes the indicator function for the set $\mathcal{C}$. We have previously encountered this idea in the context of duality, but in terms of suggesting practical algorithms, this has some obvious shortcomings. Namely, since $I_{\mathcal{C}}(\boldsymbol{x})$ is non-differentiable, we cannot apply gradient-based methods to solving (1).

However, we have encountered some algorithms for minimizing non-smooth convex functions. One that might seem particularly well-aligned with (1) is the **proximal gradient method**. Recall that proximal gradient method applies when our objective function consists of the sum of a smooth term (in this case, f) and a nonsmooth term (in this case, $I_{\mathcal{C}}$), resulting in the core iteration of

$$\boldsymbol{x}_{k+1} = \text{prox}_{\alpha_k I_{\mathcal{C}}}(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)).$$

This would yield a tractable algorithm with provable guarantees *if* we can compute $\text{prox}_{\alpha_k I_{\mathcal{C}}}$. So what does $\text{prox}_{\alpha_k I_{\mathcal{C}}}$ look like? Note that

$$\begin{aligned}
\text{prox}_{\alpha_k I_{\mathcal{C}}}(\boldsymbol{z}) &= \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min} \left(I_{\mathcal{C}}(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \right) \\
&= \underset{\boldsymbol{x}\in\mathcal{C}}{\arg\min} \;\; \|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \\
&= \mathcal{P}_{\mathcal{C}}(\boldsymbol{z}),
\end{aligned}$$

where $\mathcal{P}_{\mathcal{C}}(\boldsymbol{z})$ denotes the **projection** of $\boldsymbol{z}$ onto the set $\mathcal{C}$. Note that this holds for any $\alpha_k > 0$.

Thus, the core iteration of the proximal gradient method is equivalent to

$$\boldsymbol{x}_{k+1} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)).$$

That is, at each iteration we take a gradient step on f and then re-project onto the constraint set. Notice that for any $k \geq 1$, we are always guaranteed that $\boldsymbol{x}_k$ is feasible.

This algorithm is usually called **projected gradient descent**. It is a very simple (but often effective) method for solving constrained optimization problems when the projection onto the constraint set $\mathcal{C}$ can be computed efficiently. Note, however, that this is not always the case – sometimes computing this projection itself requires solving a challenging optimization problem.

Of course projected gradient descent inherits the convergence guarantees for the proximal gradient method. In particular, if f is L-smooth, then we know that the iterates obey

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{L}{2k} \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2$$

Other convergence results mirror those for unconstrained gradient descent. In particular, if f is smooth and strongly convex, then projected gradient descent has linear convergence. A full set of results (along with detailed proofs) can be found in [Lan20, Chap. 3].

Example: Least-squares with positivity constraints

Suppose we want to solve

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \boldsymbol{x} \geq \boldsymbol{0}.$$

This is a case where the projection onto the constraint set is relatively simple. It is easy to argue that the projection onto the set of all positive vectors is to simply just set all of the negative entries to zero. The projected gradient descent iteration is then

$$\boldsymbol{x}_{k+1} = \Big(\boldsymbol{x}_k + \alpha_k \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_k)\Big)_+,$$

where

$$(\boldsymbol{z})_+[i] = \begin{cases} z[i], & z[i] \geq 0, \\ 0, & z[i] < 0. \end{cases}$$

Barrier methods

Another popular and even more flexible approach are **barrier methods**. These can be thought of as again replacing our constrained problem with the unconstrained one in (1), but rather than attempting to minimize (1) directly, we instead solve a slight perturbation of this problem. In particular, we replace the indicator function $I_{\mathcal{C}}$ with some function b such that $\operatorname{dom} b = \mathcal{C}$ and $b(\boldsymbol{x}) \to \infty$ as $\boldsymbol{x}$ approaches the boundary of $\mathcal{C}$.

To make this concrete, consider the constrained program

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad g_m(\boldsymbol{x}) \leq 0, \quad m = 1, \ldots, M.$$

In a barrier method we replace this with the unconstrained program

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; f(\boldsymbol{x}) + \sum_{m=1}^{M} B(g_m(\boldsymbol{x})),$$

where $\operatorname{dom} B = \mathbb{R}_-$ and $B(x) \to \infty$ as $x \to 0$ from the left. Again, unless B is the indicator function, the new program is an approximation to the original.

An interesting choice is $B(x) = -\frac{1}{\tau}\log(-x)$. This particular barrier function has the properties:

- You can analyze the number of Newton iterations needed for convergence for many f of interest (using self-concordance). This is done very nicely in Chapters 9 and 11 of [BV04].
- The solution[1] $\boldsymbol{x}^\star(\tau)$ of

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; \tau f(\boldsymbol{x}) - \sum_{m=1}^{M} \log(-g_m(\boldsymbol{x}))$$

  can be used to generate a dual-feasible point (and hence a primal-dual gap certificate), and can be related to the KKT conditions for the original program.

To appreciate the second point above, we start by taking the gradient of the objective function above and setting it equal to zero. We see that

$$\tau \nabla f(\boldsymbol{x}^\star(\tau)) + \sum_{m=1}^{M} -\frac{1}{g_m(\boldsymbol{x}^\star(\tau))} \nabla g_m(\boldsymbol{x}^\star(\tau)) = \mathbf{0}.$$

So if we take

$$\lambda_m^\star(\tau) = -\frac{1}{\tau g_m(\boldsymbol{x}^\star(\tau))}, \;\; m = 1, \ldots, M,$$

we have $\lambda_m^\star(\tau) \geq 0$ and

$$f(\boldsymbol{x}^\star(\tau)) + \sum_{m=1}^{M} \lambda_m^\star(\tau) \nabla g_m(\boldsymbol{x}^\star(\tau)) = \mathbf{0}.$$

[1] We have multiplied the objective by τ to make some of what follows a little easier to express.

Since $\boldsymbol{x}^\star(\tau)$ is primal feasible, the only KKT condition we are missing is complementary slackness – we have replaced the condition

$$\lambda_m^\star g_m(\boldsymbol{x}^\star) = 0, \quad \text{with} \quad \lambda_m^\star(\tau) g_m(\boldsymbol{x}^\star(\tau)) = -1/\tau.$$

So if we set τ to be increasingly large, we obtain points that satisfy an increasingly tight approximation to the KKT conditions.

With this choice of $\boldsymbol{\lambda}^\star(\tau)$, we can also easily compute the dual of the original program:

$$\begin{aligned} d(\boldsymbol{\lambda}^\star(\tau)) &= \inf_{\boldsymbol{x}} \left(f(\boldsymbol{x}) + \sum_{m=1}^M \lambda_m^\star(\tau) g_m(\boldsymbol{x}) \right) \\ &= f(\boldsymbol{x}^\star(\tau)) + \sum_{m=1}^M \lambda_m^\star(\tau) g_m(\boldsymbol{x}^\star(\tau)) \\ &= f(\boldsymbol{x}^\star(\tau)) - M/\tau. \end{aligned}$$

Hence, we know that if $\boldsymbol{x}^\star$ is a solution to the original program, then

$$f(\boldsymbol{x}^\star(\tau)) - f(\boldsymbol{x}^\star) \;\leq\; f(\boldsymbol{x}^\star(\tau)) - d(\boldsymbol{\lambda}^\star(\tau)) \;\leq\; M/\tau.$$

Thus, we know that solving the log-barrier problem gets us within M/τ of the optimal of the original primal objective.

A full discussion of log barrier methods, including some fundamental complexity analysis, can be found in [BV04, Chap. 11]. One interesting theoretical result there is that, with a reasonable way of adjusting τ (multiplying it by 10 at every iteration, for example), the number of log-barrier iterations to make the value of the barrier functional $f(\boldsymbol{x}^\star(\tau))$ agree with the minimal value $p^\star$ to the original constrained problem to some precision. The upshot is that there is a very close match after $\sim \sqrt{M}$ iterations. This means that in theory, solving a

constrained problem is roughly as expensive as solving $\sqrt{M}$ unconstrained problems. In practice, it is actually much cheaper – standard log barrier iterations take maybe 20–50 iterations to produce good results.

Primal dual interior point methods

These are closely related to log barrier algorithms, but they take a more direct approach towards "solving" the KKT conditions. The general idea is to treat the KKT conditions like a set of nonlinear equations, and solve them using Newton's method.

We start with the same set of relaxed KKT conditions[2] we used with log barrier:

$$\boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla f(\boldsymbol{x}) + \sum_m \lambda_m \nabla g_m(\boldsymbol{x}) \\ -\lambda_1 g_1(\boldsymbol{x}) - 1/\tau \\ \vdots \\ -\lambda_m g_m(\boldsymbol{x}) - 1/\tau \end{bmatrix}$$

If we find $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ such that the $N+M$-vector $\boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{0}$, then we know we have found the same $\boldsymbol{x}^\star(\tau)$, $\boldsymbol{\lambda}^\star(\tau)$ that solve the log barrier problem.

Primal-dual interior point methods take Newton steps to try to make $\boldsymbol{r}_\tau = \boldsymbol{0}$, but they adjust τ at every step. The Newton step is characterized by

$$\boldsymbol{r}_\tau(\boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{\lambda} + \delta\boldsymbol{\lambda}) \;\approx\; \boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}) + \boldsymbol{J}_r(\boldsymbol{x}, \boldsymbol{\lambda}) \begin{bmatrix} \delta\boldsymbol{x} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = 0,$$

[2] We are again only considering inequality constraints; it is straightforward to modify everything we say here to include linear equality constraints.

where $\boldsymbol{J}_{\boldsymbol{r}_\tau}(\boldsymbol{x}, \boldsymbol{\lambda})$ is the **Jacobian** matrix for the vector-valued function $\boldsymbol{r}_\tau$ given by

$$\begin{bmatrix} \nabla^2 f(\boldsymbol{x}) + \sum_m \lambda_m \nabla^2 g_m(\boldsymbol{x}) & \nabla g_1(\boldsymbol{x}) & \nabla g_2(\boldsymbol{x}) & \cdots & \nabla g_M(\boldsymbol{x}) \\ -\lambda_1 \nabla g_1(\boldsymbol{x})^{\mathrm{T}} & -g_1(\boldsymbol{x}) & 0 & \cdots & 0 \\ -\lambda_2 \nabla g_2(\boldsymbol{x})^{\mathrm{T}} & 0 & -g_2(\boldsymbol{x}) & \cdots & 0 \\ \vdots & & & \ddots & \\ -\lambda_M \nabla g_M(\boldsymbol{x})^{\mathrm{T}} & 0 & 0 & \cdots & -g_M(\boldsymbol{x}) \end{bmatrix}.$$

The update direction is

$$\begin{bmatrix} \delta \boldsymbol{x} \\ \delta \boldsymbol{\lambda} \end{bmatrix} = -\boldsymbol{J}_{\boldsymbol{r}_\tau}^{-1} \boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}).$$

With this direction, we can perform a line search. The parameter τ is updated at every step (getting larger) based on an estimate of the duality gap.

A key feature of this type of primal dual method is that the iterates $\boldsymbol{x}_k$ and $\boldsymbol{\lambda}_k$ do not have to be feasible (although they of course become feasible in the limit).

Details on this particular algorithm, along with a full convergence analysis, can be found in [BV04, Chap. 11.7].

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Lan20] G. Lan. *First-order and Stochastic Optimization Methods for Machine Learning*. Springer, 2020.

Alternating direction primal dual methods

We will now focus on a class of algorithms that work by fixing the dual variables and updating the primal variables $\boldsymbol{x}$, then fixing the primals and updating the dual variables $\boldsymbol{\lambda}, \boldsymbol{\nu}$. An excellent source for this material is [BPC+10]. In fact, what follows here is basically a summary of the first 12 pages of that paper.

We have seen that when we have strong duality (which we will assume throughout), the optimal value of the primal program is equal to the optimal value of the dual program. That is, if $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star$ are primal/dual optimal points,

$$\begin{aligned} f(\boldsymbol{x}^\star) &= d(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star) \\ &= \inf_{\boldsymbol{x} \in \mathbb{R}^N} \ \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star), \end{aligned}$$

where $\mathcal{L}$ is the Lagrangian

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m g_m(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}).$$

If $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ has only one minimizer,[1] then we can recover the primal optimal solution $\boldsymbol{x}^\star$ from the dual-optimal solution $\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star$ by solving the *unconstrained* program

$$\boldsymbol{x}^\star = \arg\min_{\boldsymbol{x} \in \mathbb{R}^N} \ \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star).$$

"Alternating" methods search for a saddle point of the Lagrangian by fixing the dual variables $\boldsymbol{\lambda}_k, \boldsymbol{\nu}_k$, minimizing $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\nu}_k)$ with respect to $\boldsymbol{x}$, then updating the Lagrange multipliers.

[1] Which is the case when f is strictly convex, and in many other situations.

To start, we will base our discussion on **equality constrained** problems. Incorporating inequality constraints will be natural after we have developed things a bit.

Dual ascent

We want to solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

We will assume that the domain of f is all of $\mathbb{R}^N$; again, things are easily modified if this is any open set. The Lagrangian is

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}),$$

and the dual function is

$$\begin{aligned} d(\boldsymbol{\nu}) &= \inf_{\boldsymbol{x}} \;\; \mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu}) \\ &= \inf_{\boldsymbol{x}} \;\; f(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{b} \\ &= -\sup_{\boldsymbol{x}} \Big((-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu})^{\mathrm{T}}\boldsymbol{x} - f(\boldsymbol{x}) \Big) - \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{b} \\ &= -f^*(-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}) - \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{b}, \end{aligned}$$

and the dual problem is

$$\underset{\boldsymbol{\nu}\in\mathbb{R}^N}{\text{maximize}} \;\; d(\boldsymbol{\nu}).$$

Consider for a moment the problem of maximizing the dual. A reasonable thing to do would be some kind of gradient ascent:[2]

$$\boldsymbol{\nu}_{k+1} = \boldsymbol{\nu}_k + \alpha_k \nabla d(\boldsymbol{\nu}_k),$$

[2] "Ascent" instead of "descent" because g is concave instead of convex.

where α_k is some appropriate step size. The gradient of d (with respect to $\boldsymbol{\nu}$) at a point $\boldsymbol{\nu}_0$ is

$$\nabla d(\boldsymbol{\nu}_0) = \nabla \inf_{\boldsymbol{x}} \ \left(f(\boldsymbol{x}) + \boldsymbol{\nu}_0^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \right),$$

and so if $\boldsymbol{x}^+ = \arg\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) + \boldsymbol{\nu}_0^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$, then

$$\begin{aligned} \nabla d(\boldsymbol{\nu}_0) &= \nabla_{\boldsymbol{\nu}} \left(f(\boldsymbol{x}^+) + \boldsymbol{\nu}_0^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}^+ - \boldsymbol{b}) \right) \\ &= \boldsymbol{A}\boldsymbol{x}^+ - \boldsymbol{b}. \end{aligned}$$

This leads naturally to:

The **dual ascent** algorithm consists of the iteration

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} \ \mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu}_k) \\ \boldsymbol{\nu}_{k+1} &= \boldsymbol{\nu}_k + \alpha_k(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}) \end{aligned}$$

that is repeated until some convergence criteria is met.

This algorithm "works" under certain assumptions on f (that translate to different assumptions on the dual d). In particular, we need $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu})$ to be bounded for every $\boldsymbol{\nu}$, otherwise the primal update $\boldsymbol{x}_{k+1} = \arg\min_{\boldsymbol{x}} \ \mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu}_k)$ can fail.

That the Lagrangian is bounded below for every choice of $\boldsymbol{\nu}$ is far from a given. Looking at

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}),$$

we can see that if f increases slowly (sublinearly) in any direction even partially aligned with the row space of $\boldsymbol{A}$, then we can find a sequence of $\boldsymbol{x}$ that drive $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\nu}) \to -\infty$ for certain $\boldsymbol{\nu}$.

Of course, this algorithm is nicest when we can solve the unconstrained primal update problem efficiently.

The Method of Multipliers and Augmented Lagrangians

The method of multipliers (MoM) is the same idea as dual ascent, but we smooth out (augment) the Lagrangian to make the primal update more robust.

It should be clear that

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

and

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

have exactly the same set of solutions for all $\rho \geq 0$.

The Lagrangian for the second program is

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}).$$

This is called the **augmented Lagrangian** of the original problem.

Adding the quadratic term is nice – it makes (under mild conditions on f with respect to $\boldsymbol{A}$) the primal update minimization well-posed (i.e., makes the dual differentiable).

Notice that the Lagrange multipliers $\boldsymbol{\nu}$ appear in exactly the same way in the augmented Lagrangian as they do in the regular Lagrangian, so the dual update does not change.

The resulting algorithm is called the **method of multipliers**; we iterate

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} \ \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\nu}_k) \\ \boldsymbol{\nu}_{k+1} &= \boldsymbol{\nu}_k + \rho(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b})\end{aligned}$$

until some convergence criteria is met.

As a bonus, we now have a principled way of selecting the step size for the dual update – just use ρ. To see why this makes sense, recall the KKT conditions for $\boldsymbol{x}^\star$ and $\boldsymbol{\nu}^\star$ to be a solution:

$$\boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{b}, \quad \nabla f(\boldsymbol{x}^\star) + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}^\star = \boldsymbol{0}.$$

With ρ as the step size, we have

$$\begin{aligned}\boldsymbol{0} &= \nabla \mathcal{L}_\rho(\boldsymbol{x}_{k+1}, \boldsymbol{\nu}_k), \quad (\text{since } \boldsymbol{x}_{k+1} \text{ is a minimizer}), \\ &= \nabla f(\boldsymbol{x}_{k+1}) + \boldsymbol{A}^{\mathrm{T}} \left(\boldsymbol{\nu}_k + \rho(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b})\right) \\ &= \nabla f(\boldsymbol{x}_{k+1}) + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\nu}_{k+1}.\end{aligned}$$

So the dual update maintains the second optimality condition at every step.

The MoM has much better convergence properties than dual ascent. The algorithm we look at next, the alternating direction method of multipliers (ADMM), will build on this idea in a way that makes it applicable to functions that have a nonsmooth component and can be easily modified to incorporate certain kinds of inequality constraints.

Alternating direction method of multipliers (ADMM)

ADMM **splits** the optimization variables into two parts, $\boldsymbol{x}$ and $\boldsymbol{z}$, and solves programs of the form

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c}.$$

The basic idea is to rotate through 3 steps:

1. Minimize the (augmented) Lagrangian over $\boldsymbol{x}$ with $\boldsymbol{z}$ and the Lagrange multipliers $\boldsymbol{\nu}$ fixed.
2. Minimize the (augmented) Lagrangian over $\boldsymbol{z}$ with $\boldsymbol{x}$ and $\boldsymbol{\nu}$ fixed.
3. Update the Lagrange multipliers using gradient ascent as before.

If the splitting is done in a careful manner, it can happen that each of the subproblems above can be easily computed. We can also handle general convex constraints (more on this later).

To make the three steps above more explicit: the augmented Lagrangian is

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}\|_2^2,$$

and the general ADMM iteration is

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} \;\; \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{z}_k, \boldsymbol{\nu}_k) \\
\boldsymbol{z}_{k+1} &= \arg\min_{\boldsymbol{z}} \;\; \mathcal{L}_\rho(\boldsymbol{x}_{k+1}, \boldsymbol{z}, \boldsymbol{\nu}_k) \\
\boldsymbol{\nu}_{k+1} &= \boldsymbol{\nu}_k + \rho(\boldsymbol{A}\boldsymbol{x}_{k+1} + \boldsymbol{B}\boldsymbol{z}_{k+1} - \boldsymbol{c}).
\end{aligned}$$

The only real difference between ADMM and MoM is the we are splitting the primal minimization into two parts instead of optimizing over $(\boldsymbol{x}, \boldsymbol{z})$ jointly.

Scaled form.

We can write the ADMM iterations in a more convenient form by substituting

$$\boldsymbol{\mu} = \frac{1}{\rho}\boldsymbol{\nu}.$$

By "completing the square" we have that

$$\boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c})+\frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}\|_2^2 = \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}+\boldsymbol{\mu}\|_2^2-\frac{\rho}{2}\|\boldsymbol{\mu}\|_2^2,$$

and so we can write:

ADMM:

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} \left(f(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z}_k - \boldsymbol{c} + \boldsymbol{\mu}_k\|_2^2 \right) \\
\boldsymbol{z}_{k+1} &= \arg\min_{\boldsymbol{z}} \left(g(\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}_{k+1} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} + \boldsymbol{\mu}_k\|_2^2 \right) \\
\boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{A}\boldsymbol{x}_{k+1} + \boldsymbol{B}\boldsymbol{z}_{k+1} - \boldsymbol{c}
\end{aligned}$$

Example: The LASSO

Recall the LASSO:

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\tau\|\boldsymbol{x}\|_1.$$

Taking

$$f(\boldsymbol{x})=\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2 \quad \text{and} \quad g(\boldsymbol{z})=\tau\|\boldsymbol{z}\|_1,$$

we can rewrite this in ADMM form as

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \ f(\boldsymbol{x})+g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x}-\boldsymbol{z}=\boldsymbol{0}.$$

The $\boldsymbol{x}$ update is

$$\boldsymbol{x}_{k+1}=\arg\min_{\boldsymbol{x}}\left(\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\frac{\rho}{2}\|\boldsymbol{x}-\boldsymbol{z}_k+\boldsymbol{\mu}_k\|_2^2\right).$$

With both $\boldsymbol{z}_k$ and $\boldsymbol{\mu}_k$ fixed, this is a regularized least-squares problem and is equivalent to:

$$\min_{\boldsymbol{x}} \ \left\|\begin{bmatrix}\boldsymbol{A}\\ \sqrt{\rho}\,\mathbf{I}\end{bmatrix}\boldsymbol{x}-\begin{bmatrix}\boldsymbol{b}\\ \sqrt{\rho}(\boldsymbol{z}_k-\boldsymbol{\mu}_k)\end{bmatrix}\right\|_2^2.$$

This problem has a closed-form solution:

$$\begin{aligned}\boldsymbol{x}_{k+1}&=\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}+\rho\mathbf{I}\right)^{-1}\begin{bmatrix}\boldsymbol{A}^{\mathrm{T}} & \sqrt{\rho}\,\mathbf{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{b}\\ \sqrt{\rho}(\boldsymbol{z}_k-\boldsymbol{\mu}_k)\end{bmatrix}\\ &=\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}+\rho\mathbf{I}\right)^{-1}\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}+\rho(\boldsymbol{z}_k-\boldsymbol{\mu}_k)\right)\end{aligned}$$

The $\boldsymbol{z}$ update problem is:

$$\underset{\boldsymbol{z}}{\text{minimize}} \ \tau\|\boldsymbol{z}\|_1 + \frac{\rho}{2}\|\boldsymbol{z} - \boldsymbol{x}_{k+1} - \boldsymbol{\mu}_k\|_2^2.$$

You may recognize this: it is the proximal operator for the ℓ_1-norm, which as we have seen before has closed form solution:

$$\boldsymbol{z}_{k+1} = T_{\tau/\rho}(\boldsymbol{x}_{k+1} + \boldsymbol{\mu}_k),$$

where $T_\lambda(\cdot)$ is the term-by-tern soft-thresholding operator,

$$(T_\lambda(\boldsymbol{v}))[n] = \begin{cases} v[n] - \lambda, & v[n] > \lambda, \\ 0, & |v[n]| \leq \lambda, \\ v[n] + \lambda, & v[n] < -\lambda. \end{cases}$$

To summarize:

ADMM iterations for the LASSO

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \rho\mathbf{I}\right)^{-1}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} + \rho(\boldsymbol{z}_k - \boldsymbol{\mu}_k)), \\ \boldsymbol{z}_{k+1} &= T_{\tau/\rho}(\boldsymbol{x}_{k+1} + \boldsymbol{\mu}_k), \\ \boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{x}_{k+1} - \boldsymbol{z}_{k+1}. \end{aligned}$$

Convergence properties

We will state one convergence result. If the following two conditions hold:

1. f and g are closed, proper, and convex (i.e., their epigraphs are nonempty closed convex sets),
2. strong duality holds,

then

- $\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{z}_k - \boldsymbol{c} \to \boldsymbol{0}$ as $k \to \infty$. That is, the primal iterates are asymptotically feasible.
- $f(\boldsymbol{x}_k) + g(\boldsymbol{z}_k) \to f(\boldsymbol{x}^\star) + g(\boldsymbol{x}^\star)$ as $k \to \infty$. That is, the value of the objective function approaches the optimal value asymptotically.
- $\boldsymbol{\nu}_k \to \boldsymbol{\nu}^\star$ as $k \to \infty$, where $\boldsymbol{\nu}^\star$ is a dual optimal point.

Under additional assumptions, we can also have convergence to a primal optimal point, $(\boldsymbol{x}_k, \boldsymbol{z}_k) \to (\boldsymbol{x}^\star, \boldsymbol{z}^\star)$ as $k \to \infty$.

See [BPC+10, Section 3.2] for further discussion and references.

Convex constraints

Using a technique that we have seen before, we can write the general program

$$\underset{\boldsymbol{x}\in\mathcal{C}}{\text{minimize}} \;\; f(\boldsymbol{x}),$$

where $\mathcal{C}$ is a closed convex set, in ADMM form as

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0},$$

where $g(\boldsymbol{z})$ is the indicator function for $\mathcal{C}$:

$$g(\boldsymbol{z}) = \begin{cases} 0, & \boldsymbol{z} \in \mathcal{C}, \\ \infty, & \boldsymbol{z} \notin \mathcal{C}. \end{cases}$$

Note that in this case, the $\boldsymbol{z}$ update is a closest-point-to-a-convex-set problem. For fixed $\boldsymbol{v} \in \mathbb{R}^N$:

$$\arg\min_{\boldsymbol{z}} \;\; g(\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{z} - \boldsymbol{v}\|_2^2 = \arg\min_{\boldsymbol{z}\in\mathcal{C}} \;\; \|\boldsymbol{z} - \boldsymbol{v}\|_2 = \mathcal{P}_{\mathcal{C}}(\boldsymbol{v}).$$

ADMM iteration for general convex constraints:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} \left(f(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{z}_k + \boldsymbol{\mu}_k\|_2^2 \right), \\ \boldsymbol{z}_{k+1} &= \mathcal{P}_{\mathcal{C}}\left(\boldsymbol{x}_{k+1} + \boldsymbol{\mu}_k\right), \\ \boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{x}_{k+1} - \boldsymbol{z}_{k+1}. \end{aligned}$$

Of course, this algorithm is most attractive when we have a fast method for computing $\mathcal{P}_{\mathcal{C}}(\cdot)$.

Example: Basis Pursuit

As we have seen before, a good proxy for finding the sparsest solution to an underdetermined system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is to solve

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

To put this in ADMM form, we are solving

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \ f(\boldsymbol{x}) + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0},$$

with

$$f(\boldsymbol{x}) = \|\boldsymbol{x}\|_1, \quad \text{and} \quad g(\boldsymbol{z}) = \begin{cases} 0, & \boldsymbol{A}\boldsymbol{z} = \boldsymbol{b}, \\ \infty, & \text{otherwise.} \end{cases}$$

The projection onto $\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ can be given in closed form using the pseudo-inverse c of $\boldsymbol{A}$ as

$$\begin{aligned} \mathcal{P}_{\mathcal{C}}(\boldsymbol{v}) &= \boldsymbol{A}^{\dagger}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{v}) + \boldsymbol{v} \\ &= (\mathbf{I} - \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{A})\boldsymbol{v} + \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b}, \end{aligned}$$

where the last equality comes from $\boldsymbol{A}^{\dagger} = \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}$ when $\boldsymbol{A}$ has full row rank.

The updates in this case are

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} \left(\|\boldsymbol{x}\|_1 + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{z}_k + \boldsymbol{\mu}_k\|_2^2 \right) \\ &= T_{1/\rho}(\boldsymbol{z}_k - \boldsymbol{\mu}_k) \\ \boldsymbol{z}_{k+1} &= (\mathbf{I} - \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{A})(\boldsymbol{x}_{k+1} + \boldsymbol{\mu}_k) + \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b} \\ \boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{x}_{k+1} - \boldsymbol{z}_{k+1}. \end{aligned}$$

Example: Linear programming

Consider the general linear program

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \ \ \boldsymbol{x} \geq \boldsymbol{0},$$

where $\boldsymbol{A}$ is an $M \times N$ matrix with full row rank.[3] We can put this in ADMM form by first eliminating the equality constraints, then introducing the indicator function for the non-negativity constraint.

Let $\boldsymbol{Q}$ be an $N \times (N - M)$ matrix whose columns span $\text{Null}(\boldsymbol{A})$, and let $\boldsymbol{x}_0$ be any point such that $\boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{b}$. Then we can re-write the LP as

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \boldsymbol{c}^{\mathrm{T}}(\boldsymbol{x}_0 + \boldsymbol{Q}\boldsymbol{w}) \quad \text{subject to} \quad \boldsymbol{x}_0 + \boldsymbol{Q}\boldsymbol{w} \geq \boldsymbol{0},$$

which we can write in ADMM form as

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_0 + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{w} + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{Q}\boldsymbol{w} - \boldsymbol{z} = -\boldsymbol{x}_0,$$

where

$$g(\boldsymbol{z}) = \begin{cases} 0, & \boldsymbol{z} \geq \boldsymbol{0}, \\ \infty, & \text{otherwise.} \end{cases}$$

(We can drop the $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_0$ from the objective since it does not depend on either of the optimization variables.)

Notice that when $\boldsymbol{Q}$ has full column rank, the program

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \boldsymbol{v}^{\mathrm{T}}\boldsymbol{w} + \frac{1}{2}\|\boldsymbol{Q}\boldsymbol{w} - \boldsymbol{y}\|_2^2,$$

[3] The full row rank assumption is not at all essential; I am just making it to keep things streamlined.

has the closed-form solution

$$\boldsymbol{w}^\star = (\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})^{-1}(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{v}).$$

Also, the projection onto the non-negative orthant $\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{x} \geq \boldsymbol{0}\}$ is

$$\mathcal{P}_{\mathcal{C}}(\boldsymbol{v}) = (\boldsymbol{v})_+, \quad \text{or} \quad (\mathcal{P}_{\mathcal{C}}(\boldsymbol{v}))[n] = \begin{cases} v[n], & v[n] \geq 0, \\ 0, & v[n] < 0. \end{cases}$$

For the general linear program, then, the ADMM iterations are

$$\begin{aligned}
\boldsymbol{w}_{k+1} &= \arg\min_{\boldsymbol{w}} \left(\frac{1}{\rho}\boldsymbol{c}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{w} + \frac{1}{2}\|\boldsymbol{Q}\boldsymbol{w} - \boldsymbol{z}_k + \boldsymbol{x}_0 + \boldsymbol{\mu}_k\|_2^2 \right) \\
&= (\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})^{-1} \left[\boldsymbol{Q}^{\mathrm{T}}(\boldsymbol{z}_k - \boldsymbol{x}_0 - \boldsymbol{\mu}_k) - \frac{1}{\rho}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{c} \right], \\
\boldsymbol{z}_{k+1} &= P_{\mathcal{C}}(\boldsymbol{Q}\boldsymbol{w}_{k+1} + \boldsymbol{x}_0 + \boldsymbol{\mu}_k) \\
&= (\boldsymbol{Q}\boldsymbol{w}_{k+1} + \boldsymbol{x}_0 + \boldsymbol{\mu}_k)_+ \\
\boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{Q}\boldsymbol{w}_{k+1} - \boldsymbol{z}_{k+1} + \boldsymbol{x}_0.
\end{aligned}$$

Notice that especially when the columns of $\boldsymbol{Q}$ are orthogonal, $\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q} = \mathbf{I}$, all of these steps are very simple.

References

[BPC+10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.

A Primal-Dual Proximal Algorithm

We will close this section by discussion one more primal-dual algorithm that is related to, but not the same as, ADMM. It is inspired by solving unconstrained problems of the form

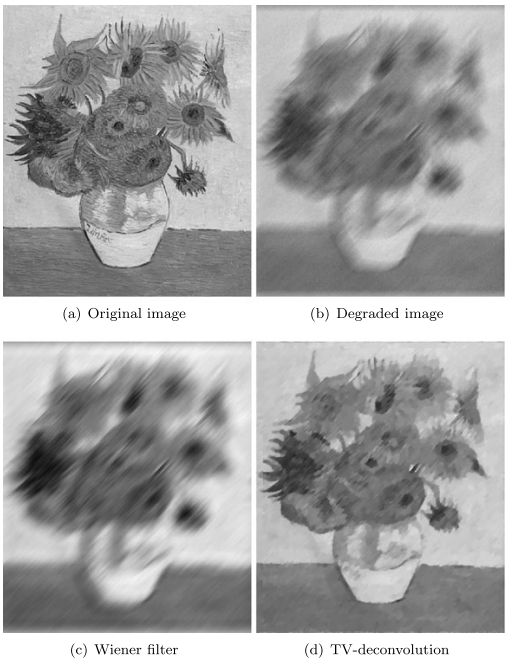
$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{K}\boldsymbol{x}) + g(\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{K}$ is a $M \times N$ matrix, and f and g are convex (but generally not smooth or even differentiable) convex functions. The "primal-dual" part comes from the way we will decompose this problem.

We looked at problems of a form similar to (1) when we talked about proximal gradient algorithms. There we saw that if f was differentiable and we had a good way to compute the prox operator for g, then we take a gradient step on $f(\boldsymbol{K}\boldsymbol{x})$ (using the chain rule to account for the $\boldsymbol{K}$) and then a prox step on g. The algorithm we consider here applies when f is not differentiable (and g might be smooth or not). In fact, recent work on this problem was in part inspired by a common problem in image restoration, where we solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\ \|\boldsymbol{D}\boldsymbol{x}\|_1 + \frac{\delta}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2, \tag{2}$$

where $\boldsymbol{D}$ is an approximate derivative operator. The idea is that we are given some kind of indirect measurements $\boldsymbol{y} \approx \boldsymbol{A}\boldsymbol{x}$ and we want to find $\boldsymbol{x}$ that is consistent with these measurements and whose derivative is "sparse". Here is an example from [CP11]:



(a) Original image (b) Degraded image

(c) Wiener filter (d) TV-deconvolution

Fig. 8 Motion deblurring using total variation regularization. (**a**) and (**b**) show the 400×470 clean image and a degraded version containing motion blur of approximately 30 pixels and Gaussian noise of standard deviation $\sigma = 0.01$. (**c**) is the result of standard Wiener filtering. (**d**) is the result of the total variation based deconvolution method. Note that the TV-based method yields visually much more appealing results

The tricky thing about solving (2) is the $\boldsymbol{D}$ in the non-smooth part of the functional; we do not have a good prox operator for $\|\boldsymbol{D}\boldsymbol{x}\|_1$. We have seen that the prox operator for the ℓ_1 norm

$$\arg\min_{\boldsymbol{x}} \left(\|\boldsymbol{x}\|_1 + \frac{1}{2\alpha}\|\boldsymbol{x} - \boldsymbol{u}\|_2^2 \right)$$

can be solved using soft thresholding, but the prox operator

$$\arg\min_{\boldsymbol{x}} \left(\|\boldsymbol{D}\boldsymbol{x}\|_1 + \frac{1}{2\alpha}\|\boldsymbol{x} - \boldsymbol{u}\|_2^2 \right)$$

does not have an easy solution.

To account for the matrix $\boldsymbol{K}$ in (1), we turn again to the Fenchel conjugate. If f is convex (and has a closed epigraph) then we can write

$$f(\boldsymbol{K}\boldsymbol{x}) = \sup_{\boldsymbol{\nu}} \left(\boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K}\boldsymbol{x} - f^*(\boldsymbol{\nu}) \right).$$

We can then recast (1) as

$$\underset{\boldsymbol{x}}{\text{minimize}} \max_{\boldsymbol{\nu}} \underbrace{\boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{x} + g(\boldsymbol{x}) - f^*(\boldsymbol{\nu})}_{\phi(\boldsymbol{x}, \boldsymbol{\nu})}. \tag{3}$$

As ϕ is convex in $\boldsymbol{x}$ and concave in $\boldsymbol{\nu}$, this is a **saddle point problem**. We are trying to maximize (over $\boldsymbol{\nu}$) the function $\boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{x} - f^*(\boldsymbol{\nu})$ while minimizing (over $\boldsymbol{x}$) the function $g(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{x}$. The bilinear form $\boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{x}$ is what is tying these problems together.

We can solve it by alternating proximal gradient steps on the first and second variables of ϕ, first descending on $f^*(\boldsymbol{u}) - \boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{x}$ (which is the same as ascending on $\boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{x} - f^*(\boldsymbol{\nu})$), then descending on $g(\boldsymbol{x}) + \boldsymbol{\nu}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{x}$. From a starting point $\boldsymbol{x}_0, \boldsymbol{\nu}_0$, we iterate

$$\begin{aligned} \boldsymbol{\nu}_{k+1} &= \operatorname{prox}_{\sigma f^*}(\boldsymbol{\nu}_k + \sigma \boldsymbol{K} \boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \operatorname{prox}_{\alpha g}(\boldsymbol{x}_k - \alpha \boldsymbol{K}^{\mathrm{T}} \boldsymbol{\nu}_{k+1}). \end{aligned}$$

The σ above is the dual stepsize, while the α is the primal stepsize. This is called the Arrow-Hurwicz algorithm [AHU58], and it is known (under appropriate choices of σ, α) to find a saddle point $(\boldsymbol{x}^\star, \boldsymbol{\nu}^\star)$ of (3) (and so $\boldsymbol{x}^\star$ will solve (1)).

Note that the algorithm above involves the prox operators of f^* and g; this algorithm is of course most attractive when these operators can be computed efficiently. We will note that the prox operator for f^* is always as easy to compute as the prox operator of f, as

$$\boldsymbol{z} = \operatorname{prox}_{\alpha f}(\boldsymbol{z}) + \alpha \operatorname{prox}_{\alpha^{-1} f^*}(\boldsymbol{z}/\alpha),$$

for any convex f and any $\alpha > 0$. This is known as the *Moreau decomposition*; a proof can be found in [Bec17, Chapter 6.6].

There are all kinds of ways that this classic iteration can be enhanced. In a well-known paper [CP11], a detailed convergence analysis was given with a slight variation:

$$\begin{aligned}\boldsymbol{\nu}_{k+1} &= \text{prox}_{\sigma f^*}(\boldsymbol{\nu}_k + \sigma \boldsymbol{K}\boldsymbol{x}_k) \\ \boldsymbol{x}'_{k+1} &= \text{prox}_{\alpha g}(\boldsymbol{x}_k - \alpha \boldsymbol{K}^{\mathrm{T}}\boldsymbol{\nu}_{k+1}) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}'_{k+1} + \theta(\boldsymbol{x}'_{k+1} - \boldsymbol{x}'_k).\end{aligned}$$

The value of θ adds momentum to the primal variable, and in theory the convergence guarantees are nicest for $\theta = 1$. (Note that $\theta = 0$ reproduces the first algorithm above.) In practice, though, sometimes $\theta = 0$ works just as well if not better than $\theta = 1$ (or something else). Also in [CP11], the introduce (and analyze) other acceleration techniques similar in nature to accelerated proximal gradient.

Exercise: Write down, in closed form, each part of the iteration for solving (2).

Exercise: Write down the ADMM iteration for solving (2) with $\boldsymbol{z} = \boldsymbol{D}\boldsymbol{x}$ and compare with the above.

References

[AHU58] K. J. Arrow, L. Hurwicz, and H. Uzawa. *Studies in Linear and Non-Linear Programming*. Stanford University Press, 1958.

[Bec17] A. Beck. *First-order methods in optimization*. SIAM, 2017.

[CP11] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Imaging and Vision*, 40(1):120–145, 2011.

The spectral and nuclear norms

Let's take a closer look at the nuclear norm and spectral (operator norm) for matrices and the associated operators for optimization.

First, recall the standard inner produce on the space of $M \times N$ matrices

$$\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \sum_{m=1}^{M} \sum_{n=1}^{N} X_{m,n} Y_{m,n} = \text{trace}(\boldsymbol{Y}^{\mathrm{T}} \boldsymbol{X}).$$

The induced norm (called the 'Frobenius norm') is

$$\|\boldsymbol{X}\|_F^2 = \langle \boldsymbol{X}, \boldsymbol{X} \rangle = \sum_{m=1}^{M} \sum_{n=1}^{N} |X_{m,n}|^2.$$

Here are some basic facts that you can prove at home that we will use repeatedly below. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ are $M \times N$ matrices. If $\boldsymbol{A}$ is a $M \times M$ matrix, then

$$\langle \boldsymbol{A}\boldsymbol{X}, \boldsymbol{Y} \rangle = \langle \boldsymbol{X}, \boldsymbol{A}^{\mathrm{T}} \boldsymbol{Y} \rangle.$$

And if $\boldsymbol{B}$ is an $N \times N$ matrix, then

$$\langle \boldsymbol{X}\boldsymbol{B}, \boldsymbol{Y} \rangle = \langle \boldsymbol{X}, \boldsymbol{Y}\boldsymbol{B}^{\mathrm{T}} \rangle.$$

Now let $\boldsymbol{U}$ be a matrix with M rows and columns that are orthonormal, i.e. $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \mathbf{I}$. Then the above implies

$$\|\boldsymbol{U}^{\mathrm{T}} \boldsymbol{X}\|_F^2 = \|\boldsymbol{X}\|_F^2.$$

Similarly, if $\boldsymbol{V}$ is a matrix with N rows and columns that are orthonormal ($\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = \mathbf{I}$), then

$$\|\boldsymbol{X}\boldsymbol{V}\|_F^2 = \|\boldsymbol{X}\|_F^2.$$

It is now easy to show that $\|\boldsymbol{X}\|_F^2$ is also the sum of the squared singular values of $\boldsymbol{X}$. Let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$ be the singular value decomposition (SVD) of $\boldsymbol{X}$; $\boldsymbol{U}$ is a $M \times R$ matrix with orthonormal columns, $\boldsymbol{V}$ is an $N \times R$ matrix with orthonormal coliumns, and $\boldsymbol{\Sigma}$ is an $R \times R$ matrix with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_R > 0$ along its diagonal (where R is the rank of $\boldsymbol{X}$). Then

$$\|\boldsymbol{X}\|_F^2 = \|\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}\|_F^2 = \|\boldsymbol{\Sigma}\|_F^2 = \sum_{i=1}^{R} \sigma_r^2,$$

The Frobenius norm is analogous to the Euclidean norm in two different ways: it is the sum of the squares of the entries in $\boldsymbol{X}$ and also the sum of the squares of the singular values. The spectral norm (or operator norm) is the maximum (ℓ_∞) norm of the singular values

$$\|\boldsymbol{X}\| = \max_{\|\boldsymbol{v}\|_2 \leq 1} \|\boldsymbol{X}\boldsymbol{v}\|_2 = \sigma_1.$$

(This matrix norm is so fundamental, that usually it is just presented without subscript or additional notation.)

The nuclear norm is the sum (ℓ_1 norm) of the singular values

$$\|\boldsymbol{X}\|_{\mathrm{nn}} = \sum_{r=1}^{R} \sigma_r.$$

As we will see below, the nuclear norm is the dual norm to the spectral norm, so often times it is written simply as $\|\boldsymbol{X}\|_*$ in the literature.

As $\|\cdot\|_F$ is an induced norm, it obeys the Cauchy-Schwarz inequality, i.e.

$$|\langle \boldsymbol{X}, \boldsymbol{Y} \rangle| \ \leq \ \|\boldsymbol{X}\|_F \cdot \|\boldsymbol{Y}\|_F,$$

with equality if and only if $\boldsymbol{Y} = \alpha \boldsymbol{X}$ for some scalar α. For matrices, there is actually a refined version of Cauchy-Schwarz, called the *Fan inequality* or the *von Neumann inequality* depending on who you are talking to. If $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_R$ be the singular values of $\boldsymbol{X}$ and $\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_R$ be the singular values of $\boldsymbol{Y}$ (just let $R = \min(M, N)$ and set any singular values past the rank of $\boldsymbol{X}$ or $\boldsymbol{Y}$ equal to zero). Then

$$|\langle \boldsymbol{X}, \boldsymbol{Y} \rangle| \leq \sum_{r=1}^{R} \sigma_r \gamma_r,$$

with equality if and only if $\boldsymbol{X}$ and $\boldsymbol{Y}$ have the same singular vectors corresponding to the singular values in order. We will not prove this here; see [lM75] for a proof[1]. Note that we call this a refinement because it gives us something in the middle of the stand Cauchy-Schwarz inequality, viz

$$|\langle \boldsymbol{X}, \boldsymbol{Y} \rangle| \leq \sum_{r=1}^{R} \sigma_r \gamma_r \leq \left(\sum_{r=1}^{R} \sigma_r^2 \right)^{1/2} \left(\sum_{r=1}^{R} \gamma_r^2 \right)^{1/2} = \|\boldsymbol{X}\|_F \|\boldsymbol{Y}\|_F.$$

[1] A pdf can be found here: https://link.springer.com/content/pdf/10.1007/BF01647331.pdf.

Exercise: Let $\boldsymbol{X}_0$ be an arbitrary $M \times N$ matrix with SVD $\boldsymbol{X}_0 = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$. Show that the closest point problem

$$\underset{\|\boldsymbol{Y}\| \leq \tau}{\text{minimize}} \ \|\boldsymbol{X}_0 - \boldsymbol{Y}\|_F$$

is solved by $\hat{\boldsymbol{X}} = \boldsymbol{U}\hat{\boldsymbol{\Sigma}}\boldsymbol{V}^{\mathrm{T}}$, where

$$\hat{\sigma}_r = \begin{cases} \tau, & \sigma_r \geq \tau \\ \sigma_r, & \sigma_r \leq \tau. \end{cases}$$

(The $\{\sigma_r\}$ are the singular values of $\boldsymbol{X}$ along the diagonal of $\boldsymbol{\Sigma}$ while the $\{\hat{\sigma}_r\}$ are the singular values of $\hat{\boldsymbol{X}}$ along the diagonal of $\hat{\boldsymbol{\Sigma}}$.) You can do this by showing

$$\langle \boldsymbol{X}_0 - \hat{\boldsymbol{X}}, \boldsymbol{Y} - \hat{\boldsymbol{X}} \rangle \leq 0, \quad \text{for all} \quad \|\boldsymbol{Y}\| \leq \tau.$$

Exercise: Show that the nuclear norm is the dual norm to the spectral norm. That is, show that for any $M \times N$ matrix $\boldsymbol{X}$,

$$\|\boldsymbol{X}\|_{\text{nn}} = \sup_{\|\boldsymbol{Y}\| \leq 1} \langle \boldsymbol{X}, \boldsymbol{Y} \rangle.$$

Exercise: Find a simple expression for the prox operator for $\|\cdot\|_{\text{nn}}$. That is, find a closed form solution to

$$\text{prox}_{\alpha\|\cdot\|_{\text{nn}}}(\boldsymbol{Z}) = \arg\min_{\boldsymbol{X}} \left(\|\boldsymbol{X}\|_{\text{nn}} + \frac{1}{2\alpha}\|\boldsymbol{X} - \boldsymbol{Z}\|_F^2 \right).$$

One way to approach this is by invoking the Moreau decomposition: for a convex function $f(\boldsymbol{x})$ with Fenchel conjugate f^*, we have the identity

$$\boldsymbol{x} = \text{prox}_{\alpha f}(\boldsymbol{x}) + \alpha\, \text{prox}_{\alpha^{-1} f^*}(\boldsymbol{x}/\alpha).$$

You have already computed the prox operator for f^* in this case ...

Exercise: Suppose we observe entries $(m,n) \in \mathcal{I}$ of a matrix $\boldsymbol{X}_0$ that we expect is low rank. Call these observations $Y_{m,n}$ (again for $(m,n) \in \mathcal{I}$). We will attempt to recover the matrix by solving the "matrix LASSO"

$$\underset{\boldsymbol{X}}{\text{minimize}} \quad \sum_{(m,n)\in\mathcal{I}} (Y_{m,n} - X_{m,n})^2 + \lambda \|\boldsymbol{X}\|_{\text{nn}}.$$

Write down a proximal-gradient algorithm for solving this problem.

Bonus Exercise: Show that the subdifferential of the nuclear norm for $M \times N$ matrices at a point $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$ is given by

$$\partial \|\boldsymbol{X}\|_{\mathrm{nn}} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}} + \boldsymbol{W},$$

for all $M \times N$ matrices $\boldsymbol{W}$ that obey,

$$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{W} = \boldsymbol{0}, \quad \boldsymbol{W}\boldsymbol{V} = \boldsymbol{0}, \quad \|\boldsymbol{W}\| \leq 1.$$

Bonus Exercise: Let $\boldsymbol{X}_0$ be an arbitrary $N \times N$ matrix. Find a closed form expression for projecting $\boldsymbol{X}_0$ onto the semidefinite cone. That is, solve

$$\underset{\boldsymbol{Y} \succeq \boldsymbol{0}}{\text{minimize}} \ \|\boldsymbol{X}_0 - \boldsymbol{Y}\|_F.$$

References

[lM75] l. Mirsky. A trace inequality of John von Neumann. *Monatshefte für Mathematik*, 79:303–306, 1975.

IV.A Further Topics: Convex Relaxation

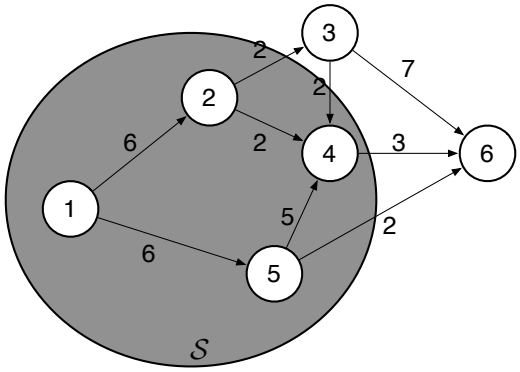
Convex relaxation

The art and science of *convex relaxation* revolves around taking a non-convex problem that you want to solve, and replacing it with a convex problem which you can actually solve – the solution to the convex program gives information about (usually a lower bound) the solution to the original program. Usually this is done by either "convexifying" the constraints or convexifying the functional – we will see examples of both below.

Minimum-Cut

Suppose that we have a directed graph with vertices indexed by $1, \ldots, N$. By convention we will denote vertex 1 as the "source" and vertex N as the "sink". Between each pair of vertices (i, j) there is a capacity $C_{i,j} \geq 0$ — if there is no edge from i to j, we take $C_{i,j} = 0$. A **cut** partitions the vertices into two sets: a $\mathcal{S}$ which contains the source, and a set $\mathcal{S}^c$ which contains the sink. The capacity of the cut is the sum of the capacities of all the edges that originate in $\mathcal{S}$ and terminate in $\mathcal{S}^C$.

In example below, we have $N = 6$ and $\mathcal{S} = \{1, 2, 4, 5\}$:



The capacity of this cut is $2 + 3 + 2 = 7$.

In general, the capacity associated with a cut $\mathcal{S}$ is

$$\sum_{i \in \mathcal{S}, j \notin \mathcal{S}} C_{i,j}.$$

If we take the vector $\boldsymbol{\nu} \in \mathbb{R}^N$ as

$$\nu_i = \begin{cases} 1, & i \in \mathcal{S}, \\ 0, & i \notin \mathcal{S} \end{cases}$$

then we can write the problem of finding the *minimum* cut as

$$\begin{aligned} \underset{\boldsymbol{\nu}}{\text{minimize}} \quad & \sum_{i=1}^{N} \sum_{j=1}^{N} C_{i,j} \max(\nu_i - \nu_j, 0) \\ \text{subject to} \quad & \nu_i \in \{0, 1\}, \ i = 1, \ldots, N, \\ & \nu_1 = 1, \quad \nu_N = 0. \end{aligned}$$

To make the objective function simpler, we introduce $\lambda_{i,j}$, and the minimum cut program can be rewritten as

$$\begin{aligned} \text{(MINCUT)} \quad \underset{\boldsymbol{\Lambda}, \boldsymbol{\nu}}{\text{minimize}} \quad & \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_{i,j} C_{i,j} \\ \text{subject to} \quad & \lambda_{i,j} = \max(\nu_i - \nu_j, 0), \ i, j = 1, \ldots, N, \\ & \nu_i \in \{0, 1\}, \ i = 1, \ldots, N, \\ & \nu_1 = 1, \quad \nu_N = 0. \end{aligned}$$

We will not do so here, but one reason why the minimum-cut problem is of interest is that its dual is the *maximum-flow* problem (i.e., given a directed graph and capacity constraints, what is the largest flow possible from the source to the sink).

As it is stated, there are two things making this program nonconvex – we have non-affine equality constraints relating $\lambda_{i,j}$ to ν_i and ν_j, and we have binary constraints on ν_i. If we simply drop the integer constraint, and relax

$$\lambda_{i,j} = \max(\nu_i - \nu_i, 0) \quad \text{to} \quad \lambda_{i,j} \geq \nu_i - \nu_j \ \text{ and } \ \lambda_{i,j} \geq 0,$$

we are left with the linear program

$$\begin{aligned} \text{(LP-R)} \qquad & \underset{\boldsymbol{\Lambda},\boldsymbol{\nu}}{\text{minimize}} \ \ \langle \boldsymbol{\Lambda}, \boldsymbol{C} \rangle \\ & \text{subject to} \ \ \lambda_{i,j} \geq \nu_i - \nu_j, \ \ \lambda_{i,j} \geq 0, \ i,j = 1,\ldots,N, \\ & \qquad\qquad \nu_1 = 1, \ \nu_N = 0. \end{aligned}$$

Note that the domain we are optimizing over in the LP relaxation is larger than the domain in the original formulation – this means that every valid cut (feasible $\boldsymbol{\Lambda}, \boldsymbol{\nu}$ for the original program) is feasible in the LP relaxation. So at the very least we know that

$$\text{LP-R}^\star \ \leq \ \text{MINCUT}^\star.$$

But the semi-amazing thing is that the solutions to the two programs turn out to agree.

We show this by establishing that for every solution of the relaxation, there is at least one cut with value less than or equal to LP-R*. We do this by generating a *random cut* (with the associated probabilities carefully chosen) and show that in expectation, it is less than LP-R*.

Let Z be a uniform random variable on $[0,1]$. Let $\boldsymbol{\Lambda}^\star, \boldsymbol{\nu}^\star$ be solutions to (LP-R). Create a cut $\mathcal{S}$ with the rule:

$$\text{if } \nu_n^\star > Z, \ \text{then take } n \in \mathcal{S}.$$

The probability that a particular edge $i \to j$ is in this cut is

$$\begin{aligned} \mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) &= \mathrm{P}\left(\nu_j^\star \leq Z \leq \nu_i^\star\right) \\ &\leq \max(\nu_i^\star - \nu_j^\star, 0) \\ &\leq \lambda_{i,j}^\star, \end{aligned}$$

where the last inequality follows simply from the constraints in (LP-R). This cut is random, so its capacity is a random variable, and its expectation is

$$\begin{aligned} \mathrm{E}[\text{capacity}(\mathcal{S})] &= \sum_{i,j} C_{i,j} \, \mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) \\ &\leq \sum_{i,j} C_{i,j} \lambda_{i,j}^\star \\ &= \text{LP-R}^\star. \end{aligned}$$

Thus there must be a cut whose capacity is at most LP-R*. This establishes that

$$\text{MINCUT}^\star \leq \text{LP-R}^\star.$$

Of course, combining this with the result above means than

$$\text{MINCUT}^\star = \text{LP-R}^\star.$$

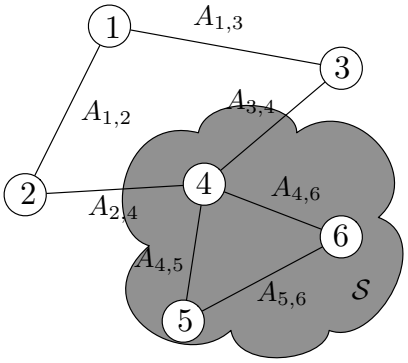
This is an example of a wonderful situation where convex relaxation costs us nothing, but makes solving the program computationally tractable.

Maximum-cut

A good resource for this section and the next is Ben-Tal and Nemirovski [BTN01].

This problem has a very similar setup as the minimum-cut problem, but it is different in subtle ways. We are given a graph; this time the edges are undirected, and have positive weights $A_{i,j}$ associated with them. Since the graph is undirected, $A_{i,j} = A_{j,i}$ and so $\boldsymbol{A}$ is symmetric. We will also assume that $A_{i,i} = 0$ for all i.

As before, a cut partitions the vertices into two sets, $\mathcal{S}$ and $\mathcal{S}^c$ – these sets can be arbitrary; there is no notion of source and sink here. For example, the cut in this example:



has value $\text{cut}(\mathcal{S}) = A_{2,4} + A_{3,4}$. The problem is to find the cut that **maximizes** the weights of the edges going between the two partitions.

We can specify a cut of the graph with a binary valued vector $\boldsymbol{x}$ of length N, where each $x_n \in \{-1, 1\}$. We set $x_n = 1$ if vertex n is in

$\mathcal{S}$ and $x_n = -1$ if vertex n is in $\mathcal{S}^c$. The value of the cut is

$$\text{cut}(\mathcal{S}) = \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}(1 - x_i x_j).$$

Note that if $x_i \neq x_j$, then $(1 - x_i x_j) = 2$, while if $x_i = x_j$, then $(1 - x_i x_j) = 0$. The factor of $1/4$ in front comes from the fact that $(1 - x_i x_j) = 2$ for edges in the cut, and that we are counting every edge twice (from i to j and again from j to i). Notice that we can write this value as a quadratic function of $\boldsymbol{x}$:

$$\text{cut}(\mathcal{S}) = \frac{1}{4}\left(\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}\right)$$

The maximum-cut problem is find the cut with the largest value:

$$\begin{aligned} \text{(MAXCUT)} \quad & \underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{maximize}} \quad \frac{1}{4}\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \frac{1}{4}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \\ & \text{subject to} \quad x_i \in \{-1, 1\}. \end{aligned}$$

Right now, this looks pretty gnarly, as $\boldsymbol{A}$ has no guarantee of being PSD, and we have integer constraints on $\boldsymbol{x}$. We can address the first concern by re-writing this as a search for a matrix $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}$. As now[1] $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \langle \boldsymbol{X}, \boldsymbol{A}\rangle$, we have

$$\begin{aligned} \underset{\boldsymbol{X}\in\mathbb{R}^{N\times N}}{\text{maximize}} \quad & \frac{1}{4}\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \frac{1}{4}\langle \boldsymbol{X}, \boldsymbol{A}\rangle \\ \text{subject to} \quad & \boldsymbol{X} \succeq \mathbf{0} \\ & X_{i,i} = 1, \quad i = 1, \ldots, N \\ & \text{rank}(\boldsymbol{X}) = 1. \end{aligned}$$

[1] Recall that the standard inner product between two $M \times N$ matrices is $\langle \boldsymbol{X}, \boldsymbol{A}\rangle = \sum_{m,n} A_{m,n}X_{m,n} = \text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X})$. This is exactly the same as "flattening out" the matrices as vectors and using the standard Euclidean inner product.

You should be able to convince yourself that $\boldsymbol{X}$ is feasible above if and only if it can be written as $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}$, where the entries of $\boldsymbol{x}$ are ± 1.

The recast program looks like a semidefinite program (SDP), except for the rank constraint. The relaxation, then, is to simply drop it and solve

$$\begin{aligned} \text{(MAXCUT-R)} \quad \underset{\boldsymbol{X} \in \mathbb{R}^{N \times N}}{\text{maximize}} \quad & \frac{1}{4}\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \frac{1}{4}\langle \boldsymbol{X}, \boldsymbol{A} \rangle \\ \text{subject to} \quad & \boldsymbol{X} \succeq \mathbf{0} \\ & X_{i,i} = 1, \ i = 1, \ldots, N. \end{aligned}$$

As we are optimizing over a larger set, the optimal value of MAXCUT-R will in general be larger than MAXCUT:

$$\text{MAXCUT-R}^{\star} \ \geq \ \text{MAXCUT}^{\star}.$$

But there is a classic result [GW95] that shows it will not be too much larger:

$$\text{MAXCUT}^{\star} \ \geq \ (0.87856) \cdot \text{MAXCUT-R}^{\star}.$$

The argument again relies on looking at the expected value of a random cut. Let $\boldsymbol{X}^{\star}$ be a solution to MAXCUT-R. Since $\boldsymbol{X}^{*}$ is PSD, it can be factored as

$$\boldsymbol{X}^{\star} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}.$$

With $\boldsymbol{v}_j$ as the j^{th} column of $\boldsymbol{V}$, this means $X^{\star}_{i,j} = \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$. Since along the diagonal we have $X^{\star}_{i,i} = 1$, this means that $\|\boldsymbol{v}_i\|_2 = 1$ as well. We can associate one column $\boldsymbol{v}_i$ with each vertex in the original

problem. To create the cut, we draw a vector $\boldsymbol{z}$ from the unit-sphere (so $\|\boldsymbol{z}\|_2 = 1$) uniformly at random,[2] and set

$$\mathcal{S} = \{i \ : \ \langle \boldsymbol{v}_i, \boldsymbol{z} \rangle \geq 0\}.$$

It should be clear that the probability that any fixed vertex is in $\mathcal{S}$ is $1/2$. But what is the probability that vertex i and vertex j are on different sides? The probability of this is simply the ratio of the angle between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ to π:

$$\mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) + \mathrm{P}\left(i \notin \mathcal{S}, j \in \mathcal{S}\right) = \frac{\arccos\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle}{\pi} = \frac{\arccos X^\star_{i,j}}{\pi}.$$

Thus the expectation of the cut value is

$$\begin{aligned} \mathrm{E}[\mathrm{cut}(\mathcal{S})] &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\left(\mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) + \mathrm{P}\left(i \notin \mathcal{S}, j \in \mathcal{S}\right)\right) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\frac{\arccos X^\star_{i,j}}{2\pi}. \end{aligned}$$

There must be at least one cut that has a value greater than or equal to the mean, so we know that

$$\mathrm{MAXCUT} \ \geq \ \mathrm{E}[\mathrm{cut}(\mathcal{S})].$$

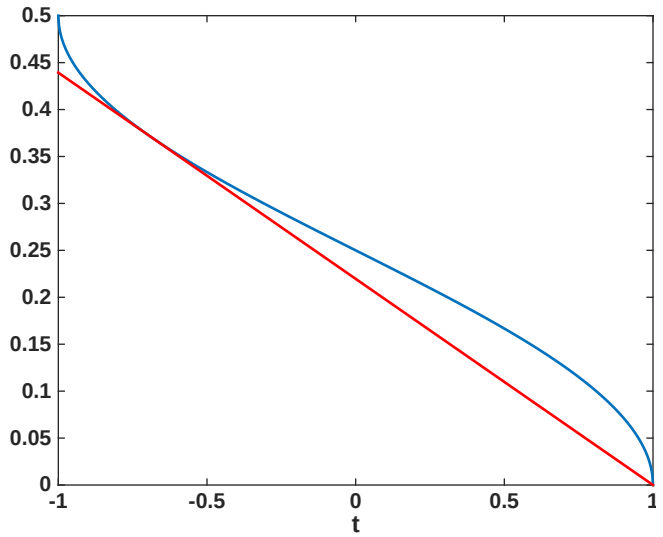
Let's compare the terms in this sum to those in the objection function for MAXCUT-R. We know that the entries in $\boldsymbol{X}^\star$ have at most unit magnitude[3] $-1 \leq X^\star_{i,j} \leq 1$, and it is a fact that:

$$\frac{\arccos t}{2\pi} \ \geq \ (0.87856)\frac{1}{4}(1-t), \quad \text{for} \ \ t \in [-1, 1].$$

[2] In practice, you could do this by drawing each entry Normal$(0, 1)$ independently, then normalizing.

[3] This follows from $X^\star_{i,j} = \langle \boldsymbol{v}_i, \boldsymbol{v}_k \rangle$, $\|\boldsymbol{v}_i\|_2 = 1$, and Cauchy-Swartz.

Here is a little "proof by plot" of this fact:



$$\text{blue} = \frac{\arccos t}{2\pi}, \text{ red} = (0.878856)\tfrac{1}{4}(1-t).$$

Thus

$$\begin{aligned}
\text{MAXCUT}^\star &\geq \text{E}[\text{cut}(\mathcal{S})] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\frac{\arccos X^\star_{i,j}}{2\pi} \\
&\geq (0.87856)\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{1}{4}A_{i,j}(1-X^\star_{i,j}) \\
&= (0.87856)\cdot \text{MAXCUT-R}^\star
\end{aligned}$$

Quadratic equality constraints

The integer constraint $x_i \in \{-1, 1\}$ in the example above might also be interpreted as a quadratic *equality* constraint:

$$x_i \in \{-1, 1\} \quad \Leftrightarrow \quad x_i^2 = 1.$$

As we are well aware, quadratic (or any other nonlinear) equality constraints make the feasibility region nonconvex.

We consider general nonconvex quadratic programs of the form

$$\begin{array}{ll} \underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} & \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A}_0 \boldsymbol{x} + 2\langle \boldsymbol{x}, \boldsymbol{b}_0 \rangle + c_0 \\ \text{subject to} & \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A}_m \boldsymbol{x} + 2\langle \boldsymbol{x}, \boldsymbol{b}_m \rangle + c_m = 0, \quad m = 1, \ldots, M, \end{array}$$

where the $\boldsymbol{A}_m$ are symmetric, but not necessarily $\succeq \boldsymbol{0}$. We will show how to recast these problems as optimization over the SDP cone with an additional (nonconvex) rank constraint. Then we will have a natural convex relaxation by dropping the rank constraint. This general methodology works for equality or (possibly nonconvex) inequality constraints, but for the sake of simplicity, we will just look at equality constraints.

We can turn a quadratic form into a trace inner product with a rank 1 matrix as follows. It is clear that

$$\begin{aligned} \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x} + 2\boldsymbol{b}^{\mathrm{T}} \boldsymbol{x} + c &= \begin{bmatrix} \boldsymbol{x}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^{\mathrm{T}} & c \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \\ &= \operatorname{trace}\left(\begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^{\mathrm{T}} & c \end{bmatrix} \boldsymbol{X}_x \right), \quad \boldsymbol{X}_x = \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^{\mathrm{T}} & 1 \end{bmatrix}. \end{aligned}$$

This means we can write the nonconvex quadratic program as

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \quad & \operatorname{trace}\left(\begin{bmatrix} \boldsymbol{A}_0 & \boldsymbol{b}_0 \\ \boldsymbol{b}_0^{\mathrm{T}} & c_0 \end{bmatrix} \boldsymbol{X}_x\right) \\ \text{subject to} \quad & \operatorname{trace}\left(\begin{bmatrix} \boldsymbol{A}_m & \boldsymbol{b}_m \\ \boldsymbol{b}_m^{\mathrm{T}} & c_m \end{bmatrix} \boldsymbol{X}_x\right) = 0, \quad m = 1, \ldots, M. \end{aligned}$$

With

$$\boldsymbol{F}_m = \begin{bmatrix} \boldsymbol{A}_m & \boldsymbol{b}_m \\ \boldsymbol{b}_m^{\mathrm{T}} & c_m \end{bmatrix},$$

we see that this program is equivalent to

$$\begin{aligned} \underset{\boldsymbol{X}\in\mathbb{R}^{N\times N}}{\text{minimize}} \quad & \langle \boldsymbol{X}, \boldsymbol{F}_0 \rangle \\ \text{subject to} \quad & \langle \boldsymbol{X}, \boldsymbol{F}_m \rangle = 0, \quad m = 1, \ldots, M \\ & \boldsymbol{X} \succeq \boldsymbol{0} \\ & \operatorname{rank}(\boldsymbol{X}) = 1. \end{aligned}$$

Again, we can get a convex relaxation simply by dropping the rank constraint. How well this works depends on the particulars of the problem. There are certain situations where it is exact; one of these is when there is a single non-convex inequality constraint.There are other situations where it is not exact but is provably good – one example is maximum-cut. There are other situations where it is arbitrarily bad.

Example: Phase retrieval

In coherent imaging applications, a not uncommon problem is to reconstruct an unknown vector $\boldsymbol{x}$ from measurements of the *magnitude* of a series of linear functionals. We observe

$$y_m = |\langle \boldsymbol{x}, \boldsymbol{a}_m \rangle|^2 \ \ (+ \text{ noise}), \quad m = 1, \ldots, M.$$

For instance, if $\boldsymbol{a}_m$ are Fourier vectors, we are observing samples of the magnitude of the Fourier transform of $\boldsymbol{x}$. If we also measured the phase, then recovering $\boldsymbol{x}$ is a standard linear inverse problem (and if we have a complete set of samples in the Fourier domain, you can just take an inverse Fourier transform). But since we do not get to see the phase, we have to estimate it along with the underlying $\boldsymbol{x}$ – this problem is often referred to as **phase retrieval**.

We can rewrite the measurements as

$$\begin{aligned} y_m = \langle \boldsymbol{a}_m, \boldsymbol{x} \rangle \langle \boldsymbol{x}, \boldsymbol{a}_m \rangle = \boldsymbol{x}^{\mathrm{H}} \boldsymbol{a}_m \boldsymbol{a}_m^{\mathrm{H}} \boldsymbol{x} &= \operatorname{trace}(\boldsymbol{a}_m \boldsymbol{a}_m^{\mathrm{H}} \boldsymbol{x} \boldsymbol{x}^{\mathrm{H}}) \\ &= \langle \boldsymbol{X}, \boldsymbol{A}_m \rangle_F, \end{aligned}$$

where $\boldsymbol{A}_m = \boldsymbol{a}_m \boldsymbol{a}_m^{\mathrm{H}}$ and $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^{\mathrm{H}}$. So solving the phase retrieval problem is the same as finding an $N \times N$ matrix with the following properties:

$$\langle \boldsymbol{X}, \boldsymbol{A}_m \rangle_F = y_m, \quad m = 1, \ldots, M, \quad \boldsymbol{X} \succeq \boldsymbol{0}, \quad \operatorname{rank}(\boldsymbol{X}) = 1.$$

The first condition is just that $\boldsymbol{X}$ obeys a certain set of linear equality constraints; the second is that $\boldsymbol{X}$ is in the SDP cone; the third is a nonconvex constraint.

One convex relaxation for this problem simply drops the rank constraint and finds a feasible point that obeys the first two conditions. Under certain conditions on the $\boldsymbol{a}_m$, there will only be one point in this intersection once M is mildly larger than N.

Example: Maximum Likelihood MIMO decoding

In MIMO (multiple input multiple output) digital communications, we have N transmit antennas and M receive antennas. Each transmit antenna sends a bit (± 1), which are collected together into an N-vector $\boldsymbol{x}$.

The receive antennas observe

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{v},$$

where $\boldsymbol{H}$ is a known $M \times N$ *fading* matrix (or channel matrix), and $\boldsymbol{v} \sim \text{Normal}(0, \sigma^2 \mathbf{I})$.

1. The *maximum likelihood decoder* finds the binary valued vector $\boldsymbol{x}$ that makes the observations $\boldsymbol{y}$ most likely:

   $$\widehat{\boldsymbol{x}}_{\text{ML}} = \underset{\boldsymbol{x} \in \{-1,1\}^N}{\arg\max}\ p(\boldsymbol{y}|\boldsymbol{x}).$$

   Show how this can be written as a least-squares problem with integer constraints.

2. Show how the optimization program above can be relaxed into a semidefinite program.

Sparse solutions to linear systems of equations

Many, many problems in scientific computing and data science amount to solving a linear system of equations. That is, given an $M \times N$ matrix $\boldsymbol{A}$ and a response vector $\boldsymbol{y} \in \mathbb{R}^M$, we want to find $\boldsymbol{x} \in \mathbb{R}^N$ such that

$$\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{y}.$$

The applications here are too numerous to list, but we will settle for kernel regression in machine learning and statistics, medical imaging, seismic exploration, radar, and channel estimation in wireless communications.

As we have seen continuously throughout this course, the starting point for this problem is the least-squares optimization program

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

In some sense, this will return the $\hat{\boldsymbol{x}}$ that come closest to explaining $\boldsymbol{y}$.

There can, however, be many solutions to the least-squares program above. This is certainly true when $\boldsymbol{A}$ is under determined, that is it has fewer rows than it has columns. We now need to choose between these solutions.

One criteria we can use is to choose the $\boldsymbol{x}$ that has the smallest number of non-zero terms, i.e. is the "sparsest". This has appeal across many of the applications listed above. For example, in regression we might want to choose the smallest number of features that explain the response. In imaging, we can often express a target as a sparse superposition of pre-determined basis functions (indeed, this is the main idea behind every image or video compression algorithm).

We might modify the least-square problem by introducing a sparsity constraint

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \text{nnz}(\boldsymbol{x}) \leq S,$$

where $\text{nnz}(\cdot)$ returns the number of non-zero terms, or by introducing a penalty (or regularizer) which is basically putting the above in Lagrange form,

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda \cdot \text{nnz}(\boldsymbol{x}),$$

It should be clear that $\text{nnz}(\cdot)$ is not a convex function, and indeed both of the programs above have been shown to be NP-hard [Nat95].

There is, however, a natural convex relaxation to the above. In place of $\text{nnz}(\cdot)$, we use the ℓ_1 norm. We replace the above with

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1.$$

The above is called the 'LASSO' [Tib96] in statistics and 'basis pursuit' [CDS99] in signal processing. Roughly speaking, this relaxation works since signals that are sparse (have a small number of nonzero terms) have small ℓ_1 norm relative to their Euclidean norm.

There is a rich theory for using ℓ_1 norm minimization for solving systems of equations. One typical result is that for "generic" $M \times N$ matrices $\boldsymbol{A}$ and an observation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^\star$, then the solution to

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y},$$

will be exactly $\boldsymbol{x}^\star$ when M is on the order of non-zero terms, even when $M \ll N$ [CRT06a, CRT06b, Don06]. Hence we can invert under determined systems even when the solutions are indeed sparse.

Throughout this course, we have seen many algorithms for solving these types of problems.

Low rank matrix recovery

Consider the following fundamental, easily stated problem. Suppose there is a $M \times N$ matrix

$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} & X_{1,5} \\ X_{2,1} & X_{2,2} & X_{2,3} & X_{2,4} & X_{2,5} \\ X_{3,1} & X_{3,2} & X_{3,3} & X_{3,4} & X_{3,5} \\ X_{4,1} & X_{4,2} & X_{4,3} & X_{4,4} & X_{4,5} \\ X_{5,1} & X_{5,2} & X_{5,3} & X_{5,4} & X_{5,5} \end{bmatrix}$$

whose entries we only partially observe,

$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & - & X_{1,3} & - & X_{1,5} \\ - & X_{2,2} & - & X_{2,4} & - \\ - & X_{3,2} & X_{3,3} & - & - \\ X_{4,1} & - & - & X_{4,4} & X_{4,5} \\ - & - & - & X_{5,4} & X_{5,5} \end{bmatrix}.$$

Is it possible to "fill in the blanks"?

Of course, in general the answer is "no". But what if the matrix is structured in that it has rank $R \ll \min(M, N)$? Then the answer (under some conditions on the underlying matrix) is "yes". Revealing just a few entries per row/column makes the problem identifiable: there is only one low rank matrix that can have exactly those entries. This matrix can be recovered (the entries "filed in") by solving

$$\underset{\boldsymbol{X}}{\text{minimize}} \;\; \text{rank}(\boldsymbol{X}) \quad \text{subject to} \quad X_{m,n} = Y_{m,n}, \quad (m,n) \in \mathcal{I},$$

where $\mathcal{I}$ is the set of observed indices, and $Y_{m,n}$ are their observed values.

This is again an NP-hard problem. But also again there is a natural convex relaxation (first made popular in [Faz02]), we replace rank with the *nuclear norm*:

$$\underset{\boldsymbol{X}}{\text{minimize}} \ \|\boldsymbol{X}\|_{\text{nn}} \quad \text{subject to} \quad X_{m,n} = Y_{m,n}, \quad (m,n) \in \mathcal{I}.$$

The nuclear norm $\|\boldsymbol{X}\|_{\text{nn}}$ is the sum of the singular values of $\boldsymbol{X}$. This is actually the dual norm of the standard matrix operator norm (maximum singular value). Note the parallels to the vector case: we relax the number of non-zero singular values (the rank) to the sum, just as we relaxed the number of non-zero entries in a vector to the sum of the absolute values. Also, just as the ℓ_1 norm is dual to the ℓ_∞ norm, the nuclear norm (sum of singular values) is dual to the operator norm (maximum of singular values).

There is also a rich theory for recovering low-rank matrices from partial observations; seminal works include [RFP10, CR09], and a review can be found in [DR16].

References

[BTN01] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Covex Optimization*. SIAM, 2001.

[CDS99] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20(1):33–61, 1999.

[CR09] E. Candès and B. Recht. Exact matrix completion via convex optimization. *Found. of Comput. Math.*, 9(6):717–772, 2009.

[CRT06a] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory*, 52(2):489–509, February 2006.

[CRT06b] E. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Comm. on Pure and Applied Math.*, 59(8):1207–1223, 2006.

[Don06] D. L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4):1289–1306, April 2006.

[DR16] M. A. Davenport and J. Romberg. An overview of low-rank matrix recovery from incomplete observations. *IEEE J. Selected Topics in Sig. Proc.*, 10(4):608–622, 2016.

[Faz02] M. Fazel. *Matrix rank minimization with applications*. PhD thesis, Stanford University, March 2002.

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comp. Mach.*, 42(6):1115–1145, November 1995.

[Nat95] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–34, 1995.

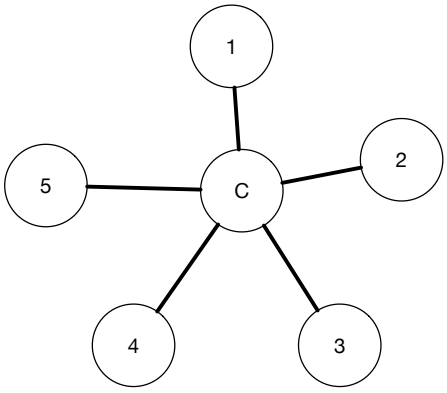
[RFP10] B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the Lasso. *J. Royal Stat. Soc. Ser. B*, 58(1):267–288, 1996.

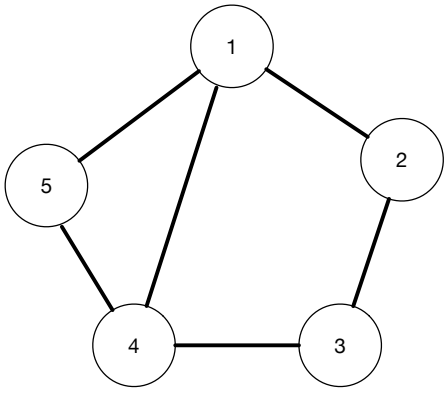
IV.B Further Topics: Distributed Optimization

In this section we will discuss different methods for taking a large optimization program and breaking into smaller ones. These smaller problems are distributed to different "agents", each of which has a computational core that can "work" on the smaller problem. As the smaller problems are in general linked together, the agents have to coordinate in some manner so that they work towards the solution of the original problem.

We will look at two types of coordination models. In the *centralized* setting, there is a central node that the agents coordinate through. In the *decentralized* setting, the nodes communication with one another on a predefined network. These two communication models can be captured using graphical models as shown below.



centralized coordinator



decentralized network

The optimization problems we look at will also be structured (in one of two different ways). The general form of the problems we consider is

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + g(\boldsymbol{x}),$$

where f and g are convex functions. In addition we will assume that either f or g have some type of separable structure. This assumed structure, however, is prevalent in supervised learning.

A typical problem in supervised learning is to find a linear combination of a features of observed data points that can come close to matching the responses in a vector $\boldsymbol{b}$. The N features for each of M data points are arranged in an $M \times N$ matrix $\boldsymbol{A}$. We are interested in solving problem of the form

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \underbrace{\text{Loss}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})}_{f(\boldsymbol{x})} \ + \ \underbrace{\text{Regularizer}(\boldsymbol{x})}_{g(\boldsymbol{x})}$$

Notice that

$$\text{Loss}(\cdot) : \mathbb{R}^M \to \mathbb{R}, \quad \text{and} \quad \text{Regularizer}(\cdot) : \mathbb{R}^N \to \mathbb{R}.$$

We will assume that one or both of these functions are separable, at least at the block level. This means we can write

$$f(\boldsymbol{x}) = \text{Loss}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \sum_{i=1}^{B} \ell_i(\boldsymbol{A}^{(i)}\boldsymbol{x} - \boldsymbol{b}^{(i)}) = \sum_{i=1}^{B} f_i(\boldsymbol{x}),$$

or

$$g(\boldsymbol{x}) = \text{Regularizer}(\boldsymbol{x}) = \sum_{i=1}^{B} r_i(\boldsymbol{x}^{(i)}),$$

where the $\boldsymbol{x}^{(i)} \in \mathbb{R}^{N_i}$ partition the vector $\boldsymbol{x}$. These two types of separability will allow us to divide up the optimization in two different ways.

We start by looking at some examples of where we can perform this decomposition.

Example: Inverse Problems and Regression

Two popular methods for solving linear inverse problems and/or calculating regressors are solving

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\tau\|\boldsymbol{x}\|_2^2,$$

(*Tikhonov regularization* or *ridge regression*), and

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\tau\|\boldsymbol{x}\|_1,$$

(*the LASSO*).

These both clearly fit both the separability criteria, as

$$\begin{aligned}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2 &= \sum_{m=1}^{M}(\boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{x}-b[m])^2,\\ \|\boldsymbol{x}\|_2^2 &= \sum_{n=1}^{N}(x[n])^2\\ \|\boldsymbol{x}\|_1 &= \sum_{n=1}^{N}|x[n]|.\end{aligned}$$

where $\boldsymbol{a}_m^{\mathrm{T}}$ is the m^{th} row of $\boldsymbol{A}$. We have a lot of flexibility in this situation, as we can partition the rows of $\boldsymbol{A}$ or the entries in $\boldsymbol{x}$ an way we want — if we have access to B agents, it makes sense divide them into B sets of equal size. We could take, for example

$$f(\boldsymbol{x})=\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2=\sum_{i=1}^{B}\|\boldsymbol{A}^{(i)}\boldsymbol{x}-\boldsymbol{b}^{(i)}\|_2^2=\sum_{i=1}^{B}f_i(\boldsymbol{x})$$

Example: Support Vector Machines

Previously, we saw how if we are given a set of M training examples $(\boldsymbol{x}_m, y_m)$, where $\boldsymbol{x}_m \in \mathbb{R}^N$ and $y_m \in \{-1, 1\}$, we can find a maximum margin linear classifier by solving

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|_2^2 \quad \text{subject to} \quad y_m(b - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1 \leq 0, \ \ m = 1, \ldots, M.$$

With the classifier trained (optimal solution $\boldsymbol{w}^\star, b^\star$ computed), we can assign a label y' to a new point $\boldsymbol{x}'$ using

$$y' = \text{sign}(\langle \boldsymbol{x}', \boldsymbol{w}^\star \rangle + b^\star).$$

Instead of enforcing the constraints above strictly, we can allow some errors by penalizing mis-classifications on the training data appropriately. One reasonable way to do this is make the loss zero if $y_m(b - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1 \leq 0$, and then have it increase linearly as this quantity exceeds zero. That is, we solve

$$\min_{\boldsymbol{w},b} \ \sum_{m=1}^{M} \ell(y_m(b - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1) + \frac{1}{2}\|\boldsymbol{w}\|_2^2,$$

where $\ell(\cdot)$ is

$$\ell(u) = (u)_+ = \begin{cases} 0, & u \leq 0, \\ u, & u > 0. \end{cases}$$

This is penalty is often called the **hinge loss**. Note that the argument for $\ell(\cdot)$ is an affine function of the optimization variables:

$$y_m(b - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1 = \begin{bmatrix} -y_m \boldsymbol{x}_m^{\mathrm{T}} & y_m \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix} + 1.$$

Both the loss function and regularizer in this formulation of the SVM are clearly separable.

ADMM: Splitting across examples

In this section[1], we will see how we can use ADMM to solve problems of the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \quad \sum_{i=1}^{B} f_i(\boldsymbol{x}) + g(\boldsymbol{x}) \tag{1}$$

using B agents with a centralized coordinator. In the context of supervised learning, this framework is useful when we have "many measurements of a small vector" or "large volumes of low-dimensional data". We can use it when

$$f(\boldsymbol{x}) = \text{Loss}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \sum_{i=1}^{B} \ell_i(\boldsymbol{A}^{(i)}\boldsymbol{x} - \boldsymbol{b}^{(i)}) = \sum_{i=1}^{B} f_i(\boldsymbol{x}),$$

where we partition the rows of $\boldsymbol{A}$ and entries of $\boldsymbol{b}$ as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}^{(1)} \\ \boldsymbol{A}^{(2)} \\ \vdots \\ \boldsymbol{A}^{(B)} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}^{(1)} \\ \boldsymbol{b}^{(2)} \\ \vdots \\ \boldsymbol{b}^{(B)} \end{bmatrix}.$$

We start by splitting the optimization in (1) variables in f and g, arriving at the equivalent (now constrained) program

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \quad \sum_{i=1}^{B} f_i(\boldsymbol{x}) + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0}.$$

We do not yet have this in a form we can distribute, as all of the f_i are tied together by working on the same argument $\boldsymbol{x}$. The trick is

[1] A good resource for the material in the next two sections of the notes is [BPC+10].

to introduce B different vectors $\boldsymbol{x}^{(i)} \in \mathbb{R}^N$, one for each agent, and then use the constraints to make them all agree. This results in the equivalent problem

$$\begin{aligned} &\underset{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(B)},\boldsymbol{z}}{\text{minimize}} \quad \sum_{i=1}^{B} f_i(\boldsymbol{x}^{(i)}) + g(\boldsymbol{z}) \\ &\text{subject to} \quad \boldsymbol{x}^{(i)} - \boldsymbol{z} = \boldsymbol{0}, \quad i = 1, \ldots, B. \end{aligned}$$

The augmented Lagrangian for this last problem is now separable and can be expressed as

$$\mathcal{L}_\rho(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(B)}, \boldsymbol{z}, \boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(B)}) = \sum_{i=1}^{B} \mathcal{L}_i(\boldsymbol{x}^{(i)}, \boldsymbol{z}, \boldsymbol{\mu}^{(i)}),$$

where

$$\mathcal{L}_i(\boldsymbol{x}^{(i)}\boldsymbol{z}, \boldsymbol{\mu}^{(i)}) = f_i(\boldsymbol{x}^{(i)}) + \frac{g(\boldsymbol{z})}{B} + \frac{\rho}{2}\|\boldsymbol{x}^{(i)} - \boldsymbol{z} + \boldsymbol{\mu}^{(i)}\|_2^2$$

and the $\boldsymbol{\mu}^{(i)}$ are the (rescaled) Lagrange multipliers for the constraint $\boldsymbol{x}^{(i)} - \boldsymbol{z} = \boldsymbol{0}$.

As the Lagrangian is separable over the B blocks, each of the primal updates for the $\boldsymbol{x}_i$ can be performed independently. The ADMM iteration amounts to

$$\begin{aligned} \boldsymbol{x}_{k+1}^{(i)} &= \arg\min_{\boldsymbol{x}^{(i)}} \left(f_i(\boldsymbol{x}^{(i)}) + \frac{\rho}{2}\|\boldsymbol{x}^{(i)} - \boldsymbol{z}_k + \boldsymbol{\mu}_k^{(i)}\|_2^2 \right), \quad i = 1, \ldots, B \\ \boldsymbol{z}_{k+1} &= \arg\min_{\boldsymbol{z}} \left(g(\boldsymbol{z}) + \frac{\rho}{2} \sum_{i=1}^{B} \|\boldsymbol{z} - \boldsymbol{x}_{k+1}^{(i)} - \boldsymbol{\mu}_k^{(i)}\|_2^2 \right) \\ \boldsymbol{\mu}_{k+1}^{(i)} &= \boldsymbol{\mu}_k^{(i)} + \boldsymbol{x}_{k+1}^{(i)} - \boldsymbol{z}_{k+1} \end{aligned}$$

The $\boldsymbol{z}$ update can be simplified by writing it in terms of the average of the $\boldsymbol{x}_{k+1}^{(i)}$ and the $\boldsymbol{\mu}_k^{(i)}$. To see how this works note that

$$\begin{aligned}\sum_{i=1}^{B} \|\boldsymbol{z}-\boldsymbol{v}_i\|_2^2 &= B\|\boldsymbol{z}\|_2^2 - 2\left\langle \boldsymbol{z}, \sum_{i=1}^{B}\boldsymbol{v}_i\right\rangle + \sum_{i=1}^{N}\|\boldsymbol{v}_i\|_2^2 \\ &= B\|\boldsymbol{z}\|_2^2 - 2B\left\langle \boldsymbol{z}, \bar{\boldsymbol{v}}\right\rangle + B\|\bar{\boldsymbol{v}}\|_2^2 + \left(-B\|\bar{\boldsymbol{v}}\|_2^2 + \sum_{i=1}^{N}\|\boldsymbol{v}_i\|_2^2\right) \\ &= B\|\boldsymbol{z}-\bar{\boldsymbol{v}}\|_2^2 + \left(-B\|\bar{\boldsymbol{v}}\|_2^2 + \sum_{i=1}^{N}\|\boldsymbol{v}_i\|_2^2\right).\end{aligned}$$

where $\bar{\boldsymbol{v}} = \frac{1}{B}\sum_{i=1}^{B}\boldsymbol{v}_i$. Thus

$$\begin{aligned}&\arg\min_{\boldsymbol{z}}\left(g(\boldsymbol{z}) + \frac{\rho}{2}\sum_{i=1}^{B}\|\boldsymbol{z}-\boldsymbol{x}_{k+1}^{(i)}-\boldsymbol{\mu}_k^{(i)}\|_2^2\right) \\ &= \arg\min_{\boldsymbol{z}}\left(g(\boldsymbol{z}) + \frac{B\rho}{2}\|\boldsymbol{z}-\bar{\boldsymbol{x}}_{k+1}-\bar{\boldsymbol{\mu}}_k\|_2^2\right)\end{aligned}$$

Distributed ADMM (splitting across data)

$$\boldsymbol{x}_{k+1}^{(i)} = \arg\min_{\boldsymbol{x}^{(i)}}\left(f_i(\boldsymbol{x}^{(i)}) + \frac{\rho}{2}\|\boldsymbol{x}^{(i)}-\boldsymbol{z}_k+\boldsymbol{\mu}_k^{(i)}\|_2^2\right), \quad i=1,\ldots,B$$

$$\boldsymbol{z}_{k+1} = \arg\min_{\boldsymbol{z}}\left(g(\boldsymbol{z}) + \frac{B\rho}{2}\|\boldsymbol{z}-\bar{\boldsymbol{x}}_{k+1}-\bar{\boldsymbol{\mu}}_k\|_2^2\right)$$

$$\boldsymbol{\mu}_{k+1}^{(i)} = \boldsymbol{\mu}_k^{(i)} + \boldsymbol{x}_{k+1}^{(i)} - \boldsymbol{z}_{k+1}, \quad i=1,\ldots,B,$$

where

$$\bar{\boldsymbol{x}}_{k+1} = \frac{1}{B}\sum_{i=1}^{B}\boldsymbol{x}_{k+1}^{(i)}, \quad \bar{\boldsymbol{\mu}}_k = \frac{1}{B}\sum_{i=1}^{B}\boldsymbol{\mu}_k^{(i)}.$$

The high-level architecture is that B separate agents solve independent optimization programs for the B $\boldsymbol{x}^{(i)}$ updates. These are collected by the coordinator and averaged, and then the coordinator solves a single optimization program to get the $\boldsymbol{z}$ update. The new $\boldsymbol{z}$ is then communicated back to each of the B units. The Lagrange multiplier update can then be easily computed by the agents before proceeding again to the $\boldsymbol{x}^{(i)}$ updates.

Example: The LASSO

(Homework ...)

Example: SVMs

For the SVM, we collect the weights and the offset into a single optimization vector

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix} \in \mathbb{R}^{N+1}$$

and set

$$\boldsymbol{A} = \begin{bmatrix} -y_1 \boldsymbol{x}_1^{\mathrm{T}} & y_1 \\ \vdots & \vdots \\ -y_M \boldsymbol{x}_M^{\mathrm{T}} & y_M . \end{bmatrix}$$

If we partition the data ($\boldsymbol{A}$) into B blocks ($\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(B)}$) then we can express the i^{th} component of the augmented Lagrangian as

$$\mathcal{L}_i(\boldsymbol{x}^{(i)}, \boldsymbol{z}, \boldsymbol{\mu}^{(i)}) = \mathbf{1}^{\mathrm{T}}(\boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)} + \mathbf{1})_+ + \frac{g(\boldsymbol{z})}{B} + \frac{\rho}{2}\|\boldsymbol{x}^{(i)} - \boldsymbol{z} + \boldsymbol{\mu}^{(i)}\|_2^2.$$

Note that the regularization does not include the last term in $\boldsymbol{z}$:

$$g(\boldsymbol{z}) = \frac{1}{2}\sum_{n=1}^{N} |z[n]|^2$$

This results in the ADMM iteration

$$\boldsymbol{x}_{k+1}^{(i)} = \arg\min_{\boldsymbol{x}^{(i)}} \left(\mathbf{1}^{\mathrm{T}}(\boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)} + \mathbf{1})_+ + \frac{\rho}{2}\|\boldsymbol{x}^{(i)} - \boldsymbol{z}_k + \boldsymbol{\mu}_k^{(i)}\|_2^2 \right),$$

$$\boldsymbol{z}_{k+1}[n] = \begin{cases} \frac{B\rho}{1+B\rho}\left(\bar{\boldsymbol{x}}_{k+1}[n] + \bar{\boldsymbol{\mu}}_k[n]\right), & n = 1, \ldots, N, \\ \bar{\boldsymbol{x}}_{k+1}[n] + \bar{\boldsymbol{\mu}}_k[n], & n = N+1, \end{cases}$$

$$\boldsymbol{\mu}_{k+1}^{(i)} = \boldsymbol{\mu}_k^{(i)} + \boldsymbol{x}_{k+1}^{(i)} - \boldsymbol{z}_{k+1}.$$

ADMM: Splitting across features

When we are solving problems of the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{A}\boldsymbol{x}) + g(\boldsymbol{x}),$$

where g is separable, which we are in all the examples above, we can also distribute the optimization program (again with a centralized coordinator) by dividing up the columns of $\boldsymbol{A}$. In learning this is often referred to as "splitting across features" as each agent will see a subset of the features for all of the data points.

If we can separate g into B blocks,

$$g(\boldsymbol{x}) = \sum_{i=1}^{B} g_i(\boldsymbol{x}^{(i)})$$

where $\boldsymbol{x}^{(i)} \in \mathbb{R}^{N_i}$, then we can do the same with the matrix product $\boldsymbol{A}\boldsymbol{x}$. We divide the $M \times N$ matrix $\boldsymbol{A}$ into B blocks with dimensions $M \times N_i$,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}^{(1)} & \boldsymbol{A}^{(2)} & \cdots & \boldsymbol{A}^{(B)} \end{bmatrix}$$

and then

$$\boldsymbol{A}\boldsymbol{x} = \sum_{i=1}^{B} \boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)}.$$

Our optimization problem is now

$$\underset{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(B)}}{\text{minimize}} \; f\left(\sum_{i1=}^{B} \boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)}\right) + \sum_{i=1}^{B} g_i(\boldsymbol{x}^{(i)}).$$

which we can recast as

$$\underset{\{\boldsymbol{x}^{(i)}\},\{\boldsymbol{z}^{(i)}\}}{\text{minimize}} \; f\left(\sum_{i=1}^{B} \boldsymbol{z}_i\right) + \sum_{i=1}^{B} g_i(\boldsymbol{x}^{(i)}) \quad \text{subject to} \quad \boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)} - \boldsymbol{z}^{(i)} = \boldsymbol{0}.$$

This gives us the ADMM updates

$$\begin{aligned}
\boldsymbol{x}_{k+1}^{(i)} &= \underset{\boldsymbol{x}^{(i)}\in\mathbb{R}^{N_i}}{\arg\min} \left(g_i(\boldsymbol{x}^{(i)}) + \frac{\rho}{2}\|\boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)} - \boldsymbol{z}_k^{(i)} + \boldsymbol{\mu}_k^{(i)}\|_2^2 \right) \\
\boldsymbol{z}_{k+1} &= \underset{\boldsymbol{z}=\{\boldsymbol{z}^{(i)}\}\in\mathbb{R}^{BM}}{\arg\min} \left(f\left(\sum_{i=1}^{B} \boldsymbol{z}^{(i)}\right) + \frac{\rho}{2}\sum_{i=1}^{B} \|\boldsymbol{A}^{(i)}\boldsymbol{x}_{k+1}^{(i)} - \boldsymbol{z}^{(i)} + \boldsymbol{\mu}_k^{(i)}\|_2^2 \right) \\
\boldsymbol{\mu}_{k+1}^{(i)} &= \boldsymbol{\mu}_k^{(i)} + \boldsymbol{A}^{(i)}\boldsymbol{x}_{k+1}^{(i)} - \boldsymbol{z}_{k+1}^{(i)}.
\end{aligned}$$

The $\boldsymbol{z}_k$ update, which is an optimization problem over BM variables, can actually be derived from an optimization over N variables: we can solve for the mean $\bar{\boldsymbol{z}}_{k+1}$ then recover the individual $\boldsymbol{z}_{k+1}^{(i)}$ from the mean. Indeed, if for any $\boldsymbol{z} \in \mathbb{R}^{BM}$ we let

$$\bar{\boldsymbol{z}} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{z}^{(i)},$$

then we can write (using the chain rule) the optimality condition for the optimization program for the $\boldsymbol{z}$ update above as

$$\boldsymbol{0} \in B\partial f(B\bar{\boldsymbol{z}}) + \rho(\boldsymbol{z}^{(i)} - \boldsymbol{A}^{(i)}\boldsymbol{x}_{k+1}^{(i)} - \boldsymbol{\mu}_k^{(i)}), \quad \text{for all} \;\; i = 1,\ldots,B.$$

Averaging over these B conditions, we also know that

$$\mathbf{0} \in B\partial f(B\bar{\boldsymbol{z}}) + \rho(\bar{\boldsymbol{z}} - \bar{\boldsymbol{b}}),$$

where

$$\bar{\boldsymbol{b}} = \frac{1}{B}\sum_{i=1}^{B}\left(\boldsymbol{A}^{(i)}\boldsymbol{x}_{k+1}^{(i)} + \boldsymbol{\mu}_k^{(i)}\right)$$

Thus is must be the case that

$$\rho(\bar{\boldsymbol{z}} - \bar{\boldsymbol{b}}) = \rho(\boldsymbol{z}^{(i)} - \boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k^{(i)}), \quad \text{for all} \;\; i = 1, \ldots, B,$$

or in other words for the optimal $\boldsymbol{z}$, we will have

$$\boldsymbol{z}^{(i)} = \bar{\boldsymbol{z}} - \bar{\boldsymbol{b}} + \boldsymbol{A}^{(i)}\boldsymbol{x}_{k+1}^{(i)} + \boldsymbol{\mu}_k^{(i)}. \tag{2}$$

Thus we can solve

$$\bar{\boldsymbol{z}}_{k+1} = \underset{\bar{\boldsymbol{z}}\in\mathbb{R}^M}{\arg\min}\left(f(B\bar{\boldsymbol{z}}) + \frac{B\rho}{2}\left\|\bar{\boldsymbol{z}} - \bar{\boldsymbol{b}}\right\|_2^2\right),$$

and then update

$$\boldsymbol{z}_{k+1}^{(i)} = \bar{\boldsymbol{z}}_{k+1} - \bar{\boldsymbol{b}} + \boldsymbol{A}^{(i)}\boldsymbol{x}_{k+1}^{(i)} + \boldsymbol{\mu}_k^{(i)}.$$

Also note that with (2), the Lagrange multiplier update becomes

$$\boldsymbol{\mu}_{k+1}^{(i)} = \bar{\boldsymbol{b}} - \bar{\boldsymbol{z}}_{k+1},$$

meaning that all B sets of Lagrange multipliers are equal to each other.

Distributed ADMM (splitting across features)

$$\boldsymbol{x}_{k+1}^{(i)} = \underset{\boldsymbol{x}^{(i)} \in \mathbb{R}^{N_i}}{\arg\min} \left(g_i(\boldsymbol{x}^{(i)}) + \frac{\rho}{2} \| \boldsymbol{A}^{(i)} \boldsymbol{x}^{(i)} - \boldsymbol{z}_k^{(i)} + \boldsymbol{\mu}_k \|_2^2 \right), \quad i = 1, \ldots, B$$

$$\bar{\boldsymbol{b}}_{k+1} = \frac{1}{B} \sum_{i=1}^{B} \left(\boldsymbol{A}^{(i)} \boldsymbol{x}_{k+1}^{(i)} . + \boldsymbol{\mu}_k \right)$$

$$\bar{\boldsymbol{z}}_{k+1} = \underset{\bar{\boldsymbol{z}} \in \mathbb{R}^M}{\arg\min} \left(f(B\bar{\boldsymbol{z}}) + \frac{B\rho}{2} \left\| \bar{\boldsymbol{z}} - \bar{\boldsymbol{b}}_{k+1} \right\|_2^2 \right)$$

$$\boldsymbol{z}_{k+1}^{(i)} = \bar{\boldsymbol{z}}_{k+1} - \bar{\boldsymbol{b}}_{k+1} + \boldsymbol{A}^{(i)} \boldsymbol{x}_{k+1}^{(i)} + \boldsymbol{\mu}_k, \quad i = 1, \ldots, B$$

$$\boldsymbol{\mu}_{k+1} = \bar{\boldsymbol{b}}_{k+1} - \bar{\boldsymbol{z}}_{k+1}.$$

To make the communication explicit, one way this can work is

1. Each agent solves for $\boldsymbol{x}_{k+1}^{(i)}$ in parallel.
2. The quantities $\boldsymbol{A}^{(i)} \boldsymbol{x}_{k+1}^{(I)} + \boldsymbol{\mu}_{k+1}$ are communicated to the coordinator.
3. The coordinator computes $\bar{\boldsymbol{b}}_{k+1}$ and then solves the optimization program to compute $\bar{\boldsymbol{z}}_{k+1}$.
4. The coordinator updates each $\boldsymbol{z}_{k+1}^{(i)}$ and the $\boldsymbol{\mu}_{k+1}$ and communicates them back to the agents.
5. Repeat.

Note that the coordinator could also broadcast the $\bar{\boldsymbol{z}}_{k+1}, \bar{\boldsymbol{b}}_{k+1}$ to all of the agents, an they could update the $\boldsymbol{z}_{k+1}^{(i)}$ and $\boldsymbol{\mu}_{k+1}$ individually.

Example: the LASSO

Suppose we want to solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \underbrace{\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2}_{f(\boldsymbol{A}\boldsymbol{x})} + \underbrace{\lambda\|\boldsymbol{x}\|_1}_{g(\boldsymbol{x})}$$

by distributing across features. In this case, the ADMM iterations above reduce to

$$\begin{aligned}
\boldsymbol{x}_{k+1}^{(i)} &= \underset{\boldsymbol{x}^{(i)}\in\mathbb{R}^{N_i}}{\arg\min}\left(\lambda\|\boldsymbol{x}^{(i)}\|_1 + \frac{\rho}{2}\left\|\boldsymbol{A}^{(i)}\boldsymbol{x}^{(i)} - \boldsymbol{A}^{(i)}\boldsymbol{x}_k^{(i)} - \bar{\boldsymbol{z}}_k + \bar{\boldsymbol{b}}_k\right\|_2^2\right) \\
\bar{\boldsymbol{z}}_{k+1} &= \frac{1}{B+\rho}\left(\boldsymbol{y} + \rho\bar{\boldsymbol{b}}_{k+1}\right) \\
\boldsymbol{\mu}_{k+1} &= \bar{\boldsymbol{b}}_k - \bar{\boldsymbol{z}}_{k+1}.
\end{aligned}$$

Note that in this case, each of the agents has to solve a smaller LASSO problem at every iteration to do the $\boldsymbol{x}^{(i)}$ updates.

References

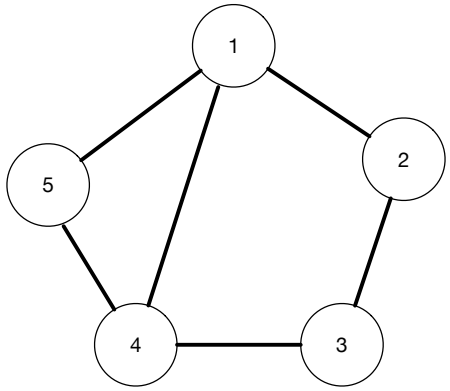
[BPC+10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.

Decentralized Optimization

In this set of notes, we again look at solving problems of the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \sum_{i=1}^{B} f_i(\boldsymbol{x}), \quad (+ \text{ a regularizer, possibly})$$

Here, though, we will assume that the B agents, each of which has access to one of the f_i, can only communicate on a decentralized network:



To warm up we will start by discussion one of the most fundamental problems in network science.

Network consensus

Consider the following problem. We have B agents arranged in a network as in the example above., each of which holds a vector $\boldsymbol{x}^{(i)}$. The agents can communicated with their neighbors on the graph above. The goal is for the agents to figure out what the *average* of $\boldsymbol{x}^{(i)}$ is; that is, each one want to learn

$$\bar{\boldsymbol{x}} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{x}^{(i)}.$$

Let's make all of the above a little more precise. The network can be described by a graph $(\mathcal{V}, \mathcal{E})$ where the B vertices in $\mathcal{V}$ are in one-to-one correspondence with the agents, and the edges $\mathcal{E}$ dictate which agents can communicate with one another. In the picture above

$$\begin{aligned}\mathcal{V} &= \{1,2,3,4,5,6\},\\ \mathcal{E} &= \{(1,2),(1,4),(1,5),(2,3),(3,4),(4,5),\\ &\qquad (2,1),(4,1),(5,1),(3,2),(4,3),(5,4)\}.\end{aligned}$$

Note that the graph is *undirected*, so $(i,j) \in \mathcal{E} \Leftrightarrow (j,i) \in \mathcal{E}$. We will also use $\mathcal{N}(i)$ to denote the *neighborhood* of agent i, that is all other agents connected to i along with i itself

$$\mathcal{N}(i) = \{j : (i,j) \in \mathcal{E}\} \cup \{i\}.$$

So again in the example above

$$\mathcal{N}(1) = \{1,2,4,5\}, \quad \mathcal{N}(2) = \{1,2,3\}, \quad \text{etc.}$$

You can probably think of numerous schemes that can be used to compute the average, but here is a relatively simple way that is extremely effective and resource-lite. The agents simply take an average of their current estimate with their neighbors', then repeat. That is, we take

$$\boldsymbol{z}_0^{(i)} = \boldsymbol{x}^{(i)},$$

then each agents compute

$$\boldsymbol{z}_{k+1}^{(i)} = \sum_{j \in \mathcal{N}(i)} w_{i,j} \boldsymbol{z}_k^{(j)}, \quad i = 1, \ldots, B, \tag{1}$$

where the weights $w_{i,j}$ obey

$$\sum_{j \in \mathcal{N}(i)} w_{i,j} = 1, \quad i = 1, \ldots, B, \tag{2}$$

and $w_{i,j} \geq 0$. For simplicity in the exposition below, we will also assume that $w_{i,j} = w_{j,i}$, although all the essentials still hold if this is not the case. As the sum for agent i in (1) involves only their own state $\boldsymbol{z}_k^{(i)}$ and that of its neighbors, it can be executed with communication over edges on the graph. We will see that this simple iteration results in all of the $\boldsymbol{z}_k^{(i)} \to \bar{\boldsymbol{x}}$ as $k \to \infty$. While the rate of convergence will depend on the structure of the graph, we will also see that it is in general very fast.

We can write the iteration (1) as a vector-matrix product. To simplify notation here, we will assume that $N = 1$, so each $\boldsymbol{x}^{(i)} = x^{(i)}$ is a scalar; nothing changes in our discussion below for $N > 1$ except the notation. Collect the weights $\{w_{i,j},\ (i,j) \in \mathcal{E}\}$ into the $B \times B$ matrix $\boldsymbol{W}$ with

$$W_{i,j} = \begin{cases} w_{i,j}, & (i,j) \in \mathcal{E}, \\ 0, & (i,j) \notin \mathcal{E}. \end{cases}$$

We collect the $\boldsymbol{z}^{(i)} = z^{(i)} \in \mathbb{R}$ into a vector of length B,

$$\boldsymbol{z} = \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(B)} \end{bmatrix}. \tag{3}$$

Then (1) becomes

$$\boldsymbol{z}_{k+1} = \boldsymbol{W}\boldsymbol{z}_k. \tag{4}$$

We have now turned this problem into one of the most fundamental constructs in applied mathematics: a linear dynamical system. We can learn almost all there is to know about the convergence of such a system from the eigenstructure of $\boldsymbol{W}$ (which is in turn related to the geometric structure of the graph).

We start the convergence analysis by noting one particular fact: that the vector $\mathbf{1} \in \mathbb{R}^B$ of all ones is an eigenvector with eigenvalue 1,

$$\boldsymbol{W}\mathbf{1} = \mathbf{1}.$$

This follows directly from the fact that every row of $\boldsymbol{W}$ sums[1] to 1 as dictated by (2). A slightly less obvious fact is that all of the eigenvalues of $\boldsymbol{W}$ must have magnitude ≤ 1. To see this, suppose that $\boldsymbol{v}$ is an eigenvector of $\boldsymbol{W}$ with eigenvalue λ, so $\boldsymbol{W}\boldsymbol{v} = \lambda\boldsymbol{v}$. Then is must be true that for every $i = 1, \ldots, B$

$$\begin{aligned}\lambda v_i &= \sum_{j=1}^{B} W_{i,j} v_j \\ &\leq \max_j |v_j|,\end{aligned}$$

which can only be true if $|\lambda| \leq 1$. So we can take as the eigenvalue decomposition $\boldsymbol{W} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}$, where $\mathrm{diag}(\Lambda) = \{\lambda_1, \lambda_2, \ldots, \lambda_B\}$ with $\lambda_1 = 1$ and the rest of the eigenvalues sorted by magnitude, $1 \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_B|$.

We now show that (4) will result in every entry in $\boldsymbol{z}_k$ to converge to the mean $\bar{x}$, that is $\boldsymbol{z}_k \to \bar{x}\mathbf{1}$. We have

$$\begin{aligned}\|\boldsymbol{z}_k - \bar{x}\mathbf{1}\|_2 &= \|\boldsymbol{W}\boldsymbol{z}_{k-1} - \bar{x}\mathbf{1}\|_2 \\ &= \|\boldsymbol{W}(\boldsymbol{z}_{k-1} - \bar{x}\mathbf{1})\|_2 \quad (\text{since } \boldsymbol{W}\mathbf{1} = \mathbf{1}) \\ &\leq |\lambda_2| \, \|\boldsymbol{z}_{k-1} - \bar{x}\mathbf{1}\|_2 \quad (\text{see below}) \\ &\leq |\lambda_2|^k \, \|\boldsymbol{z}_0 - \bar{x}\mathbf{1}\|_2 \\ &= |\lambda_2|^k \, \|\boldsymbol{x} - \bar{x}\mathbf{1}\|_2,\end{aligned}$$

where λ_2 is the second largest magnitude eigenvalue of $\boldsymbol{W}$. To see why the first inequality above holds, note that $\frac{1}{B}\mathbf{1}^{\mathrm{T}}\boldsymbol{z}_k = \bar{x}$ for

[1] Matrices like these are called *row stochastic*.

all k (the average of the state at each agent is invariant). Thus $\mathbf{1}^{\mathrm{T}}(\boldsymbol{z}_{k-1} - \bar{x}\mathbf{1}) = 0$, and $\boldsymbol{z}_{k-1} - \bar{x}\mathbf{1}$ is orthogonal to the eigenvector corresponding to the largest eigenvalue λ_1, and the most applying $\boldsymbol{W}$ can grow its norm is the magnitude of the second largest eigenvalue.

We can see from above that we have linear convergence when $|\lambda_2| < 1$. The magnitude of this eigenvalue obviously depends on the weight matrix $\boldsymbol{W}$, both in its support structure (the locations of its non-zero terms, as dictated by the edges $\mathcal{E}$ of the graph). In fact, the quantity $1 - \lambda_2$ is often referred to as the *spectral gap* of the graph.

[Spectral gap examples]

This all extends very naturally to the case where $N > 1$. Now the vector $\boldsymbol{z}$ in (3) becomes a block vector of size BN:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{z}^{(1)} \\ \boldsymbol{z}^{(2)} \\ \vdots \\ \boldsymbol{z}^{(B)} \end{bmatrix}.$$

The update in (1) can still be written as a matrix equation; we take $\tilde{\boldsymbol{W}}$ to be the $BN \times BN$ matrix $\tilde{\boldsymbol{W}} = \boldsymbol{W} \otimes \mathbf{I}$, where $\otimes$ is the tensor product. To create $\tilde{\boldsymbol{W}}$, we simply replace each element of $\boldsymbol{W}$

with the corresponding scalar multiple of the identity. For example, if $N = B = 2$ and

$$\boldsymbol{W} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \text{then} \quad \tilde{\boldsymbol{W}} = \begin{bmatrix} a\mathbf{I} & b\mathbf{I} \\ c\mathbf{I} & d\mathbf{I} \end{bmatrix} = \begin{bmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ c & 0 & d & 0 \\ 0 & c & 0 & d \end{bmatrix}.$$

It should be clear that this simply means we are averaging all of the individual elements in the $\boldsymbol{z}^{(k)}$ in exactly the same way. The eigenstructure of $\tilde{\boldsymbol{W}}$ is also closely related to that of $\boldsymbol{W}$: it has the same B eigenvalues, but they are each repeated N times. The convergence rate of consensus is determined by the second largest unique eigenvalue.

Decentralized gradient algorithms

We can also use a variation on network consensus to solve optimization programs of the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \sum_{i=1}^{B} f_i(\boldsymbol{x}). \tag{5}$$

As before, we will assume agent i has access to the function f_i and can communicate with its neighbors $\mathcal{N}(i)$ on a connected graph.

The decentralized gradient algorithm is simple: at each iteration, you average your current state with your neighbors (as in consensus), then take a gradient step. As before, each agent has their own version of the decision variables $\boldsymbol{x}^{(i)}$. The main iteration, executed at each agent simultaneously, is

$$\boldsymbol{x}_{k+1}^{(i)} = \sum_{j \in \mathcal{N}(i)} w_{i,j} \boldsymbol{x}_k^{(j)} - \alpha_k \boldsymbol{g}_k^{(i)}, \quad i = 1, \ldots, B, \tag{6}$$

where $\boldsymbol{g}_k^{(i)} \in \partial f_i(\boldsymbol{x}_k^{(i)})$. For convex f_i, and for appropriately decreasing α_k, the iteration (6) converges as [NO09]

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \;\leq\; O\left(\frac{\log k}{\sqrt{k}}\right).$$

At the solution, we will have $\boldsymbol{x}_\star^{(1)} = \boldsymbol{x}_\star^{(2)} = \cdots = \boldsymbol{x}_\star^{(B)}$ and the $\boldsymbol{x}_\star^{(i)}$ will solve (5). This rate of convergence is essentially the same as what we saw for the centralized case.

When each f_i is L-smooth (has a Lipschitz gradient) and is μ-strongly convex, we can modify the iteration as

$$\boldsymbol{x}_{k+1}^{(i)} = \sum_{j \in \mathcal{N}(i)} w_{i,j} \boldsymbol{x}_k^{(j)} - \alpha_k \nabla f_i(\boldsymbol{x}_k^{(i)}), \quad i = 1, \ldots, B. \tag{7}$$

However, to get the algorithm to converge, we still need to decrease the step size, taking $\alpha_k \to 0$ (recall that in the centralized case, we got linear convergence for smooth and strongly convex f with a fixed step size). The reason for this is that at the solution $\boldsymbol{x}_\star$ we will have $\sum_i \nabla f_i(\boldsymbol{x}_\star) = \mathbf{0}$ but not necessarily $\nabla f_i(\boldsymbol{x}_\star) = \mathbf{0}$. Decreasing the step size in this manner greatly hinders the convergence rate, keeping the iteration (7) from approaching the solution at a linear rate (the convergence rate is $O(1/k^{2/3})$ which is much slower that $O(\beta^{-k})$).

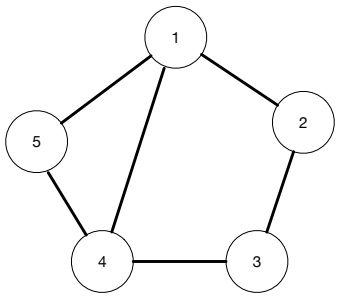
Fortunately, the iteration above is easily modified for smooth and strongly convex functions. In [QL18], it is shown that the iteration

$$\begin{aligned}
\boldsymbol{x}_{k+1}^{(i)} &= \sum_{j \in \mathcal{N}(i)} w_{i,j} \boldsymbol{x}_k^{(j)} - \alpha \boldsymbol{s}_k^{(i)} \\
\boldsymbol{s}_{k+1}^{(i)} &= \sum_{j \in \mathcal{N}(i)} w_{i,j} \boldsymbol{s}_k^{(j)} + \nabla f_i(\boldsymbol{x}_{k+1}^{(i)}) - \nabla f_i(\boldsymbol{x}_k^{(i)})
\end{aligned}$$

will converge to $\boldsymbol{x}_k^{(i)} \to \boldsymbol{x}_\star$ and $\boldsymbol{s}_k^{(i)} \to \mathbf{0}$ at a linear rate. Note, however, that now we need to communicate two vectors, $\boldsymbol{x}^{(i)}$ and $\boldsymbol{s}^{(i)}$, between neighbors at every step.

Consensus as Optimization

Let's return to the consensus problem and look at it from another point of view, one that will lead us to an optimization program. Suppose again that we have B agents who can communicate on a graph:



As before, each agent is in control of a vector $\boldsymbol{x}^{(i)}$. If the graph is connected (meaning that there is a path from every node to every other node), then the condition that all $\boldsymbol{x}^{(i)}$ are equal to one another

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(2)} = \cdots = \boldsymbol{x}^{(B)},$$

can be rewritten as

$$\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(j)} \quad \text{for all } (i,j) \in \mathcal{E}.$$

For example, for the graph above

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(2)} = \boldsymbol{x}^{(3)} = \boldsymbol{x}^{(4)} = \boldsymbol{x}^{(5)} \tag{8}$$

if and only if

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(2)}, \quad \boldsymbol{x}^{(2)} = \boldsymbol{x}^{(3)} \quad \boldsymbol{x}^{(3)} = \boldsymbol{x}^{(4)}$$
$$\boldsymbol{x}^{(4)} = \boldsymbol{x}^{(1)}, \quad \boldsymbol{x}^{(4)} = \boldsymbol{x}^{(5)}, \quad \text{and} \quad \boldsymbol{x}^{(5)} = \boldsymbol{x}^{(1)}.$$

Another way to say this is that we have *consensus* (meaning (8)) if and only if the $\{\boldsymbol{x}^{(i)}\}$ are a solution to the optimization program

$$\underset{\{\boldsymbol{x}^{(i)}\}_{i=1}^{B}}{\text{minimize}} \sum_{(i,j)\in\mathcal{E}} \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2^2 = \frac{1}{2}\sum_{i=1}^{B}\sum_{j\in\mathcal{N}(i)} \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2^2. \tag{9}$$

Obviously the solution to the above is not unique, but then neither are the $\{\boldsymbol{x}^{(i)}\}$ that obey the equality constraints (8). We start with an application where solving this type of is useful.

Swarm robotics

Suppose that we have B robots with positions $\boldsymbol{p}^{(1)}, \boldsymbol{p}^{(2)}, \ldots, \boldsymbol{p}^{(B)}$, where each $\boldsymbol{p}^{(i)}$ is a vector in $\mathbb{R}^N$, with $N = 2$ or 3, depending on the application. Suppose that we want these robots to meet at the same location. We do not care where this is, we simply want the robots to all converge to the same point. We can pose this as the solution to a convex optimization problem. Specifically, set

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}^{(1)} \\ \boldsymbol{p}^{(2)} \\ \vdots \\ \boldsymbol{p}^{(B)} \end{bmatrix},$$

so that $\boldsymbol{x} \in \mathbb{R}^{NB}$. As above, for each robot we define a neighborhood $\mathcal{N}(i)$ corresponding to the robots to which robot i can measure its

relative position. In other words, if $j \in \mathcal{N}(i)$, robot i can compute the vector $\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}$. We will assume for the sake of simplicity that these are symmetric in the sense that $j \in \mathcal{N}(i)$ if and only if $i \in \mathcal{N}(j)$. We would like all of these distances to be zero, so a natural objective function that we might want to minimize is

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \sum_{j \in \mathcal{N}(i)} \|\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}\|_2^2.$$

We can compute the gradient of this function by noting that

$$\nabla_{\boldsymbol{p}^{(i)}} f(\boldsymbol{x}) = \sum_{j \in \mathcal{N}(i)} 2(\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}) + \sum_{j \,:\, i \in \mathcal{N}(j)} 2(\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}).$$

With our assumption that the neighborhoods are symmetric, this simplifies to

$$\nabla_{\boldsymbol{p}^{(i)}} f(\boldsymbol{x}) = 4 \sum_{j \in \mathcal{N}(i)} (\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}).$$

Putting this all together, we can write

$$\nabla f(\boldsymbol{x}) = 4 \begin{bmatrix} \sum_{j \in \mathcal{N}(1)} (\boldsymbol{p}^{(1)} - \boldsymbol{p}^{(j)}) \\ \sum_{j \in \mathcal{N}(2)} (\boldsymbol{p}^{(2)} - \boldsymbol{p}^{(j)}) \\ \vdots \\ \sum_{j \in \mathcal{N}(B)} (\boldsymbol{p}^{(B)} - \boldsymbol{p}^{(j)}) \end{bmatrix}.$$

In this case the gradient descent update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$ nicely de-couples so that the i^{th} robot has the update rule (ignoring the multiplicative factor of 4):

$$\boldsymbol{p}_{k+1}^{(i)} = \boldsymbol{p}_k^{(i)} - \alpha_k \sum_{j \in \mathcal{N}(i)} (\boldsymbol{p}_k^{(i)} - \boldsymbol{p}_k^{(j)}).$$

This update rule plays a fundamental role in many swarm robotics problems and is known as the **consensus equation**. Note that the update for each robot depends only on *local* information (the difference between its own position and that of its neighbors), and hence each robot can compute its own update without any form of global coordination.

We know from our knowledge of gradient descent that for appropriately chosen α_k, this algorithm is guaranteed to converge. We can also see that every global optimum of this problem will have all the robots to converging to the same point.

Gradient descent for consensus optimization

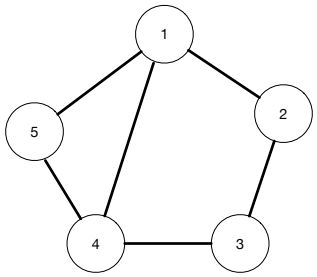
Suppose again (temporarily) that $N = 1$. Then for a given graph, the function

$$\sum_{(i,j)\in\mathcal{E}} (x^{(i)} - x^{(j)})^2 = \frac{1}{2}\sum_{i=1}^{B}\sum_{j\in\mathcal{N}(i)} (x^{(i)} - x^{(j)})^2 = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{x}, \qquad (10)$$

where $\boldsymbol{L}$ is the *graph Laplacian*. The graph Laplacian is constructed from the *degree matrix* $\boldsymbol{D}$ and *adjacency matrix* $\boldsymbol{A}$, where

$$D_{ii} = \text{number of edges into node } i, \quad A_{ij} = \begin{cases} 1, & (i,j) \in \mathcal{E}, \\ 0, & \text{otherwise}, \end{cases}$$

and $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$. For our example graph



we have

$$\boldsymbol{D} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} 3 & -1 & 0 & -1 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ -1 & 0 & -1 & 3 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix}.$$

It takes a little work to check the relation (10), and I encourage you to put in this work at home.

The gradient decent step for solving (9) can thus be written

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{L}\boldsymbol{x}.$$

As noted in the swarm robotics example above, the ith component of this gradient can be computed from knowledge of just the $x^{(j)} \in \mathcal{N}(i)$.

Although the solution to (9) is not unique, we can show that the gradient descent algorithm will converge to a unique point that we can specify. Notice that by construction, the vector of all ones is in the null space of $\boldsymbol{L}$:

$$\boldsymbol{L}\mathbf{1} = \mathbf{0}.$$

Thus since $\boldsymbol{L}$ is symmetric, $\mathbf{1}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{x} = \mathbf{0}$ for all $\boldsymbol{x}$, and

$$\mathbf{1}^{\mathrm{T}}\boldsymbol{x}_{k+1} = \mathbf{1}^{\mathrm{T}}\boldsymbol{x}_k - \alpha_k \mathbf{1}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{x}_k = \mathbf{1}^{\mathrm{T}}\boldsymbol{x}_k.$$

That is, the sum of the entries in the $\boldsymbol{x}_k$ *does not change* during gradient descent. If the nodes are initialized with

$$\boldsymbol{x}_0 = \begin{bmatrix} x_0^{(1)} \\ x_0^{(2)} \\ \vdots \\ x_0^{(B)} \end{bmatrix},$$

then gradient descent converges to a vector whose entries are all equal to one another and whose sum is the same as $\boldsymbol{x}_0$,

$$\boldsymbol{x}_\star = \bar{x}\mathbf{1}, \quad \bar{x} = \frac{1}{B}\sum_{i=1}^{B} x_0^{(i)}.$$

We see then that solving (9) using gradient descent gives us the same answer as the network consensus iteration (1) while using the same communication structure.

Again, for $N > 1$, we can solve (9) in the same way with a Laplacian matrix that has been tensored up, $\tilde{\boldsymbol{L}} = \boldsymbol{L} \otimes \mathbf{I}$, just as we discussed for network consensus.

A primal-dual method for decentralized optimization

There is another approach to decentralized optimization that is easily adapted to f_i that are not smooth. As we discussed in the last section, the consensus condition (8) is equivalent to $\tilde{\boldsymbol{L}}\boldsymbol{x} = \mathbf{0}$, meaning that

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \sum_{i=1}^{B} f_i(\boldsymbol{x})$$

can be re-written as

$$\underset{\{\boldsymbol{x}^{(i)}\}_{i=1}^B}{\text{minimize}} \sum_{i=1}^{B} f_i(\boldsymbol{x}^{(i)}) \quad \text{subject to} \quad \tilde{\boldsymbol{L}}\boldsymbol{x} = \mathbf{0}. \tag{11}$$

Recall the primal-dual proximal algorithm from [CP11] for solving problems of the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^{BN}}{\text{minimize}}\, f(\boldsymbol{K}\boldsymbol{x}) + g(\boldsymbol{x}),$$

through the iteration

$$\begin{aligned}\boldsymbol{\nu}_{k+1} &= \text{prox}_{\sigma f^*}(\boldsymbol{\nu}_k + \sigma \boldsymbol{K}\boldsymbol{x}_k)\\ \boldsymbol{x}'_{k+1} &= \text{prox}_{\alpha g}(\boldsymbol{x}_k - \alpha \boldsymbol{K}^{\mathrm{T}}\boldsymbol{\nu}_{k+1})\\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}'_{k+1} + \theta(\boldsymbol{x}'_{k+1} - \boldsymbol{x}'_k),\end{aligned}$$

where σ, α are step sizes and θ is an acceleration parameter. We can apply this algorithm to (11) by taking

$$\boldsymbol{K} = \tilde{\boldsymbol{L}}, \quad f(\boldsymbol{w}) = \begin{cases} 0, & \boldsymbol{w} = 0, \\ \infty, & \text{otherwise}, \end{cases} \quad g(\boldsymbol{x}) = \sum_{i=1}^{B} f_i(\boldsymbol{x}^{(i)}).$$

The Fenchel conjugate of f in this case is

$$f^*(\boldsymbol{\nu}) = \sup_{\boldsymbol{w}\in\mathbb{R}^N} \left(\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{w} - f(\boldsymbol{w})\right) = \boldsymbol{0}, \quad \text{for all} \quad \boldsymbol{\nu} \in \mathbb{R}^N,$$

and so

$$\text{prox}_{\sigma f^*}(\boldsymbol{z}) = \arg\min_{\boldsymbol{w}} \left(f^*(\boldsymbol{w}) + \frac{1}{2\sigma}\|\boldsymbol{w} - \boldsymbol{z}\|_2^2 \right) = \boldsymbol{z}.$$

Also note that

$$\text{prox}_{\alpha g}(\boldsymbol{z}) = \begin{bmatrix} \text{prox}_{\alpha f_1}(\boldsymbol{z}^{(1)}) \\ \text{prox}_{\alpha f_2}(\boldsymbol{z}^{(2)}) \\ \vdots \\ \text{prox}_{\alpha f_B}(\boldsymbol{z}^{(B)}) \end{bmatrix},$$

and, due to the structure of $\tilde{\boldsymbol{L}}$, that the matrix-vector product $\boldsymbol{\nu} = \tilde{\boldsymbol{L}}\boldsymbol{x}$ can be written as

$$\begin{bmatrix} \boldsymbol{\nu}^{(1)} \\ \boldsymbol{\nu}^{(2)} \\ \vdots \\ \boldsymbol{\nu}^{(B)} \end{bmatrix} = \begin{bmatrix} L_{11}\boldsymbol{x}^{(1)} - \sum_{j\in\mathcal{N}(1), j\neq 1} L_{ij}\boldsymbol{x}^{(j)} \\ L_{22}\boldsymbol{x}^{(2)} - \sum_{j\in\mathcal{N}(2), j\neq 2} L_{2j}\boldsymbol{x}^{(j)} \\ \vdots \\ L_{BB}\boldsymbol{x}^{(B)} - \sum_{j\in\mathcal{N}(B), j\neq B} L_{Bj}\boldsymbol{x}^{(j)} \end{bmatrix} = \begin{bmatrix} \sum_{j\in\mathcal{N}(1)} L_{1j}\boldsymbol{x}^{(j)} \\ \sum_{j\in\mathcal{N}(2)} L_{2j}\boldsymbol{x}^{(j)} \\ \vdots \\ \sum_{j\in\mathcal{N}(B)} L_{Bj}\boldsymbol{x}^{(j)} \end{bmatrix}$$

This gives us the iteration

$$\boldsymbol{\nu}_{k+1}^{(i)} = \boldsymbol{\nu}_k^{(i)} + \sigma \sum_{j \in \mathcal{N}(i)} L_{ij} \boldsymbol{x}_k^{(j)}, \quad i = 1, \ldots, B$$

$$\boldsymbol{x}_{k+1}^{'(i)} = \text{prox}_{\alpha f_i} \left(\boldsymbol{x}_k^{(i)} - \alpha \sum_{j \in \mathcal{N}(i)} L_{ij} \boldsymbol{\nu}_{k+1}^{(j)} \right), \quad i = 1, \ldots, B,$$

$$\boldsymbol{x}_{k+1}^{(i)} = \boldsymbol{x}_{k+1}^{'(i)} + \theta(\boldsymbol{x}_{k+1}^{'(i)} - \boldsymbol{x}_k^{'(i)}), \quad i = 1, \ldots, B.$$

The flow of this decentralized optimization algorithm is

1. communication of $\{\boldsymbol{x}_k^{(i)}\}$ between neighbors on graph
2. each agent updates $\boldsymbol{\nu}_{k+1}^{(i)}$ in parallel
3. communication of $\{\boldsymbol{\nu}_{k+1}^{(i)}\}$ between neighbors on graph
4. each agent solves a prox problem to update $\boldsymbol{x}_{k+1}^{'(i)}$ in parallel
5. each agent updates $\boldsymbol{x}_{k+1}^{(i)}$ in parallel

So just as in the centralized case, we can break down the solution of (5) to solving a series of subproblems in parallel. This kind of technique is useful when the f_i are expensive to compute or the knowledge of the f_i is restricted to a single agent.

The convergence rate for this algorithm follows directly from the results for the general iteration. Basically, these rates all match the accelerated centralized results with constants that depend on the graph structure (i.e. the spectral gap). Details can be found in [CP11, CP16].

References

[CP11] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Imaging and Vision*, 40(1):120–145, 2011.

[CP16] A. Chambolle and T. Pock. On the ergodic convergence rates of a first-order primal-dual algorithm. *Math. Program., Ser. A*, 159:253–287, 2016.

[NO09] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Auto. Control*, 54(1):48–61, 2009.

[QL18] G. Qu and N. Li. Harnessing smoothness to accelerate distributed optimization. *IEEE Trans. Control Net. Sys.*, 5(3):1245–1260, 2018.