CME 364A: Convex Optimization I

Lecturer: Professor Stephen Boyd

Notes by: Andrew Lin

Winter 2026

Introduction

The course website will be pretty static (updated once a week), and all of the important information will be found on Ed. Course staff can also be contacted either on Ed or by emailing the course staff mailing list (rather than emailing individual people).

We'll have weekly homework (the first of which is posted), a midterm quiz, and a final exam. In the past, the final was a 24-hour take-home exam where we could actually do something, but this year we're going back to an in-class final exam (which is kind of a bummer, but it's necessary because of the rise of LLMs). So the point is that we won't need to know all of the annoying syntax for code and so on when we do this in a summer internship (which is great!), but we do need to be able to tell whether something that is generated is nonsense or not. So the LLM policy for this class is that we're encouraged, and in fact recommended, to use them, but we should really attempt or do the homework ourselves first.

The next three weeks or so of the course will be nothing but math, with some hints to things that might be useful, and then the seven weeks after that will actually be useful. And we'll go fast enough that no one can really absorb it, but once we see it in applications we'll get it all. The midterm will basically test the basic rules of convex analysis with no attempt to make it interesting – this is kind of like learning how to take the integral of $t^2 \sin t$ or something, though it's going to be more structured than something like ordinary calculus. Then the final exam will be in a normal three-hour slot for the first time for Professor Boyd in 40 years. Grading will be something like 10% homework and 90% exams, though it will take us many hours to do the problem sets, and there will be real substantial applications.

This course will also have weekly problem-solving sections taught by Professor **Babak Ayazifar** which will be related to the homework. And we're of course encouraged to work in groups so that we can learn better.

We will be writing "baby scripts" in this class (they don't really cross the threshold to programming) using CVXPY. There are packages in many other languages as well, but we'll only use this one; once we know how one works, we basicalaly understand all of them.

1 January 6, 2026

We'll do an overview lecture today, starting with the high-level concept of "why we are here." In an **optimization problem**, we want to choose some vector-valued variable $x \in \mathbb{R}^n$ by optimizing some **objective function** $f_0(x)$ (the smaller it is, the better we do), subject to some **inequality constraints** $f_i(x) \leq 0$ and some **equality constraints** $g_i(x) = 0$. (We can think of constraints as hard requirements and the objective function as a soft requirement; we have guarantees on the former but not the latter.) We also don't care about the margin on our constraints. There are of course various variations (maximizing objective or using multiple objective functions) depending on the application or the field.

The reason we want to solve optimization problems is that it's a formalization of **making good choices**: think of x as representing an action, like the trades of a portfolio, control surface deflections of an airplane, schedules or assignments of a surgical center, or resource allocation to a data center. In each of these situations, we have some limits on the possible actions or conditions on the possible outcome, and conditioned on those limits we want to make our objective (negative profit, fuel use, deviation from some desired target outcome, and so on) as small as we can. And we will actually write code to solve problems for all of these situations in this course.

But we can also take another perspective and have x represent the **parameters of a model** (statistical or otherwise) instead, where our constraints are now some requirements like nonnegativity. (For example, if we're estimating a Gaussian, we need its covariance to have nonnegative eigenvalues.) The objective in these "fitting" or "learning" problems is usually some sum of prediction error (loss) and regularization, the latter of which penalizes model complexity. The point is that we very rarely just care about fitting past data well (unless we're doing something like data compression) – we actually want to be able to make good predictions for future data. So we want to avoid overfitting by keeping in mind that our training data is only a surrogate for our true problem.

Yet another area is **worst-case analysis**, in which we interpret x not as something that we control but as something under the control of an adversary. The idea then is that constraints limit the possible values of parameters, and so the worst possible ones come from minimizing $-f_0(x)$. If that worst possible value is still tolerable, then it's okay. Keep in mind that even if we sample millions of points from simulations, it'll catch some very egregious cases but is still extremely empty in high-dimensional problems. So doing an actual optimization problem will be much better, for example if we want to know whether there are any circumstances where a combination of many errors could cause an airplane to crash.

And one more area is that we sometimes use optimization to **build models**. The idea is that entities (agents) take actions that solve some optimization problems to maximize some expected utility or reproductive success in biology or power in a circuit. The very interesting thing is that if you model a cell by simply maximizing growth rate subject to constraints on reaction rates (ignoring all of the biology and chemistry), it's going to be a very crude model, and yet it actually works very well in practice! In particular, it gives us a way to tell whether cells will die off even if we remove some set of reactions.

Remark 1. *Many fields of applied math have very different "mathematical accents," and when we're in a mixed group like this course, we switch to the language we all speak, which is mathematics. So in theory, anyone can speak to anyone else and understand what we're talking about.*

So the point is this: instead of saying how to an action or model x, we state some problem which articulates what we want, and then an algorithm decides on x instead. (And there isn't really any depth in what this is saying.) The problem is that in general **we cannot solve optimization exactly** (and often in classes we get bogged down in the math rather than understanding whether we're solving the real problem – we can usually just solve approximately instead). But the exception is **convex optimization**, which includes a big class of problems including linear programming and quadratic programming, in which we can solve problems reliably and efficiently. In embedded systems (e.g. a self-driving car or an airplane making decisions), we often can't have human intervention and we need to be completely reliable.

In nonlinear optimization, we usually end up finding local optima (meaning that we find a point where we're minimizing f_0 among points nearby) but this requires an initial guess and generally babysitting by tuning parameters and learning rates, and it provides no information about how suboptimal things are. Of course, in high-value problems, this is important and we should indeed do this. But then global optimization problems are basically universally based on solving convex subproblems.

Definition 2

In a **convex optimization** problem, we place curvature constraints on our constraints. That is, we have equality constraints which are linear, meaning that they are of the form $Ax = b$, and the objective f_0 and the inequality constraints $f_i(x) \le 0$ are **convex**, meaning that for any $\theta \in [0, 1]$ and any x, y.

$$f_i(\theta x + (1-\theta)y) \le \theta f_i(x) + (1-\theta)f_i(y).$$

Convex problems are much more tractable than nonconvex ones. It might feel weird that there are these asymmetric conditions (as in, replacing convexity with concavity or adding a negative sign suddenly make the problem intractable), but it turns out that's just the way it is – problems turn out to arise in this way as well.

The classical view is that "linear problems" (zero curvature) is easy, but nonlinear problems are hard. This is wrong – it's that nonnegative curvature is easy but negative curvature is hard. Convex optimization problems can be solved with many different algorithms (interior-point methods for up to tens of thousands of variables, and then first-order methods like gradient descent for larger ones), and these algorithms **do not require** any babysitting or tuning or initial points. So there's also reliable solve times and so they can be embedded.

Remark 3. *We will see some modeling languages for convex optimization in a few weeks, in which we describe a problem in high-level language and then the language can automatically compile or transform it to a standard form which can be solved. The point is that this makes it much easier to develop optimization-based applications or to learn and teach this material – we get to the basic idea of "say what you want, not how to get it."*

Example 4

Suppose we want to do nonnegative least squares, meaning that we want to minimize $||Ax - b||_2^2$ with $x_1, \cdots, x_n \ge 0$. The `cvxpy` code is pretty short for doing this: we initialize A, b, define x and the objective, write down the constraint, and then we just run `.solve` on the minimization problem. And various operations have been overloaded so things can be extremely human-readable.

For a brief history of this subject, the theory of convex analysis was developed mostly from 1900 to 1970 (though some of the inequalities came before that). The simplex algorithm for linear programming came around 1947 (due to Dantzig), and then early interior-point methods were developed in the 1960s (Fiacco and McCormick, Dikin, and so on). Note that the 1940s is just around when computers started their rise, so this was the first time where non-analytical solutions could really shine. The ellipsoid method and other subgradient methods (which we won't talk about in this course) were developed in the 1970s, and then interior-point methods became more popular in the 1980s and 1990s. Since then, many other methods have been developed for large-scale convex optimization. Problems like linear programming could immediately be used in areas like operations research, aerospace engineering, and finance before 1990; since then it's had applications in engineering (control, signal processing, communications, circuit design) and then machine learning, statistics, and further in finance.

We'll now begin the math, and again, the point is that everything we do will come up later.

Definition 5

Given two points $x_1, x_2 \in \mathbb{R}^n$, the **line** through x_1, x_2 is the set of points $\{\theta x_1 + (1-\theta)x_2 : \theta \in \mathbb{R}\}$.

(So as θ ranges from 0 to 1, we interpolate from x_2 to x_1, but θ can also be larger than 1 or smaller than 0.)

Definition 6

An **affine set** is a set S which contains the line through any two distinct points in S.

In particular, the solution set to any system of linear equations $Ax = b$ is always affine, and conversely, every affine set can be expressed as some solution set of linear equations (this is a good exercise to check). So we know everything we need to know from linear algebra.

Now we'll only think about the points that are actually a mixture of the two points on the boundary:

Definition 7

Given two points $x_1, x_2 \in \mathbb{R}^n$, the **line segment** through x_1, x_2 is the set of points $\{\theta x_1 + (1-\theta)x_2 : \theta \in [0, 1]\}$. A **convex set** is a set C which contains the line segment through any two points in the set, meaning that

$$x_1, x_2 \in C, \quad 0 \le \theta \le 1 \quad \implies \quad \theta x_1 + (1-\theta)x_2 \in C.$$

(Keep in mind that we may visualize geometric examples of things in two and three dimensions, but we'll be solving problems in many dimensions and we don't need to be able to visualize the feasible set for those cases.)

Definition 8

The **convex combination** of a set of points $x_1, \cdots, x_k$ is any point of the form $x = \theta_1 x_1 + \cdots + \theta_k x_k$, where $\theta_i \ge 0$ and $\theta_1 + \cdots + \theta_k = 1$. The **convex hull** of a set S, denoted $\operatorname{conv} S$, is the set of all convex combinations of points in S.

Note that S may be finite but then $\operatorname{conv} S$ is infinite, and also a point in the convex hull may not have a unique representation as a convex combination.

Definition 9

A **conic combination** of x_1, x_2 is any point of the form $x = \theta_1 x_1 + \theta_2 x_2$, where $\theta_1, \theta_2 \ge 0$. A **convex cone** is a set where conic combinations of points in the set is also in the set (in other words, convex cones are closed under nonnegative scaling and addition).

(The picture to keep in mind here is that if we have two vectors $x_1, x_2 \in \mathbb{R}^2$, then the convex cone is the whole sector bounded by x_1, x_2.)

Definition 10

A **hyperplane** is a set of the form $\{x : a^T x = b\}$ for some vectors $a \ne 0, b$. Similarly, a **halfspace** is a set of the form $\{x : a^T x \le b\}$.

In both cases, a is the normal vector to the plane or space. In the language above, hyperplanes are affine and convex and halfspaces are convex. We'll see how to make use of these notions next time!

2 January 8, 2026

(As a reminder, all of the official slides for the course are found on the course website in various places, with much more detail than we'll cover in class.)

Last time, we started introducing some of the main objects needed for convex analysis. We were discussing hyperplanes and hyperspaces; for example, the set of points where $a^T x = 0$ is the orthogonal plane to a given vector, so $a^T x = b$ specifies some parallel plane to that, and $a^T x \leq b$ specifies one of the two sides of that plane. Specifically a will be the "outward-pointing normal."

Definition 11

The **Euclidean ball** with center x_c and radius r is defined in terms of the Euclidean or ℓ^2 distance. We can write it either in a constraint form or a parameter form:

$$B(x_c, r) = \{x : ||x - x_c||_2 \leq r\} = \{x_c + ru : ||u||_2 \leq 1\}.$$

If we replace the inequality with an equality, we are instead defining the (Euclidean) **sphere**.

Definition 12

An **ellipsoid** is a set of points of the form

$$\{x : (x - x_c)^T P^{-1} (x - x_c) \leq 1\}$$

for some symmetric positive-definite matrix P (for this, we will use the notation $P \in S^n_{++}$, where the double plus sign indicates strictly positive definiteness). Equivalently, we can write this in parameter form as $\{x_c + Au : ||u||_2 \leq 1\}$ for any square, nonsingular matrix A.

Notice that if P is the identity matrix, we just get the ordinary Euclidean ball. Next, we'll generalize the notion of a Euclidean norm:

Definition 13

A **norm** is a function $|| \cdot ||$ (taking in a vector and outputting a number) satisfying the following properties:

- We have nonnegativity, meaning $||x|| \geq 0$ with equality only if and only if $x = 0$,
- We have homogeneity of degree 1, meaning that $||tx|| = |t|\,||x||$ for any $t \in \mathbb{R}$.
- We have the triangle inequality, meaning that $||x + y|| \leq ||x|| + ||y||$. (Note that the addition is between vectors on the left and scalars on the right.)

The idea is that norms are like an abstraction of an absolute value; if we put a symbol underneath, like $|| \cdot ||_2$, then we're referring to a particular norm. (For example, we have things like the "jerk norm" in contexts like mechanical engineering, since people are actually sensitive to that in practice rather than velocity or acceleration.)

Definition 14

The **norm ball** with center x_c and radius r is the set $\{x : ||x - x_c|| \leq r\}$, and the **norm cone** is the set $\{(x, t) : ||x|| \leq t\}$. The latter should be thought of as representing the norm as a graph (like how we would plot a function).

Norm balls and norm cones are always convex; in the Euclidean case the Euclidean norm cone

$$\{(x, t) : ||x||_2 \leq t\} \subset \mathbb{R}^{n+1}$$

just looks like an actual cone and we typically call it the **second-order cone**.

Definition 15

A **polyhedron** is a solution set of finitely many linear inequalities and equalities, which we can write as

$$\{x : Ax \preceq b, Cx = d\}$$

for matrices $A \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{p\times n}$ (here $\preceq$ is componentwise inequality).

Each row of the constraint $Ax \preceq b$ give us a halfspace specification, and each row of the constraint $Cx = d$ gives us a hyperplane specification (for example, maybe we have a safe workspace or a flight envelope in which we're allowed to actually operate).

Definition 16

Elaborating more on the previous notation, let S^n denote the set of all symmetric $n \times n$ matrices (this is an $\frac{n(n+1)}{2}$-dimensional vector space). Then $S^n_+ = \{X \in S^n : X \succeq 0\}$ is the set of positive semidefinite symmetric $n \times n$ matrices, which indeed forms a convex cone which we call the **positive semidefinite cone**. Similarly, S^n_{++} denotes the set of positive definite symmetric $n \times n$ matrices.

We can think of positive semidefinite matrices as the set of possible covariance matrices, and convex combinations correspond to mixtures of distributions. Or in a geometric perspective, positive definite matrices define ellipsoids. If we want a visualization, we can try plotting the set of points (x, y, z) such that $\begin{bmatrix} x & y \\ y & z \end{bmatrix}$ is positive semidefinite, and that will look like the second-order cone, but once we go to 3×3 matrices the visualization becomes much harder.

We'll now turn to **operations that preserve convexity**. There will turn out to not be that many, and in practice what's really most useful is constructive methods (rather than checking the Hessian of something and so on). So it's kind of like calculus in that we memorize a short list of base things to work with and then use composition or product or other rules.

Fact 17

In general, if we want to establish convexity of some given set C, we could apply the definition but this is really only recommended for very simple sets. Otherwise, we could use convex functions (which we'll see next lecture), or show that C is obtained from simple basic convex sets (hyperplanes, halfspaces, norm balls, etc.) by operations that preserve convexity. We'll now talk about what those operations are (intersection, affine mapping, perspective mapping, linear-fractional mapping).

Proposition 18

The intersection of any number of convex sets is convex (we can write this out algebraically to convince ourselves).

For example, consider the set $S = \{x \in \mathbb{R}^m : |p(t)| \le 1 \text{ for } |t| \le \frac{\pi}{3}\}$ for some trigonometric polynomial $p(t) = x_1 \cos t + \cdots + x_m \cos(mt)$. So for every vector in $\mathbb{R}^m$ we get some polynomial, and our goal is to see whether the polynomial is uniformly bounded by 1. We can't really work this out analytically in general, but we can just write the set as an infinite intersection

$$S = \bigcap_{|t|\le\frac{\pi}{3}} \{x : |p(t)| \le 1\},$$

since for any fixed t the polynomial p is linear in x. Since each of those sets is a slab (it's where a single linear function is between two values), which is convex because it's the intersection of halfspaces, the whole thing is convex.

Visually, we can think of a smoothly varying collection of slabs which constrain our set, even though we can't even describe the set itself explicitly.

Proposition 19

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be an affine map, meaning it is of the form $f(x) = Ax + b$. Then the image and inverse image of convex sets are convex. That is,

$$S \subseteq \mathbb{R}^n \text{ convex} \implies f(S) = \{f(x) : x \in S\} \text{ convex},$$

$$C \subseteq \mathbb{R}^m \text{ convex} \implies f^{-1}(C) = \{x \in \mathbb{R}^n : f(x) \in C\} \text{ convex}.$$

Note that f^{-1} is not quite the inverse function here, because there could be multiple points in $\mathbb{R}^n$ that map to the same output. But if f were invertible, then the two notations f^{-1} would be consistent.

For example, this implies that scalings and translations of convex sets $aS + b = \{ax + b : x \in S\}$ are also convex (for any real numbers a, b), and so is projection onto some subset of coordinates. Similarly, the solution set of a linear matrix inequality

$$\{x : x_1 A_1 + \cdots + x_m A_m \preceq B\}$$

is convex, since it is the inverse image of the positive semidefinite cone under the affine mapping $B - x_1 A_1 - \cdots - x_m A_m$.

Definition 20

The **perspective function** is a map $P : \mathbb{R}^{n+1} \to \mathbb{R}^n$ defined by

$$P(x, t) = \frac{x}{t},$$

and the domain of $\operatorname{dom} P$ is the set $\{(x, t) : t > 0\}$.

It turns out that **images and inverse images of convex sets under the perspective function are convex**. And composing with affine functions yields the following:

Definition 21

A **linear fractional function** is a function of the form

$$f(x) = \frac{Ax + b}{c^T x + d}.$$

Then again **images and inverse images of convex sets under such functions will always be convex**. For example, consider the function $f(x) = \frac{1}{x_1 + x_2 + 1} x$. This fits into the framework above, so applying f or f^{-1} preserves convexity. This function is actually pretty close to modeling how we see in real life: if we're flying a drone and looking down at the ground, then certain points closer to us will appear bigger, but **convex shapes will still look convex no matter where we're looking from**.

Definition 22

A convex cone $K \subseteq \mathbb{R}^n$ is a **proper cone** if it is **closed** (contains its limit points), **solid** (has nonempty interior), and **pointed** (contains no line).

The idea is that we're excluding degenerate cases – we don't want to think of all of $\mathbb{R}^3$ as really a cone, and we also don't want to think of a single ray as one. Proper cones in $\mathbb{R}^2$ are "pizza slices." (As a quick note, it's useful for us all to take analysis as a "deadly martial art" to protect ourselves intellectually so that we're comfortable with these kinds of notions, even if we don't really want to need it. And in practice it really doesn't matter if we think about things intuitively.)

Examples of proper cones include the nonnegative orthant

$$\mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x_i \geq 0 \text{ for } 1 \leq i \leq n\},$$

or the positive semidefinite cone S_n^+ mentioned before. (And in $\mathbb{R}$, the only proper cones are $[0, \infty)$ and $(-\infty, 0]$.)

Definition 23

For each proper cone K, we can define the generalized inequalities

$$x \preceq_K y \iff y - x \in K, \qquad x \prec_K y \iff y - x \in \text{int}\, K.$$

For example, componentwise inequality $x \preceq_{\mathbb{R}^+_n} y$ is just inequality with respect to the nonnegative orthant (this is just the default inequality between two vectors). And similarly, we write that $X \preceq_{S^n_+} Y$ for two symmetric matrices if $Y - X$ is positive semidefinite. For those two examples we drop the subscript because they're so often used.

Many properties of this generalized inequality are the same as how $\leq$ works on $\mathbb{R}$; for example, it's preserved under addition. But not all of them hold; for example, it's not always true that either $x \preceq_K y$ or $y \preceq_K x$ (for example take $x = (1, 0)$ and $y = (0, 1)$); think of this as saying that not all vectors and covariance matrices are directly comparable unless we choose a particular direction.

Proposition 24 (Separating hyperplane theorem)

Let C, D be nonempty disjoint convex sets. Then there is some hyperplane that separates them, meaning that there are some $a \neq 0, b$ such that

$$a^T x \leq b \text{ for } x \in C, \quad a^T x \geq b \text{ for } x \in D.$$

A lot of stuff in convex analysis will come down to this property. Note that this separation is not strict, and more conditions are required to get strict separation.

Proposition 25 (Supporting hyperplane theorem)

Let x_0 be a boundary point of a set $C \subseteq \mathbb{R}^n$. A **supporting hyperplane** to C at x_0 is of the form $\{x : a^T x = a^T x_0\}$ for $a \neq 0$, such that $a^T x \leq a^T x_0$ for all $x \in C$ (in other words, the entire set is on one side of a hyperplane touching x_0). If C is convex, then there is a supporting hyperplane at every boundary point.

Definition 26

A function $f : \mathbb{R}^n \to \mathbb{R}$ is **convex** if its domain $\text{dom}\, f$ is convex, and we have

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

for any $0 \leq \theta \leq 1$ and $x, y \in \text{dom}\, f$. We say that f is **strictly convex** if we have strict inequality when $x \neq y$ and $0 < \theta < 1$, and similarly that f is (strictly) **concave** if $-f$ is (strictly) convex.

The idea is that the line segment connecting the points $(x, f(x))$ and $(y, f(y))$ lies above the function. This concept already has some nice consequences: if two different values x, y give the same value of a convex function f, then any mixture of will be at least as good.

There are many examples of convex functions on $\mathbb{R}$, like affine ones $ax+b$, exponentials e^{ax}, powers x^α on $\mathbb{R}_{++}$ for $\alpha \geq 1$ or $\alpha \leq 0$, powers of absolute values $|x|^p$ on $\mathbb{R}$ for $p \geq 1$, and the "relu function" or positive part $\max(0, x)$. And there are also many useful concave functions, such as affine ones (these are the only functions that are both concave and convex), powers x^α for $0 \leq \alpha \leq 1$, logarithms, entropy $-x \log x$, and the negative part $\min(0, x)$. Moving to $\mathbb{R}^n$, again affine functions are convex, but also any norm (such as the ℓ^p or ℓ^∞ norm), or the sum of squares, or the maximum of the coordinates, or the softmax

$$\log(\exp(x_1) + \cdots + \exp(x_n))$$

are all convex functions on $\mathbb{R}^n$ as well. And notice that functions don't need to be differentiable to be convex (since for example the absolute value and max functions have V shapes in them).

Moving now to problems where the variables are actually matrices $X \in \mathbb{R}^{m \times n}$, a general affine function now takes the form

$$f(x) = \text{tr}(A^T X) + b = \sum_{i=1}^m \sum_{j=1}^n A_{ij} X_{ij} + b$$

for $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}$. This is again convex and also concave. We also have the spectral norm (maximum singular value)

$$f(X) = ||X||_2 = \sigma_{\max}(X) = (\lambda_{\max}(X^T X))^{1/2},$$

which is convex, and the log-determinant

$$f(X) = \log \det X$$

for positive definite matrices, which turns out to be concave (so this is related to the information of a Gaussian random variable). And this will all make sense and be more intuitive as we work with more examples.

Definition 27

Suppose f is convex on $\mathbb{R}^n$ with some domain $\text{dom} f$. Then we can extend it to all of $\mathbb{R}^n$ by allowing it to return ∞ if $x \notin \text{dom} f$, and we call this the extended-value version $\tilde{f}$.

This extension is a useful notion because it often simplifies notation: for example, the condition that

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

(as an inequality in the extended reals) actually encodes both the ordinary convexity condition, **plus** the fact that $\text{dom} f$ is convex.

Proposition 28

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if it's always convex restricted to any line, meaning that the function $g : \mathbb{R} \to \mathbb{R}$ defined by $g(t) = f(x + tv)$ is convex in t for any $x \in \text{dom} f$ and $v \in \mathbb{R}^n$.

Thus we can check convexity by just checking convexity in a single variable, which checks out with how the notion is defined. (And so in a high-dimensional problem you could pick millions of random lines and check whether the convexity condition fails; any counterexample would prove that it isn't convex.)

Example 29

Suppose we define the function $f(X) = \log\det X$ on symmetric matrices with domain the positive definite matrices. We previously said this should be concave.

Indeed, consider a line $X + tV$ in matrix space (where X must be positive definite because it's in the domain, but V can be any symmetric matrix); we have

$$\begin{aligned} g(t) &= \log\det(X + tV) \\ &= \log\det(X^{1/2}(I + tX^{-1/2}VX^{-1/2})X^{1/2})) \\ &= \log\det X + \log\det(I + tX^{-1/2}VX^{-1/2}) \\ &= \log\det X + \sum_{i=1}^{n} \log(1 + t\lambda_i) \end{aligned}$$

for λ_i the eigenvalues of $X^{-1/2}VX^{-1/2}$. But then $\log(1 + ct)$ is always concave, so this expression for g is concave in t; thus f is concave as well.

Remark 30. *On the other hand, if we defined this same function on all matrices rather than just the symmetric ones, it is no longer concave (we can try finding a counterexample). But in most real applications of the determinant (e.g. covariance matrices), everything is indeed symmetric anyway.*

3 January 13, 2026

As a reminder, our first problem session is this afternoon, and now is around when we should start looking at the problem set. Remember that our homework is still "just math," and starting next week we'll transition more into what this class is actually about.

Last time, we showed one way to establish that a function is convex or concave, which is to restrict to arbitrary lines and plot the function. If the result is always convex, then the function is convex, and we saw this last time with the log determinant of a positive definite matrix (which is quite a complicated function). We'll explore some other conditions now:

Definition 31

A function f is **differentiable** if the domain $\mathrm{dom}\, f$ is open and the gradient $\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \cdots, \frac{\partial f(x)}{\partial x_n}\right)$ exists at each $x \in \mathrm{dom}\, f$.

Proposition 32

The **first-order condition** says that a differentiable f with convex domain is convex if and only if for all $x, y \in \mathrm{dom}\, f$, the affine approximation is always a global underestimator, meaning that

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

In other words, convex functions are always above their first-order Taylor approximation. What's nice is that even in complicated problems, convexity means we get global information just from first-order properties, rather than just local information like we typically do with derivatives.

Definition 33

A function f is **twice differentiable** if its domain is open and the Hessian $\nabla^2 f(x)$ exists at every point in the domain.

We should really think about this in the same way that we think about sign analysis in ordinary calculus.

Proposition 34

For a twice differentiable f with convex domain, a function f is convex if and only if $\nabla^2 f(x) \succeq 0$ for all $x \in \operatorname{dom} f$, and it is strictly convex if and only if $\nabla^2 f(x) \succ 0$.

Example 35

The gradient of a **quadratic function** $f(x) = \frac{1}{2}x^T Px + q^T x + r$ (for P symmetric) is $\nabla f(x) = Px + q$, and so $\nabla^2 f(x) = P$. So if P has only nonnegative eigenvalues, it's convex, and if it has nonpositive eigenvalues, it's concave.

Example 36

The **least-squares objective** $f(x) = ||Ax - b||_2^2$ satisfies $\nabla f(x) = 2A^T(Ax - b)$ and $\nabla^2 f(x) = 2A^T A$, so in fact

Example 37

The **quadratic-over-linear** function $f(x, y) = \frac{x^2}{y}$ for $y > 0$ is indeed convex, since

$$\nabla^2 f(x, y) = \frac{2}{y^3}\begin{bmatrix} y \\ -x \end{bmatrix}\begin{bmatrix} y \\ -x \end{bmatrix}^T \succeq 0.$$

This is a rank-1 matrix, so there is one direction in which it has zero curvature.

Also keep in mind that a multivariate jointly convex function must always be convex in each of its variables, and indeed both x^2 and $\frac{1}{y}$ are convex. On the other hand, we cannot just check convexity per variable, since that would be like only checking whether the diagonal entries of the Hessian are positive.

Example 38

The **logsumexp** function $f(x) = \log\left(\sum_{k=1}^n \exp(x_k)\right)$ is convex (this is for example what we use to get the decibel reading of a combination of sounds). Indeed, we can explicitly check that if $z_k = \exp(x_k)$, then

$$\nabla^2 f(x) = \frac{1}{\mathbf{1}^T z}\operatorname{diag}(z) - \frac{1}{(\mathbf{1}^T z)^2} zz^T.$$

To show that this is positive semidefinite, we just need to check that $v^T \nabla^2 f(x) v \geq 0$ for all v. Indeed,

$$v^T \nabla^2 f(x) v = \frac{(\sum_k z_k v_k^2)(\sum_k z_k) - \left(\sum_k v_k z_k\right)^2}{(\sum_k z_k)^2}$$

and the numerator is nonnegative by Cauchy-Schwarz.

A similar proof also shows that the **geometric mean** $f(x) = \left(\prod_{k=1}^n x_k\right)^{1/n}$ is concave on $\mathbb{R}^n_{++}$.

So now convexity as an attribute applies to both convex sets and convex functions, and we now want to understand the connection. This actually manifests through the following object which is related to the graph of a function:

Definition 39

The **epigraph** of a function $f : \mathbb{R}^n \to \mathbb{R}$ is the set

$$\operatorname{epi} f = \{(x, t) \in \mathbb{R}^{n+1} : x \in \operatorname{dom} f, f(x) \leq t\}.$$

(Here "epi' means "above," so this is everything above the graph.)

Proposition 40

A function f is a convex function if and only if $\operatorname{epi} f$ is a convex set.

Recall that a convex function is defined by saying that the line segment between two points on a graph is always above the function. This can be generalized to arbitrary mixtures of vectors:

Proposition 41 (Jensen's inequality)

Suppose f is a convex function and Z is a random variable on $\operatorname{dom} f$. Then $f(\mathbb{E}[Z]) \leq \mathbb{E}[f(Z)]$.

Indeed, the "basic inequality" for convexity is a special case with the discrete distribution $\mathbb{P}(Z = x) = \theta$, $\mathbb{P}(Z = y) = 1 - \theta$.

Example 42

Let $X \sim N(\mu, \sigma^2)$ be Gaussian. Then $Y = e^X$ is what is called a **lognormal** random variable. Defining $f(x) = e^x$, we have $\mathbb{E}[f(X)] = e^{\mu+\sigma^2/2}$, and Jensen's inequality says that

$$e^\mu = f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)] = e^{\mu+\sigma^2/2}.$$

Remark 43. *Note that in basically all modern applications, differentiability doesn't actually matter for convex optimization – it's useful for thinking about things in the background or quickly showing certain things, but for about 30 years or so most of the problems which we work with aren't actually differentiable.*

So with this, we can turn back to understanding "how to show a function is convex," since that's what's necessary in a lot of applications. Much like we discussed last lecture with convex sets, there's a few main strategies: we can verify the definition by restricting to a line segment and compute (though this is rarely the right strategy), or we can explicitly show $\nabla^2 f(x) \succeq 0$ if f is twice differentiable (though this is only recommended if the function is very simple – trying to calculate the Hessian of $\log\det X$ would be pretty annoying). But more generally, what's best is to **show that we can obtain it from simple convex functions by operations that preserve convexity** (nonnegative weighted sum, composition with affine functions, pointwise max / supremum, composition, minimization, perspective), which we'll now talk about.

Proposition 44

Some of these operations are very easy to check. For example, a nonnegative multiple of a convex function is convex. Similarly, the sum of convex functions is convex, and so are arbitrary infinite sums. Therefore, if $f(x, \alpha)$ is convex in x for each $\alpha \in \mathcal{A}$, then any arbitrary combination (integral)

$$\int_{\alpha \in \mathcal{A}} f(x, \alpha) d\alpha$$

is also convex. (And there are analogous rules for concavity as well.)

Proposition 45

The pre-composition with an affine function is convex. In other words, if f is convex, then $f(Ax + b)$ is convex as well. (Indeed, for the differentiable case, $A^T \nabla^2 f A$ is positive definite if $\nabla^2 f$ is.)

An example is the log barrier for linear inequalities

$$f(x) = -\sum_i \log(b_i - a_i^T x)$$

(where the domain is the set of x where $a_i^T x < b_i$ for all i) or the norm approximation error $||Ax - b||$ for any norm.

Proposition 46

The pointwise maximum of convex functions $f(x) = \max(f_1(x), \cdots, f_m(x)\}$ is convex.

One way to think about this is that the intersection of the epigraphs is exactly the epigraph of the pointwise maximum, and the intersection of convex sets is convex. Therefore, piecewise-linear functions which are maximums of affine functions, as well as the **sum of the r largest components** of a vector $x \in \mathbb{R}^n$, are convex – the latter of these is exactly the maximum over all $\binom{n}{r}$ possible choices of indices. (Note that both of these example functions are not differentiable everywhere.)

As a sidenote, if we take the sum of the r largest absolute values of the coordinates, we get some interesting objects: if $r = 1$ we get the ℓ_∞ norm, and if $r = n$ we get the ℓ_1 norm.

Proposition 47

The concept above extends to suprema over arbitrary sets: if $f(x, y)$ is convex for all $y \in \mathcal{A}$, then $g(x) = \sup_{y \in \mathcal{A}} f(x, y)$ is convex. For example, the **distance to the farthest point** in a set $f(x) = \sup_{y \in C} ||x - y||$ is convex and so are the **maximum eigenvalue** of a symmetric matrix $\lambda_{\max}(X) = \sup_{||y||_2 = 1} y^T X y$ and the **support function** of a set $S_C(x) = \sup_{y \in C} y^T x$.

The point is that there aren't really analytical formulas for a lot of these functions, but each individual function in the supremum is quite simple (for example, $y^T X y$ is quadratic in y and linear in X, so checking convexity in either of those variables is pretty easy if we needed that).

Proposition 48

The function $g(x) = \inf_{y \in C} f(x, y)$ is the **partial minimization** of f with respect to y. It turns out that partial minimization also preserves convexity **if C is convex**.

For example, take a quadratic form $f(x, y) = x^T Ax + 2x^T By + y^T Cy$ where $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succeq 0$ and $C \succ 0$. We find that $g(x) = \inf_y f(x, y) = x^T(A - BC^{-1}B^T)x$ by an explicit computation, and we get that g is convex. So what this tells us is that the **Schur complement** $A - BC^{-1}B^T$ is positive semidefinite (this is what we get when we compute the impedance matrix when we short some connections in a circuit).

Proposition 49

The **composition** of two functions $g : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R} \to \mathbb{R}$ is $f(x) = h(g(x))$. Then informally, **a convex increasing function of a convex function is convex**. More precisely, if h is convex and nondecreasing for the extended-value extension $\tilde{h}$, and g is convex, then f is convex. Similarly, if h is convex and nonincreasing for the extended-value extension $\tilde{h}$, and g is concave, then f is again convex. One proof for the differentiable one-variable case is that

$$f''(x) = h''(g(x))g'(x)^2 + h'(g(x))g''(x),$$

so if h is convex the first term is nonnegative, and if h is increasing and g is convex (or h is decreasing and g is concave) the second term is nonnegative as well.

So for example $e^{g(x)}$ is convex if g is convex, and $\frac{1}{g(x)}$ is convex if g is positive and concave.

Theorem 50 (General composition rule)

More generally, suppose $g : \mathbb{R}^n \to \mathbb{R}^k$ and $h : \mathbb{R}^k \to \mathbb{R}$ are two functions, so we have $f(x) = h(g(x)) = h(g_1(x), \cdots, g_k(x))$. Then f is convex if h is convex and for each i, one of the following holds:

- g_i is convex and $\tilde{h}$ is nondecreasing in its ith argument,
- g_i is concave and $\tilde{h}$ is nonincreasing in its ith argument,
- g_i is affine.

This rule will be constantly used throughout the course and thus we should commit it to memory. It's really just "arguing about sign analysis of first and second derivatives," but it works for things that are completely nondifferentiable.

Example 51

If the functions g_i are convex, then $\log\left(\sum_{i=1}^m \exp(g_i(x))\right)$ is convex. The wrong way to try proving this is the following: $\exp(g_i(x))$ is convex, so the sum of those functions is convex, but then things fail because log is concave. Instead, logsumexp is convex and motonone increasing in all of its arguments, so we can plug in convex functions in and the result will be convex.

Similarly, $f(x) = \frac{p(x)^2}{q(x)}$ is convex provided that p is nonnegative and convex, and q is positive and concave, using the base function $\frac{x^2}{y}$. Indeed, $f(u, v) = \frac{u^2}{v}$ is increasing in the first argument and decreasing in the second argument if we specify u to be nonnegative and v to be positive.

Notice that the composition rule subsumes lots of other rules (such as saying that the weighted sum or max of convex functions is convex). So really we can derive all of the other rules from this one, and that's a good exercise.

Fact 52

We'll now talk about how to do **constructive convexity verification**: we start with a function written as an expression, and we can parse it into a parse tree. The leaves of our tree are variables or constants, and nodes will be various functions of its child expressions. We can then use the composition rule to tag each subexpression in the tree (from the leaves up to the root) as either convex, concave, affine, or none. If the root is labeled convex or concave, then f is convex or concave, respectively.

This is something we can do explicitly, since for each node function we can ask for its curvature and monotonicity with respect to its arguments and then run through the composition rule cases to see if we can deduce convexity or concavity of those subexpressions. And furthermore, we can also tag the sign of each expression and use sign-dependent monotonicity (for example, that would let us deduce whether x^2 is increasing or decreasing). So really all we need is whether the 0th, 1st, and 2nd derivatives of our functions are positive or negative. This can be readily automated and is a sufficient way to prove convexity or concavity, but of course it may not work if we don't choose the right parse tree.

Example 53

Consider the function $f(x) = \frac{(x-y)^2}{1-\max(x,y)}$ on the domain $x < 1, y < 1$. We will construct a parse tree to show that this is convex. Our leaves are $x, y, 1$, which are all affine. Then $x - y$ is affine and $\max(x, y)$ is convex. Therefore the denominator $1 - \max(x, y)$ is concave. Now $\frac{u^2}{v}$ is convex and monotone decreasing in v for $v > 0$, so because f is the composition of that function with $u = x - y$ (affine) and $v = 1 - \max(x, y)$ (concave), it is convex.

We can see this being explicitly written out in a parse tree at https://dcp.stanford.edu/analyzer by typing in the function as `quad_over_lin(x-y, 1-max(x, y))`. Each node is indeed labeled with its sign and curvature. (And note that the `quad_over_lin` function's domain is only where the second argument is positive, which is why we know that the final result must be positive even if we didn't explicitly specify $x < 1, y < 1$.)

We can do all of this in `cvxpy` as well, and the code isn't too complicated: we initialize variables x, y and form the expression that we're looking for, and we can query `expr.curvature` or `expr.sign` and it will automatically return `Convex` and `Positive`. And this is something we'll become fairly comfortable with very soon!

4 January 13, 2026 (Problem Session)

As a reminder, these sessions are being taught by Professor Babak Ayazifar. The idea is that we'll go through some problems on the problem set and pull some concepts from the past week, and hopefully it'll be helpful for tackling the problems. (Videos will be posted afterward as well.) We'll be given time to ask questions and work with those around us too.

Problem 54

Suppose we're given two points $a, b \in \mathbb{R}^n$ and are curious about the set of points $\{x : ||x - a|| \le ||x - b||\}$ closer in (Euclidean) distance to a than b. We want to prove that this is a halfspace – whenever we see absolute values and norms, things are usually curved, but here we actually have something linear going on.

In two dimensions, we can draw the perpendicular bisector between a and b, and anything on the side closer to a (including the bisector itself) is in our set. But we want to show that this is the case algebraically, meaning that we want to rewrite the set in the form $c^T x \le d$.

To do this, we start with $||x-a|| \le ||x-b||$ and square both sides to get rid of the norm (which is valid since both sides start off nonnegative). This is done whenever we study least-squares, and it yields

$$(x-a)^T(x-a) \le (x-b)^T(x-b)$$
$$\implies x^Tx - x^Ta - a^Tx - a^Ta \le x^Tx - x^Tb - b^Tx + b^Tb.$$

This is nice because the quadratics in x drop out and we're left with a linear inequality; noting that $x^Ta = a^Tx$, we get

$$-2a^Tx + a^Ta = -2b^Tx + b^Tb,$$

and now this looks basically like the halfspace inequality we want: rearranging and collecting terms yields

$$2(b^T - a^T)x \le b^Tb - a^Ta,$$

so we can let $c = 2(b-a)$ and $d = b^Tb - a^Ta$. (Indeed, we see that c is pointing from a to b, so it is the halfspace of our perpendicular bisector in the 2D case but it works in any dimension.)

Problem 55

Next, we'll consider some sets and see whether or not they are convex:

$$S_1 = \left\{(x,y) \in \mathbb{R}^2_{++} : \frac{x}{y} \le 1\right\}, \quad S_2 = \left\{(x,y) \in \mathbb{R}^2_+ : xy \le 1\right\}, \quad S_3 = \left\{(x,y) \in \mathbb{R}^2_+ : xy \ge 1\right\}$$

(The reason for requiring positivity in S_1 but not S_2, S_3 is so we don't divide by zero.)

The first set S_1 is exactly "half of the first quadrant," since the inequality says that $y \ge x$ and so we just have to be above the line $y = x$ (including that line but not including the y-axis). One way to justify that this set is convex is that it is the epigraph of a convex function, and another is that it's the intersection of various halfspaces ($y \ge x$ and $x > 0$), or that it's the intersection of the halfspace $y \ge x$ and the convex quadrant $\mathbb{R}^2_{++}$.

In the second set S_2, we're looking at the set of points below the graph of $y = \frac{1}{x}$ (and also including the axes in this case). But since this "curves inward," it's not going to be convex – the line segment between $(2, \frac{1}{2})$ and $\left(\frac{1}{2}, 2\right)$ goes outside the region. On the other hand, S_3 is the points above the curve, which is indeed convex because it's the epigraph of $\frac{1}{x}$, which is convex.

Problem 56

For some more examples, now we'll try to determine the convexity of the sets

$$S_1 = \{x \in \mathbb{R}^n : \alpha \le a^Tx \le \beta\}, \quad S_2 = \{x \in \mathbb{R}^n : a_1^Tx \le b_1, a_2^Tx \le b2_\},$$

$$S_3 = \{x : x \text{ is closer to } x_0 \text{ than some set } S\}, \quad S_4 = \{x : x \text{ is closer to one set } S \text{ than another set } S'\}.$$

S_1 is indeed convex, because it is the intersection of two parallel halfspaces with normal vector a ($\alpha \le a^Tx$ can be rewritten as $(-a)^Tx \le -\alpha$ so that it's in the same form). This is called a **slab**. Note that even if $\beta < \alpha$, the set would still be convex, because it would just be empty and thus vacuously satisfy the necessary condition.

S_2 is convex for exactly the same reason except that it is now generally the intersection of two not-necessarily-parallel halfspaces. The result will now look like a wedge when a_1, a_2 point in different directions (and the total angle of the wedge is supplementary to the angle between a_1 and a_2).

Remark 57. *Wedges are also convex cones when they're centered at the origin (because they contain arbitrary conic combinations $\theta_1 x_1 + \theta_2 x_2$ of points with θ_1, θ_2 nonnegative), but not otherwise.*

S_3 is a bit more abstract (we don't even know what the set S is), but we can rewrite it as

$$\{x : ||x - x_0|| < ||x - y|| \text{ for all } y \in S\},$$

which is the intersection of half-spaces indexed by $y \in S$. So this is an infinite intersection, and in particular it's quite hard to visualize what the set might be. But since we're the (possibly uncountable) intersection of convex sets, we are again convex. And S_4 turns out to be not convex in general; we can come up with counterexamples (for example the set of points closer to a sphere than its center).

The point of these exercises is that we often want to de-clutter our problem and convert it to a convex problem; we should be able to recognize these common shapes and make use of them even in more complex settings.

Problem 58

Next, we'll consider convexity with respect to probability distributions p. We'll use the notation X for a random variable and x for a particular value that the variable takes, and we write $p_X(x) = \mathbb{P}(X = x)$ for the probability of taking on that value. Suppose X can only take on the values $a_1 < \cdots < a_n$ with some probabilities $p = (p_1, \cdots, p_n)$ which are nonnegative and sum to 1. (In our notation we write this as $p \succeq 0$ and $\mathbf{1}^T p = 1$ – this is the **probability simplex**, which is the convex hull of the canonical unit vectors.) We are curious about the sets

$$S_1 = \{p : \alpha \le \mathbb{E}_p[f(X)] \le \beta\}, \quad S_2 = \{p : \mathbb{P}(X \ge \alpha) \le \beta\},$$

$$S_3 = \{p : \text{Var}_p(X) \ge \alpha\}.$$

For S_1, we need to write out the expected value $\mathbb{E}[f(X)] = \sum_x f(x)p(x) = \sum_{i=1}^n f(a_i)p_i = f(a)^T p$. What's important is that this expression is linear in p, regardless of the values of a_i or $f(a_i)$, and so the condition is a slab, hence convex. (And to be precise, we have to also take the intersection with the conditions specifying the probability simplex, which are an orthant and a hyperplane.)

For S_2, note that $\mathbb{P}(X \ge \alpha)$ is specifying some fixed threshold (event), and so we can write it as some sum of p_is (for the indices i where $a_i \ge \alpha$). So the condition for the probability vector is of the form $p_j + \cdots + p_n \le \beta$ for some j, which will always be some halfspace condition and hence yield a convex set.

Finally, for S_3, one of the equivalent ways of representing the variance is

$$\begin{aligned}
\text{Var}_p(X) &= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \\
&= \sum_{i=1}^n a_i^2 p_i - \left(\sum_{i=1}^n a_i p_i\right)^2 \\
&= (a^2)^T p - (a^T p)^2 \\
&= (a^2)^T p - a^T p a^T p \\
&= (a^2)^T p - p^T a a^T p.
\end{aligned}$$

Now the first term is linear and the second term being subtracted off is $p^T A p$ for a positive semidefinite symmetric matrix $A \succeq 0$; thus $\text{Var}_p(X)$ is actually concave in p and thus the condition $\text{Var}_p(X) \ge \alpha$ leads to a convex set. (And also this means the set where $\text{Var}_p(X) \le \alpha$ will typically not be convex; we can indeed find a counterexample.)

Remark 59. *The key fact we're using here is the following: if $C \subseteq \mathbb{R}^n$ is the solution of a quadratic inequality $\{x \in \mathbb{R}^n : x^T A x + b^T x + c \le 0\}$ with A symmetric and $b \in \mathbb{R}^n, c \in \mathbb{R}$, and if A is positive semidefinite, then C is* *convex. The proof of this is to* **take the intersection of an arbitrary line** $\{z + tv : t \in \mathbb{R}\}$ **with** C *and prove that it must always be a line segment (convex). This is true because the eventual condition on the scalar t will be a quadratic with nonnegative leading coefficient, since $v^T Av$ is always nonnegative.*

5 January 15, 2026

Last time, we introduced constructive convexity analysis, which is an important thing that we will actually do over and over again. In the computer science sense, we get an expression as a parse tree, and we match to each node (function of its children nodes) positive/negative, increasing/decreasing, convex/concave using the composition rule. So effectively, we're doing the equivalent of crude sign analysis from calculus, but it's an automated procedure.

Fact 60

In **disciplined convex programming (DCP)**, users construct convex and concave functions as expressions coming from variables, constants, and certain **atomic functions** from a certain library (powers, norms, min, max, etc.). These atomic functions have known convexity, monotonicity, and sign properties (for example, exp is convex, increasing, and positive), and the idea is that a valid DCP is "syntactically" convex by construction (from the composition rule).

In particular, convexity then doesn't actually depend on the actual functions, only their attributes – for example, we could swap $\exp(\cdot)$ and "positive part" $(\cdot)_+$ in a function, and the convexity would remain the same.

Remark 61. *In* `cvxpy`*, we initialize variables like* `x = cp.Variable()` *(or* `x = cp.Variable(n)` *if we want a vector). What's important to remember is that* `x` *is not a number – it's a variable, and it'll have attributes like "affine." If we then define an expression like* `cp.quad_over_lin(x-y, 1-cp.maximum(x, y))`*, note that the "minus" operation is overloaded to return an expression instead of being ordinary subtraction, and the atomic function* `quad_over_lin(u, v)` *is* **defined** *to include the domain constraint $v > 0$, which implicitly forces $x < 1, y < 1$. Then* `expr.curvature` *returns* `Convex` *and* `expr.sign` *returns* `Positive` *(because* `cvxpy` *can just traverse the tree), and* `expr.is_dcp()` *would return* `True` *(meaning that we've parsed the expression and verified that it is what we said).* `is_dcp` *will also be useful when we start applying it to optimization problems.*

Example 62

It's important though that **DCP is only sufficient, not necessary** – for example, the function $f(x) + \sqrt{1+x^2}$ is convex, but `f1 = cp.sqrt(1 + cp.square(x))` is not DCP. (If we start at the leaves, we see that `1 + cp.square(x)` is convex, but the square root of a convex function doesn't tell us anything about convexity since the composition rule would only tell us about the square root of a concave function.) On the other hand, `f2 = cp.norm2([1, x])` is "exactly the same function," but it is DCP. So `cvxpy` may not recognize everything.

Definition 63

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function. THe **perspective** of f is the function $g : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ given by $g(x, t) = tf(x/t)$ (with domain the set of points where $\frac{x}{t} \in \operatorname{dom} f$ and $t > 0$).

This should look a lot like the perspective mapping, and indeed g is convex if f is (because the epigraph of the perspective function is the perspective mapping of the epigraph). For example, if $f(x)$ is the transaction cost for executing a large trade (it's linear for small trades but ends up being more if you move the market), where x is the order size, and $t = 10$ is the number of days over which we execute that order, then $g(x, t)$ is the total cost if we split it over time.

Example 64

Since $f(x) = x^T x$ is convex, the perspective $g(x, t) = \frac{x^T x}{t}$ is also convex for $t > 0$ – this is just quad-over-lin. And since $f(x) = -\log x$ is convex (on $\mathbb{R}_{++}$), its perspective $g(x, t) = t \log t - t \log x$ is convex on $\mathbb{R}^2_{++}$ – this is called the **relative entropy**.

Definition 65

The **conjugate** of a (not necessarily convex) function f is given by

$$f^*(y) = \sup_{x \in \text{dom } f} \left\{ y^T x - f(x) \right\}.$$

The intuition to keep in mind is that f is the vector cost of choosing quantities x of various things to produce or buy (this is called the "production function"). If y is the cost, then $y^T x$ is the revenue, so $y^T x - f(x)$ is the profit; thus $f^*(y)$ is the **optimal** profit over all quantities. And graphically, we can construct the conjugate in one dimension by imagining $f^*(y)$ to be the biggest distance that a line of slope y through the origin is above the function.

The most important fact is that f^* is always convex, even if f is not – this will end up being very useful when we solve problems later. Indeed, $y^T x - f(x)$ is affine in y, hence convex, and then $f^*(y)$ is the supremum of many different convex functions, which is still convex. But we'll come back to this later and it'll make more sense in context.

Example 66

If $f(x) = -\log x$, then we can compute

$$f^*(y) = \sup_{x>0} \{xy + \log x\} = \begin{cases} -1 - \log(-y), & y < 0, \\ \infty, & \text{otherwise}, \end{cases}$$

so it's basically $-\log$ back again. Also, if we define $f(x) = \frac{1}{2} x^T Q x$, it turns out $f^*(y) = \frac{1}{2} y^T Q^{-1} y$ (so we get the quadratic form of the inverse matrix).

We'll now extend convexity to a more general concept – usually there's some big diagram of relations between them and they're all not very useful, but there are two that are above the threshold of utility, which are **quasiconvexity** and **log-convexity** (the latter of which we won't discuss, though it's very useful in statistics and probability).

Definition 67

A function $f : \mathbb{R}^n \to \mathbb{R}$ is **quasiconvex** if $\text{dom } f$ is convex, and the **sublevel sets** $S_\alpha = \{x \in \text{dom } f : f(x) \le \alpha\}$ are convex for all α. We call f **quasiconcave** if $-f$ is quasiconvex and **quasilinear** if it is both quasiconvex and quasiconcave.

Another word for "quasiconvex" is "unimodal" – we can't have a function with two separated bumps.

Example 68

The function $\sqrt{|x|}$ is quasiconvex because its sublevel sets are intervals, but it's very not convex because it has a "cusp" at 0. The ceiling function $\lceil x \rceil = \inf\{z \in \mathbb{Z} : z \ge x\}$ is actually quasilinear on $\mathbb{R}$, and so is $\log x$ on $\mathbb{R}_{++}$. And for some slightly more complicated examples, $f(x_1, x_2) = x_1 x_2$ is quasiconcave on $\mathbb{R}^2_{++}$, and linear-fractional functions $f(x) = \frac{a^T x + b}{c^T x + d}$ are linear on the domain $\{c^T x + d > 0\}$.

Example 69

Suppose $x = (x_0, \cdots, x_n)$ is a company's cash flow, meaning that x_i is payment that the company receives in period x. We'll assume that $x_0 < 0$ (so for example an initial investment is made) but $x_0 + \cdots + x_n > 0$. The **net present value** of the cash flow for some interest rate r is then

$$\text{PV}(x, r) = \sum_{i=0}^{n} (1 + r)^{-i} x_i,$$

and the **internal rate of return** $\text{IRR}(x)$ is the smallest interest rate for which $\text{PV}(x, r) = 0$. (We discount future values because we can't benefit from interest for them.)

With the notation above, IRR is quasiconcave, since its superlevel set is the intersection of open halfspaces:

$$\text{IRR}(x) \ge R \iff \sum_{i=0}^{n} (1 + r)^{-i} x_I > 0 \text{ for all } 0 \le r < R.$$

This is an example of a quantity we do actually want to maximize, so it's good that it is a quasiconcave function!

Proposition 70

If f is quasiconvex, it satisfies the **modified Jensen inequality**

$$0 \le \theta \le 1 \quad \iff \quad f(\theta x + (1 - \theta) y) \le \max\{f(x), f(y)\}.$$

Also, a differentiable f with convex domain is quasiconvex if and only if

$$f(y) \le f(x) \iff \nabla f(x)^T (y - x) \le 0.$$

There are still composition rules for quasiconvexity, but they're far reduced – for example, any increasing function of a quasiconvex function is quasiconvex. And it's also true that quasiconvexity is equivalent to quasiconvexity (unimodality) on any line. On the other hand, the sum of quasiconvex functions are not necessarily quasiconvex (for example, the sum of two Gaussians is not unimodal).

We're ready to move to convex optimization **problems** now.

Definition 71

An **optimization problem in standard form** is of the form

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \le 0, \quad i = 1, \cdots, m, \\ & h_i(x) = 0, \quad i = 1, \cdots, p. \end{aligned}$$

Here $x \in \mathbb{R}^n$ is our optimization variable that we want to choose, $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the objective function that we want to be as small as possible, and $f_i : \mathbb{R}^n \to \mathbb{R}$ and $h_i : \mathbb{R}^n \to \mathbb{R}$ are the inequality and equality constraint functions which absolutely must be satisfied (but for which the margin doesn't matter).

Definition 72

In an optimization problem (with the notation above), a point $x \in \mathbb{R}^n$ is **feasible** if it is in the domain of all functions and satisfies the constraints. We let X_{opt} denote the set of all optimal points. The **optimal value** is

$$p^\star = \inf\{f_0(x) : f_i(x) \le 0 \text{ for } i = 1, \cdots, m, h_i(x) = 0 \text{ for } i = 1, \cdots, p\}.$$

(note $\star$ instead of $*$). If $p^\star = \infty$ then the problem is **infeasible**, and if $p^\star = -\infty$ the problem is **unbounded below** (which probably means that we've set up the problem wrong), and a feasible point x is **optimal** if $f_0(x) = p^\star$. A point x is **locally optimal** if it's the best point within some radius $R > 0$ (which we can think of as having an optimization problem in z but just adding an additional constraint $||z - x|| < R$).

Example 73

In the one-dimensional case with no hard constraints, if $f_0(x) = \frac{1}{x}$ with $\operatorname{dom} f_0 = \mathbb{R}_{++}$, then $p^\star = 0$ and there is no optimal point. On the other hand, for $f_0(x) = x \log x$ on $\mathbb{R}_{++}$ we have $p^\star = -\frac{1}{e}$ with the optimal point $x = \frac{1}{e}$. Finally, for $f_0(x) = x^3 - 3x$, $p^\star = -\infty$ but $x = 1$ is locally optimal.

Notice that optimization problems has both implicit constraints (being simultaneously in the domain of f_0, f_i, h_i, which we call the **domain** of the problem $\mathcal{D}$) and explicit constraints $f_i(x) \le 0, h_i(x) = 0$. We say a problem is **unconstrained** if there are no explicit constraints, but we might still have implicit constraints; for example

$$\text{minimize} \quad f_0(x) = -\sum_{i=1}^{k} \log(b_i - a_i^T x)$$

is an unconstrained problem whose solutions must implicitly lie in the intersection of k half-spaces.

Fact 74

In a **feasibility problem**, we wish to find some x subject to some constraints $f_i(x) \le 0, h_i(x) = 0$. But we can just formulate this as an optimization problem with $f_0(x) = 0$, so that if $p^\star = 0$ the constraints are feasible and any feasible x is optimal, and if $p^\star = \infty$ the constraints are infeasible.

Definition 75

In a **standard form convex optimization problem**, we wish to solve

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \le 0, \quad i = 1, \cdots, m, \\ & a_i^T x = b_i, \quad i = 1, \cdots, p, \end{aligned}$$

where $f_0, \cdots, f_m$ are convex. (We often write the affine equality constraints as $Ax = b$.) We say the problem is **quasiconvex** if f_0 is instead quasiconvex (but f_i are still convex and h_i are still affine)

By definition, the feasible and optimal sets of a convex optimization problem are always convex.

Example 76

Suppose we want to minimize $x_1^2 + x_2^2$ subject to $f_1(x) = \frac{x_1}{1+x_2^2} \le 0$ and $h_1(x) = (x_1 + x_2)^2 = 0$. Then f_1 is not convex and h_1 is not affine, so this problem is not convex. But it is **equivalent (but not identical)** to just saying that we want to minimize $x_1^2 + x_2^2$ subject to $x_1 \le 0$ and $x_1 + x_2 = 0$, which is now a convex problem.

Here equivalence means that there is some reduction from one problem to the other, but the point is that whether a problem is convex or not depends on how it's written – when we say whether two optimization problems are "equal," we literally meant that every f_i and h_i is exactly the same function.

Proposition 77

Any locally optimal point of a convex problem is globally optimal.

The cool idea here is that calculus is basically a constructive system of things that let us understand facts about a function locally, and convexity tells us that we now get to turn this into a global statement!

Proof sketch. If x is locally optimal but there is some feasible y with $f_0(y) < f_0(x)$, then we can draw a line segment between x and y and find points arbitrarily close enough to x which must be smaller than x as well. $\square$

Proposition 78

Suppose f_0 is differentiable. A point x is optimal for a convex problem if and only if it is feasible and $\nabla f_0(x)^T(y - x) \ge 0$ for all feasible y. In such a situation, if $\nabla f_0(x)$ is nonzero, then $-\nabla f_0(x)$ defines a supporting hyperplane to the feasible set at x.

Thus for an unconstrained differentiable problem, x minimizes a function f_0 if and only if $\nabla f_0(x) = 0$ (like we learned in calculus). Similarly, x minimizes $f_0(x)$ subject to $Ax = b$ if and only if there exists some ν such that $\nabla f_0(x) + A^T \nu = 0$. (This is just Lagrange multipliers, and we'll go into it in great detail.)

We'll now look at some **standard convex problems**. This material is a bit weird and dated – usually we use convex optimization with `cvxpy`, so we don't need to know any problems with special names because we just call the `solve` method (and next time we'll see exactly what it is). But historically people made up specific problems with names and those were implemented, and so it's kind of nice to know that.

Definition 79

In a **linear program (LP)**, all functions are affine, meaning that we want to minimize $c^T x + d$ subject to $Gx \preceq h$ and $Ax = b$.

Linear programs are convex problems with affine objective and constraint functions, and so the feasible set is always a polyhedron – we want to find the point in that polyhedron which is furthest along in some vector direction $-c$, since the level sets of an affine function are hyperplanes. (This was actually written about by Fourier, and the modern advent of this really started with computers because usually there aren't closed-form analytic solutions.)

So our job is to stay in the feasible set and find the best point, and it's going to always be one of the vertices. The problem is that if we're in $\mathbb{R}^{1000}$ and we have 3000 constraints, we get an enormously exponential number of vertices, and so we can't just check those vertices one by one. But it actually turns out we can solve such an LP on our phone in about 15 milliseconds with the right algorithm, and in general if we can reduce a problem to a linear program then we've basically solved it.

Example 80

For historical significance, we can think about the "diet problem." Suppose we want to choose some nonnegative quantities of foods $(x_1, \cdots, x_n)$, and each unit of food costs c_j and has some nutrients A_{ij}. A healthy diet requires at least b_i of nutrient i. Thus, to find the cheapest healthy diet, we want to minimize $c^T x$ subject to $Ax \succeq b$ and $x \succeq 0$, which can be expressed in standard LP form as

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & \begin{bmatrix} -A \\ -I \end{bmatrix} x \preceq \begin{bmatrix} -b \\ 0 \end{bmatrix}. \end{aligned}$$

(We won't really bother too much with this kind of "routine conversion" to standard form much, but it's good to remember that it can be done.) We'll actually be able to solve this problem very soon, and we'll start seeing how to do that next week!

6 January 20, 2026

Our midterm quiz will be at the end of next week (in class on Thursday, with an alternate time the day before). It'll nominally cover chapters 1–4 of the book and problem sets 1–3; last year's quiz will be posted next week along with the solutions.

We're now in the part of the course where we list convex problems. The point is that if a practical problem reduces to a known problem, everything is completely tractable and it is fully solved (in the best sense). So since we'll start seeing hints that this is useful, we should become familiar with the common forms.

Last time, we showed that the diet problem (minimizing cost while meeting the minimum daily amounts of nutrients) is a linear program which can be expressed in standard form, but software will do the conversion for us so we'll omit this "matrix stuffing" from now on.

Example 81

Suppose we want to minimize a convex piecewise linear function of the form $f_0(x) = \max_{1 \le i \le m}(a_i^T x + b_i)$. Even though this is very far from affine, it can be equivalently written in the form

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & a_i^T x + b_i \le t \text{ for } 1 \le i \le m, \end{aligned}$$

which is essentially the epigraph variant of f_0. In words, $(x, t) \in \mathbb{R}^n \times \mathbb{R}$ can be anywhere in the convex set above the function, and we want to minimize t.

Example 82

The **Chebyshev center** of a polyhedron $\{x : a_i^T x \le b_i \text{ for } 1 \le i \le m\}$ is the center of the largest inscribed ball $\{x_c + u : ||u||_2 \le r\}$. (This is useful for things like the safe operating radius of a robot, or finding a safe deep point within a valid region). It might not be clear that this is an LP because of the "Euclidean ball" part, but notice that the inequality $a_i^T x \le b_i$ holds for all points in the r-ball if and only if

$$b_i \ge \sup\{a_i^T (x_c + u) : ||u||_2 \le r\} = a_i^T x_c + r||a_i||_2$$

(by Cauchy-Schwarz, since the worst case is when u points in the same direction as a_i). Therefore we can determine both x_c and r by solving the linear program in (x_c, r) variables,

$$\begin{aligned} \text{maximize} \quad & r \\ \text{subject to} \quad & a_i^T x_c + r||a_i||_2 \le b_i \text{ for } 1 \le i \le m. \end{aligned}$$

Indeed, even though there's an $||a_i||_2$ appearing in this inequality, the objective and our conditions are indeed linear in x_c and r. So if we want to find the Chebyshev center of any polyhedron, we know now that it is completely tractable in 1000 dimensions with 1000000 constraints.

Remark 83. *For the moment, we're holding aside the concept of what it actually means to solve convex problems – solvers typically deliver a single solution, even if there are multiple. (And if the problem is infeasible or unbounded, solvers can usually prove that in a concise way.)*

Definition 84

In a **quadratic problem (QP)** (introduced around 1952 for building portfolios in finance), we have a nonnegative definite P now and we want to solve

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} x^T P x + q^T x + r \\ \text{subject to} \quad & Gx \preceq h \\ & Ax = b. \end{aligned}$$

Geometrically, the level curves of a quadratic function are ellipsoids, going down towards some unconstrained minimizer. We then want to actually find the constrained minimizer while staying within some polyhedron.

At any minimizing point, the idea is that the negative gradient (where we want to point to decrease the function) will be an outward normal to the feasible set. This was used to design extremely light structures in aerospace engineering, where we need to save every kilogram possible.

The most famous quadratic program is just **least-squares** (minimize $||Ax - b||_2^2$) with no constraints, and that's a rare case where we have an analytical solution ($x^* = A^\dagger b$ for $A^\dagger$ the pseudoinverse). But then we can add linear constraints like nonnegative least squares $x \succeq 0$ or isotonic regression $x_1 \le \cdots \le x_n$, and the point is that with our methods it doesn't take any longer to solve even though we stop getting an analytic solution in general. (For example, if we want to measure wear of an industrial machine, and we add the prior that wear is monotone nondecreasing, then that's isotonic regression.)

Example 85

Consider a linear program with random cost, which works as follows. We have some LP with cost c, but now c is random with mean $\overline{c}$ and covariance Σ (for example, we have to commit to the orders beforehand even though we don't know the price). Our objective $c^T x$ is then also a random variable with mean $\overline{c}^T x$ and variance $x^T \Sigma x$, so we form the **risk-averse problem**

$$\begin{aligned} \text{minimize} \quad & \mathbb{E}[c^T x + \gamma \operatorname{Var}(c^T x)] \\ \text{subject to} \quad & Gx \preceq h, \\ & Ax = b, \end{aligned}$$

where $\gamma > 0$ is the risk-aversion parameter which controls tradeoff between cost and risk. Plugging in the mean and variance from above then turns this into a QP.

Note that if we have $\gamma < 0$ instead, turning the problem into a **risk-seeking** one, the problem immediately becomes intractable (NP-hard) and it's not convex. But it's very rare to have problems of that form that we actually want to solve, so it's good that $\gamma > 0$ is the tractable direction.

There is also a notion of a **quadratically constrained quadratic program (QCQP)**, which is like a QP with quadratic constraints.

Definition 86

In **second-order cone programming problems (SOCPs)**, we want to solve

$$\begin{aligned} \text{minimize} \quad & f^T x \\ \text{subject to} \quad & ||A_i x + b_i||_2 \le c_i^T x + d_i \text{ for } 1 \le i \le m, \\ & Fx = g, \end{aligned}$$

where $A_i \in \mathbb{R}^{n_i \times n}$ and $F \in \mathbb{R}^{p \times n}$. The naming comes from how the constraint is the inverse mapping of a second-order cone under a matrix mapping – we call these second-order constraints (SOCs). This form turns out to be more general than QCQP, since we can rewrite the objective of a QCQP in terms of a quadratic constraint.

Even if we haven't seen this exact form before, it turns out that 95% of convex optimization problems people solve reduce to this after conversion behind the scenes! We'll come back to this later.

Example 87

Suppose that our constraint vectors in a linear program are uncertain (since there's no way we know exactly how many nutrients there are in each ingredient). There are two common approaches to handling such "robust optimization" problems. At one extreme, we have the **deterministic worst-case** approach, where we model a_i as being in uncertainty ellipsoids and require $a_i^T x \le b_i$ for all a_i in the ellipsoid. And at the other extreme, we have the **stochastic** approach where we model a_i as a random variable and insist that the constraints hold with some probability $\mathbb{P}(a_i^T x \le b_i) \ge \eta$ (we call this a **chance constraint**).

In practice, the most common way we would actually handle uncertainty is just to ignore it and then do a posterior check to see if we care enough, but we could also add a small margin to each b_i and that's precisely what's going to happen in the subsequent discussion.

The point is that if we have uncertainty ellipsoids, we can convert the robust LP (which originally looks like it has infinitely many constraints)

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & a_i^T x \le b_i \text{ for all } a_i \in \mathcal{E}_i \text{ for } 1 \le i \le m \end{aligned}$$

into an SOCP with **finitely** many constraints

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & \overline{a}_i^T x + ||P_i^T x||_2 \le b_i \text{ for } 1 \le i \le m. \end{aligned}$$

So this basically gives us a "sophisticated way to auto-margin in different directions depending on uncertainty." (And uncertainty sets don't have to be ellipsoids – different fields use different things for various reasons.)

Similarly, we can handle the stochastic approach as an SOCP as well: let a_i be normal with parameters $(\overline{a}_i, \Sigma_i)$, so we can write the probability our constraint is satisfied in terms of the normal CDF. Then

$$\mathbb{P}(a_i^T x \le b_i) \ge \eta \quad \Longleftrightarrow \quad \overline{a}_i^T x + \Phi^{-1}(\eta)||\Sigma_i^{1/2} x||_2 \le b_i,$$

and therefore if $\eta \ge \frac{1}{2}$ this is positively curved (we need the satisfactory probability to be at least 50 percent) and we can rewrite this as

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & \overline{a}_i^T x + \Phi^{-1}(\eta)||\Sigma_i^{1/2} x||_2 \le b_i \text{ for } 1 \le i \le m. \end{aligned}$$

In particular, notice that these two forms are actually identical even though they come from different formulations and fields, and this is not the last time we'll see this. It's kind of like how there are many ways to arrive at least-squares regression, which is a sign that it's a good concept.

Definition 88

More generally, a **conic form problem** takes the form

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Fx + g \preceq_K 0 \\ & Ax = b \end{aligned}$$

where the constraint now involves a generalized inequality with respect to a proper cone K. (Linear programming is then a conic form problem with $K = \mathbb{R}^m_+$.) Just like with standard convex problems, feasible and optimal sets are convex and local optima are global.

Example 89

In a **semidefinite program**, we have symmetric matrices F_i, G and want to solve

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & x_1 F_1 + \cdots + x_n F_n + G \preceq 0 \\ & Ax = b. \end{aligned}$$

We thus have a linear matrix inequality as constraint, and this lets us solve problems like matrix norm minimization: if we want to minimize the norm of a matrix $||A(x)||_2$ for $A(x) = A_0 + x_1 A_1 + \cdots + x_n A_n$ (where the $A_i \in \mathbb{R}^{p \times q}$ may not be symmetric or even square), we can write it equivalently as

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & \begin{bmatrix} tI & A(x) \\ A(x)^T & tI \end{bmatrix} \succeq 0 \end{aligned}$$

for variables $x \in \mathbb{R}^n, t \in \mathbb{R}$, since the matrix above is affine in each of x, t. Indeed, this follows from $||A||_2 \le t \iff A^T A \preceq t^2 I$, so we have the epigraph representation to the rescue again.

There's also a hierarchy here where SDP actually subsumes LPs and SOCPs: for example, the LP constraint can be written as a diagonal matrix constraint. But it's not so important for us.

We'll now move to the idea of transforming problems, which will be super useful for lots of applications. Decades ago, we'd learn a lot of these tricks and implement them in FORTRAN or something – we don't have to do that now but it's still useful to know the types of transformations that occur.

Proposition 90

Suppose $\phi : \mathbb{R}^n \to \mathbb{R}^n$ is a one-to-one mapping. Then we can change variables $x = \phi(z)$ and solve the equivalent problem in z by replacing f_i with $\tilde{f}_i(z) = f_i(\phi(z))$ and h_i with $\tilde{h}_i(z)$. Then the optimal solution can be recovered via $x^* = \phi(z^*)$.

Example 91

Suppose we have the nonconvex problem

$$\begin{aligned} \text{minimize} \quad & \frac{x_1}{x_2} + \frac{x_3}{x_1} \\ \text{subject to} \quad & \frac{x_2}{x_3} + x_1 \le 1, \end{aligned}$$

where we have an implicit constraint $x \succeq 0$ because all variables appear in the denominator. This is not jointly convex (even though it is convex in each variable, since the off-diagonal Hessian entries cause problems), but if we change variables via $x = e^z$ (take the log) to turn this into

$$\begin{aligned} \text{minimize} \quad & \exp(z_1 - z_2) + \exp(z_3 - z_1) \\ \text{subject to} \quad & \exp(z_2 - z_3) + \exp(z_1) \le 1, \end{aligned}$$

which is indeed now a convex problem (we can check this by repeatedly applying the composition rule).

In a lot of applications, we will be given a problem that we want to solve and it will be convex, but then in a lot of other problems we can do weird parametrizations (like replacing speed with $\frac{1}{\sqrt{\text{speed}}}$) which make it convex. And one practical piece of advice is that unless we find two isolated local minima or someone has proved that a problem is NP-hard, we should not claim that something is definitely not convex.

Proposition 92

Suppose ϕ_0 is monotone increasing, $\psi_i(u) \le 0$ if and only if $u \le 0$ for all i, and $\phi_i(u) = 0$ if and only if $u = 0$ for all i. Then we can take our standard form optimization and replace f_0, f_i, h_i with $\phi_0(f_0(x)), \psi_i(f_i(x)), \phi_i(u_i(x))$ and the problem is equivalent.

Proposition 93

Suppose ϕ_0 is a monotone decreasing problem (for example $f(x) = -x$ or $f(x) = \frac{1}{x}$). Then **maximizing** $f_0(x)$ **is equivalently to minimizing** $\phi_0(f_0(x))$.

Proposition 94

We can **eliminate linear equality constraints** in a problem by reducing to the nullspace as follows: we can always write $Ax = b$ equivalently as $x = Fz + x_0$ for some matrix F and vector x_0, so

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & Ax = b \end{aligned}$$

is equivalent to

$$\begin{aligned} \text{minimize} \quad & f_0(Fz + x_0) \\ \text{subject to} \quad & f_i(Fz + x_0) \le 0 \text{ for } 1 \le i \le m, \end{aligned}$$

where now we minimize over z instead of x.

On the other hand, we can also "un-eliminate" or add new variables: we can take the problem

$$\begin{aligned} \text{minimize} \quad & f_0(A_0 x + b_0) \\ \text{subject to} \quad & f_i(A_i x + b_i) \le 0 \text{ for } 1 \le i \le m \end{aligned}$$

and equivalently add a bunch more variables and instead try to solve

$$\begin{aligned} \text{minimize} \quad & f_0(y_0) \\ \text{subject to} \quad & f_i(y_i) \le 0 \text{ for } 1 \le i \le m, \\ & y_i = A_i x + b_i \text{ for } 0 \le i \le m \end{aligned}$$

over the variables x, y_i. This "un-eliminating constraints" will turn out to actually be quite useful, even though it has more variables and constraints and thus might seem like a harder problem!

Proposition 95

We can introduce **slack variables** for linear inequalities: the constraint $a_i^T x \le b_i$ is equivalent to requiring that $a_i^T x + s_i = b_i$ and $s_i \ge 0$, where now instead of minimizing over x we minimize jointly over (x, s) and have simpler-looking inequalities to deal with.

The next transformation is something we've seen in multiple examples already:

Proposition 96

The standard form convex problem is equivalent to the **epigraph form**

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & f_0(x) - t \le 0, \\ & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & Ax = b, \end{aligned}$$

where we now minimize over both x and t.

Our next example is connected to the fact that partial minimization preserves convexity. Dynamic programming is a particular example of this:

Proposition 97

Suppose we want to minimize $f_0(x_1, x_2)$ jointly over the variables x_1, x_2 subject to $f_i(x_1) \le 0$ for all $1 \le i \le m$. Then defining $\tilde{f}_0(x_1) = \inf_{x_2} f_0(x_1, x_2)$, it is equivalent to just minimize $\tilde{f}_0$ subject to the constraints.

Definition 98

In **convex relaxation**, we start with a nonconvex problem where we want to minimize some function $h(x)$ subject to $x \in C$. If we find a **convex minorant** $\hat{h}$ with $\hat{h} \le h$ pointwise, and we find some set $\hat{C} \supseteq C$ (such as the convex hull) which can be described by some linear inequalities and equalities. We can then solve the **convex relaxation problem**

$$\begin{aligned} \text{minimize} \quad & \hat{h}(x) \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m \\ & Ax = b. \end{aligned}$$

The optimal value on this relaxation is a lower bound on the optimal value for the original problem, since we have a smaller function and a larger set of possible points to consider.

Example 99

In **boolean LP**, we begin with a mixed integer linear problem with two sets of variables x, z, where each z_i must be in the set $\{0, 1\}$. This problem is generally hard to solve, but in the LP relaxation we now relax to requiring $z_i \in [0, 1]$. Possibly unintuitively, this actually **makes the problem tractable** even though we now have an infinite set of possibilities, it gives a lower bound on the original problem, and it can be used as a heuristic for approximately solving the original problem (for example, by rounding our optimal solution based on a threshold).

There's actually an entire industry based around mixed integer linear programming; for example, almost all problems in operations research reduce to this. If we do these heuristic methods, we can say things like "the best solution is 21.6 and I can find a solution with 23" even if we can't prove that it's optimal.

Remark 100. *Most of the solvers for MILPs are unfortunately commercial, making it kind of hard to replicate research (and often those commercial solvers will try to get people to buy their products so that the solution can be "actually optimal"). But a lot of the time, many of the constraints end up being kind of fake (for example, it's something made up about "how much utility is lost if it takes 1 day"), and so one defense in these situation is that "arguing about being 5 percent suboptimal" can be countered by "arguing that the model is 5 percent accurate," which is generally quite difficult to do.*

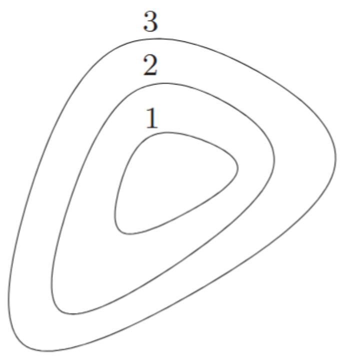
In practice, we will do none of the more basic transformations here, since they will be done for us automatically by solvers. But other changes of variables will have to be done ourselves, and we'll talk about that next time!

7 January 20, 2026 (Problem Session)

Problem 101

We'll begin by relating the concepts of level curves and convex functions $f : \mathbb{R}^2 \to \mathbb{R}$. Our goal is to understand the strongest statements we can make (convex, concave, quasiconvex, quasiconcave) given the level curves $f(x_1, x_2) = c$.

Consider the example shown below (where the curves correspond to $f(x_1, x_2) = 1, 2, 3$):



First of all, notice that we need to have monotonicity for the level sets to correspond to anything convex or concave (or quasiconvex or quasiconcave). For example, if there were a level curve $f(x, y) = 5$ between the curves with $f(x, y) = 1$ and $f(x, y) = 2$, then a line segment between the 1-set and 2-set intersecting the 5-set would automatically break convexity.

This example above **is consistent with a quasiconvex function**, since the sublevel sets are convex regions (remember that the sublevel set $f \leq 2$ includes both the second ring and the inner ring). And it looks like it might be "bowl-shaped," so at first we might think it could be convex. However **convexity actually breaks down**, because on the rightmost part the level set of 2 is closer to that of 1 than that of 3. Thus if we draw a line segment on the right side, we break the convexity assumption.

Meanwhile, **quasiconcavity** is automatically false because the superlevel set $\{f(x, y) \geq 1\}$ is definitely not convex. And therefore **concavity will automatically fail** as well. (Alternatively, we can also draw a line segment along the bottom left direction to show that the concavity condition fails.)

Problem 102

Next, we'll show that the running average

$$F(x) = \frac{1}{x}\int_0^x f(t)dt$$

for a convex function $f : \mathbb{R} \to \mathbb{R}$ (with domain at least containing $\mathbb{R}_+$) is convex.

This problem can be done with the u-substitution $t = sx$, $dt = x\,ds$, so that

$$\frac{1}{x}\int_0^x f(t)dt = \frac{1}{x}\int_0^1 f(sx)xds = \int_0^1 f(sx)ds.$$

Now each $f(sx)$ is convex (by precomposition of f), so the integral over s will also be convex (since that's essentially an infinite weighted sum). But more directly, we can say that if $z = \theta x + (1-\theta)y$, then

$$\begin{aligned}\boxed{F(z)} &= \int_0^1 f((\theta x + (1-\theta)y)s)ds \\ &= \int_0^1 f(\theta(xs) + (1-\theta)(ys))ds \\ &= \int_0^1 (\theta f(xs) + (1-\theta)f(ys))\,ds,\end{aligned}$$

where in the last step we've used convexity of f on the line segment connecting xs and ys. And now by linearity this can be written out as

$$\theta\int_0^1 f(xs)ds + (1-\theta)\int_0^1 f(ys)ds = \boxed{\theta F(x) + (1-\theta)F(y)}$$

which is exactly what we wanted to show.

Remark 103. *This "running average" can be thought of more concretely in the following way: $(f_1, \cdots, f_n)$ is a sequence of values with increasing adjacent differences, then the discrete running average is $F(k) = \frac{f_1+\cdots+f_k}{k}$. And it's worth remembering that Riemann sum approximations of integrals essentially come from averaging a bunch of values of the function at adjacent points too.*

Note also that there is no need for the function f to be differentiable – for example, it could be a piecewise linear function which is still continuous and convex. So we should not use an approach which requires taking a second derivative of F.

Problem 104

For a practical example now, consider a problem where a vehicle uses fuel at a rate $f(s)$ of the speed s it is traveling at, where f is positive, increasing, and convex on $\mathbb{R}_+$. We wish to prove that the following functions are convex:

(a) $g(d, t)$, the total fuel used when a vehicle moves distance d in time t at a constant speed,

(b) $h(d)$, the minimum fuel used to move a distance d at some constant speed s.

For part (a), we know that the vehicle moves at speed $\frac{d}{t}$, meaning that it consumes fuel at a rate of $f(\frac{d}{t})$ and thus

$$g(d, t) = tf\left(\frac{d}{t}\right).$$

This is the **perspective** of a convex function f, hence convex.

Meanwhile for part (b), we're now optimizing over the best possible amount of time to take from part (a), so

$$h(d) = \min_{t>0}\left\{tf\left(\frac{d}{t}\right)\right\} = \min_{t>0} g(d, t).$$

But because g is jointly convex in d, t and we minimize over a convex set for t, "partial minimization" tell us that $h(d)$ is also convex. (Note that partial minimization only works if we're minimizing over one of the variables in which we know it's already jointly convex, and also if we minimize over a convex set.)

Problem 105

For our next problem, we will determine algebraically whether a function is convex /concave and also whether it is quasiconvex /quasiconcave.

First consider $f(x) = e^x - 1$ on $\mathbb{R}$. Drawing a picture, we see that the function curves upward but is monotone increasing. Thus it will be **convex** and also **both quasiconvex and quasiconcave** but not concave. Indeed, $f''(x) = e^x$ is positive for all x, and the sublevel and superlevel sets are empty, all of $\mathbb{R}$, or half-infinite intervals.

Next for a more complicated example, let's do $f(x_1, x_2) = x_1x_2$ on the positive orthant $\mathbb{R}^2_{++}$. This is a bit harder to visualize, so we can compute the Hessian matrix (of second derivatives): we find that

$$\mathcal{H}_f = \nabla^2 f(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

But the eigenvalues of this matrix are 1 and -1 (one nice way to see this is that the sum of the eigenvalues is the trace of the matrix, 0, and the product is the determinant of the matrix, -1; alternatively $\det(\lambda I - \mathcal{H}_f) = \lambda^2 - 1$), so the Hessian is indefinite. Thus the function is neither convex nor concave.

This is a rather intuitive way to solve the problem, so we can also think about this graphically in one of two ways.

- First of all, along the line $x_1 = x_2$, the function takes on the values $f(x_1, x_1) = x_1^2$, so it is convex along that line, but along the line $x_1 + x_2 = c$, it instead takes on the value $f(x_1, c - x_1) = x_1(c - x_1)$, meaning it is concave along that line.
- Secondly, we can take the approach from the beginning of this problem session: draw the level curves $x_1x_2 = 1, 4, 9$ passing through $(1, 1), (2, 2), (3, 3)$ respectively. The sublevel set for $x_1x_2 = 1$ is not convex, so the function is not quasiconvex and thus not convex. (On the other hand, the superlevel sets are in fact convex here, so the function is still **quasiconcave**.) And concavity fails because the midpoint $(2, 2)$ of the segment between $(1, 1)$ and $(3, 3)$ has $f(2, 2) = 4$, which is smaller than $\frac{1}{2}f(1, 1) + \frac{1}{2}f(3, 3) = 5$.

Problem 106

Finally, for another practical example used in economics, consider the "CRRA (Constant Relative Risk Aversion)" utility functions

$$u_0(x) = 0, \quad u_\alpha(x) = \frac{x^\alpha - 1}{\alpha} \text{ for } 0 < \alpha \le 1$$

on $\mathbb{R}_+$. The idea is that x is an outcome and the utility function tells us how desirable the outcome is; there are diminishing returns and thus we want u to be concave. Indeed, we can check that all u_α are concave and monotone increasing, and $u_\alpha(1) = 0$ and $\lim_{\alpha \to 0} u_\alpha(x) = u_0(x)$ for all x.

Indeed, we check that last statement by computing the limit

$$\lim_{\alpha \to 0} \frac{x^\alpha - 1}{\alpha} = \lim_{\alpha \to 0} \frac{(\log x)x^\alpha}{1} = \frac{\log(x)1}{1}$$

by L'Hopital's rule. (The tricky step here is to compute the derivative of x^α as a function of α, but we can do this with implicit differentiation or by rewriting it as $e^{\alpha \log x}$.) And to show monotonicity and concavity and $u_\alpha(1) = 0$, we plug in 1 to check the last fact, and then we take derivatives and do sign analysis for the others:

$$u_\alpha'(x) = \frac{1}{\alpha}(\alpha x^{\alpha-1}) = x^{\alpha-1}, \quad u_\alpha''(x) = (\alpha - 1)x^{\alpha-2},$$

so the derivative is always positive but the second derivative is always negative, which is what we wanted. (These same properties also hold for $\log x$, by again taking the appropriate derivatives.)

8 January 22, 2026

We got up to the concept of relaxation last time – we'll see lots of practical instances of that, and one of the main lessons is that having a smaller constraint set (even finitely many points instead of infinitely many) can actually make the problem more difficult. The boolean LP we discussed last time is such an example, since we can write problems like 3-SAT as a mixed integer linear program.

Fact 107

This gets us now to the useful concept of **disciplined convex programming**. To specify a DCP-compliant problem, we specify our objective in the form "minimize this scalar convex expression" or "maximize this sclara concave expression," and then we specify constraints in Python like `(convex expression) <= (concave expression)` or `(concave expression) >= (convex expression)` or `(affine expression) == (affine expression)`, where all expressions are actually DCP certified via the composition rule.

All problems of this form (which return `True` for `is_dcp`) can be automatically transformed to standard forms and then solved, and notably most of us don't really even need to know how that process was carried out.

For example, if we want to minimize $||x||_1$ subject to $Ax = b$ and $||x||_\infty \leq 1$, where A, b are given parameters, we can write it as follows:

```
------------------
import cvxpy as cp

A, b = (given values)

x = cp.Variable(n)
objective = cp.norm(x, 1)
constraints = [
    A @ x == b,
    cp.norm(x, 'inf') <= 1,
]
problem = cp.Problem(cp.Minimize(objective, constraints))
prob.solve()
------------------
```

Here remember `x` is not a vector – it's a variable with properties like `Affine`, and then `objective` is the expression we're trying to minimize (more precisely, the objective is specifically to minimize that quantity). Things like `@` are then overloaded to apply to variables, and in our constructor we pass in the objective and constraints. Then `solve` will compile our problem if it is DCP.

Remark 108. *During the solving process, CVXPY has the form of a "rewriting compiler," meaning that we perform a series of reductions / transformations to equivalent problems. (Equivalent here means that we have two methods that let us transform optimal variables in a bijective manner.) So we have a sequence of equivalent problems $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_N$, until we match a standard form like an LP, QP, SOCP, or SDP. A specialized solver is then used on that problem, and then we pull everything back through the retrievals to get the original solution. And when we run `solve`, a side effect of that is that every variable in our problem will have its `value` attributes overwritten by optimal values.*

Definition 109

There is a particular class of problems called **geometric programming** which are not convex but where we can transform them to a convex problem. It involves the following notions: in this field a **monomial function** is a function of the form $f(x) = cx_1^{a_1} \cdots x_n^{a_n}$ on the domain $\mathbb{R}^n_{++}$, where $c > 0$ and a_i can be arbitrary **real** numbers, and **posynomial functions** are sums of monomials. A **geometric program** then take the form

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_1(x) \le 1 \text{ for } 1 \le i \le m, \\ & h_i(x) = 1 \text{ for } 1 \le i \le p, \end{aligned}$$

where f_i are posynomial and h_i are monomial.

The generic transformation to use for these problems is that because our variables are positive, we can use the variables $y_i = \log x_i$ instead. The monomial $cx_1^{a_1} \cdots x_n^{a_n}$ then becomes the affine function $a^T y + \log c$, and posynomials become logsumexps. Thus we end up minimizing a convex function subject to convex and affine constraints, and so the class of GPs is sort of like the class of QPs but where you can only use logsumexp instead of only using quadratic functions.

Example 110

Suppose we want to do Frobenius norm diagonal scaling, meaning that we want a diagonal matrix $D = \text{diag}(d)$ which minimizes $||DMD^{-1}||_F^2$. (So we rescale our basis vectors and want to minimize the sum of the squares of the matrix entries.) Our objective can be expressed as

$$||DMD^{-1}||_F^2 = \sum_{i,j=1}^{n} (DMD^{-1})_{ij}^2 = \sum_{i,j=1}^{n} \frac{M_{ij}^2 d_i^2}{d_j^2},$$

which is a posynomial in d with exponents $0, 2, -2$. So in convex form with $y = \log d$, our objective is $\log\left(\sum_{i,j=1}^{N} \exp\left(2(y_i - y_j) + \log|M_{ij}|\right)\right)$, which is indeed convex.

We'll now move to **quasiconvex optimization**, which sets up problems of the form

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & Ax = b \end{aligned}$$

where f_0 is quasiconvex and the other f_is are convex. Note that **locally optimal points can now no longer be globally optimal**, so we have to be a little more careful than before!

Example 111

A famous example of this is **linear-fractional program** of the form

$$\begin{aligned} \text{minimize} \quad & \frac{c^T x + d}{e^T x + f} \\ \text{subject to} \quad & Gx \preceq h, \\ & Ax = b, \end{aligned}$$

which we can indeed solve using general quasiconvex optimization problems. It turns out that in this particular case we can actually do a very clever transformation by adding in a scalar variable to homogenize, yielding the equivalent LP

$$\begin{aligned} \text{minimize} \quad & c^T y + dz \\ \text{subject to} \quad & Gy \preceq hz, \quad Ay = bz, \quad e^T y + fz = 1, \quad z \geq 0, \end{aligned}$$

and then via a perspective transformation we recover $x^* = \frac{y^*}{z^*}$.

Example 112 (von Neumann model of a growing economy)

Suppose $x, x^+ \in \mathbb{R}^n_{++}$ are the activity levels of n different economic sectors in this period and the next, and let A, B be matrices that tell us how much we produce and consume per unit of activity (more precisely, $(Ax)_i$ is how much of good i is produced in this period, and $(Bx^+)_i$ is how much we consume in the next period). Impose the constraint that $Bx^+ \preceq Ax$, so next period's consumption can be no more than this period's production. (We could have other variations on this too, which would be like adding an endowment.) Our goal is then to maximize growth rate; specifically, if $\frac{x_i^+}{x_i}$ is the growth rate of sector i, we want to solve the following problem over x, x^+:

$$\begin{aligned} \text{maximize} \quad & \min_{1 \leq i \leq n} \frac{x_i^+}{x_i} \\ \text{subject to} \quad & x^+ \succeq 0, \\ & Bx^+ \preceq Ax. \end{aligned}$$

This is not a convex problem, but it is quasiconvex: we have to confirm that $\min_{1 \leq i \leq n} \frac{x_i^+}{x_i}$ is quasiconvex, meaning that if we look at all pairs of economic activities where we get at least 10 percent growth in all sectors, that set is convex. Indeed, such a set is a polyhedron because it's bounded by linear constraints.

In the general case, quasiconvex functions are represented by a family of functions ϕ_t, where each $\phi_t(x)$ is convex in x, and where the t-sublevel sets of f_0 are the 0-sublevel-sets of ϕ_t:

$$f_0(x) \leq t \iff \phi_t(x) \leq 0.$$

For example if $f_0(x) = \frac{p(x)}{q(x)}$, where p is convex and nonnegative and q is concave and positive. This is always quasiconvex, since the t-sublevel set for any $t > 0$ is exactly $p(x) - tq(x) \geq 0$; thus we define $\phi_t(x) = p(x) - tq(x)$, and this is indeed convex in x.

Proposition 113

Thus, we can approximately solve quasiconvex optimization via the following bisection method. For each fixed t, consider the feasibility problem

$$\phi_t(x) \le 0, \quad f_i(x) \le 0 \text{ for } 1 \le i \le m, \quad Ax = b.$$

If we have a feasible point of this form, then we know that the optimal solution p^* satisfies $p^* \le t$, otherwise $t \le p^*$. Thus we can do binary search to see where the problem goes from being feasible to infeasible, which requires $\lceil \log_2(\text{window}/\text{error}) \rceil$ iterations of a convex problem.

Finally, yet another variant is the idea of **multicriterion optimization**, where instead of having one objective we have q of them and want to solve

$$\begin{aligned} \text{minimize} \quad & f_0(x) = (F_1(x), \cdots, F_q(x)) \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & Ax = b. \end{aligned}$$

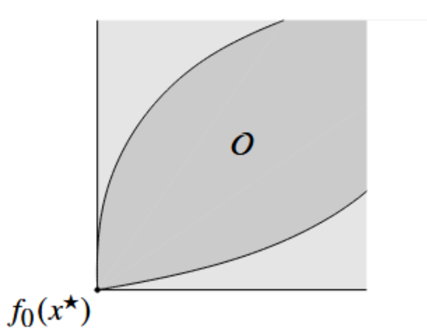
Really, most problems are multi-objective and we need to do tradeoffs between them while wanting all of them to be small. Minimizing a vector can be interpreted in various ways; we say that x^* is **optimal** if $f_0(x^*) \preceq f_0(y)$ for all feasible y. But optimal points only occur if the objectives are noncompeting, meaning we have a simultaneous minimizer; this does not happen that often. So instead we have the following concept:

Definition 114

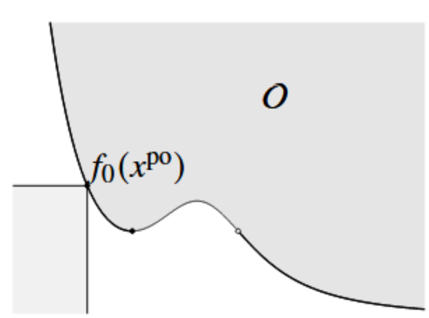
A feasible x **dominates** another feasible $\tilde{x}$ if $f_0(x) \preceq f_0(\tilde{x})$, and $F_i(x) < F_i(\tilde{x})$ for at least one i. A feasible point is **Pareto optimal** if it is not dominated by any other feasible point.

For example if we have two objectives, we can imagine plotting the values of $f_0 = (F_1, F_2)$ in the plane. Then x dominates $\tilde{x}$ if the value $f_0(x)$ is within the bottom-left rectangle bounded by $f_0(\tilde{x})$. Of course, there can be many Pareto optimal points if they are incomparable as vectors – we call the set of Pareto optimal objective values the **optimal trade-off curve** or **optimal trade-off surface** for $q = 2, 3$.

In both figures below (taken from the slides for the course), $\mathcal{O}$ denotes the set of achievable objective values. In the left case, we have an optimal point because $f_0(x)$ is actually the **minimum** value of $\mathcal{O}$, and in the right case we have a set of **minimal** values of $\mathcal{O}$ which are all Pareto optimal (because there are no points in those bounded rectangles drawn). The dark curve in the right diagram shows the set of Pareto optimal points.



$x^\star$ is optimal

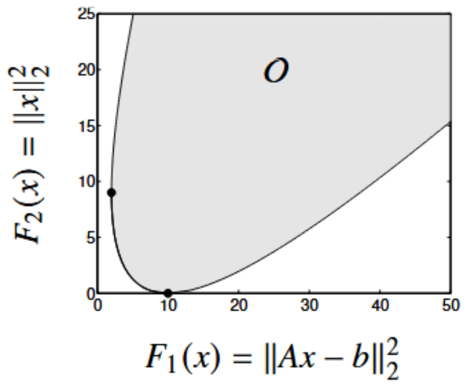


x^{po} is Pareto optimal

Example 115

In regularized least-squares we have the two objectives $(||Ax - b||_2^2, ||x||_2^2)$ corresponding to loss and regularization. The set of Pareto optimal points is then giving us a continuous curve where we need to have a tradeoff between loss and regularization.

The example below (also taken from the slides) is a case where $A \in \mathbb{R}^{100\times 10}$. The dark curve on the bottom left is the set of Pareto optimal points, and we'll see soon that for convex problems it will always be this kind of nice continuous curve (unlike in the example above).



Example 116

Suppose we want to do risk-return trade-off in portfolio optimization, where $x \in \mathbb{R}^n$ is a vector where x_i represents the fraction invested in asset i and we can only buy assets. Suppose $\overline{p} \in \mathbb{R}^n$ is the the mean return and Σ is the covariance of the asset returns; our goal is then to minimize $(-\overline{p}^T x, x^T \Sigma x)$ subject to $\mathbf{1}^T x = 1$ and $x \succeq 0$. We then get a curve of Pareto optimal portfolios which is the "optimal risk-return set." Solving a bunch of QPs (as we will do later in the course) will let us do exactly that.

Proposition 117

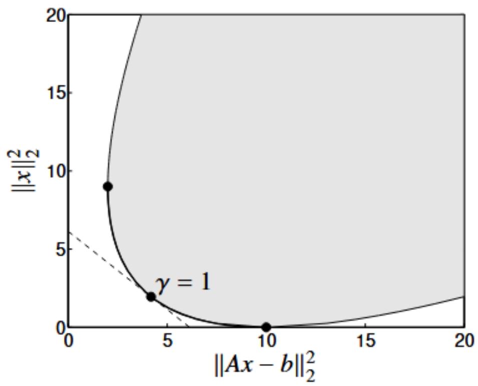
There are lots of ways to find Pareto optimal points, and in practice we want to compute the whole optimal curve. The standard way to do this is **scalarization**, where we combine the multiple objectives into a single one. For example, we can choose some $\lambda \succ 0$ and solve the problem

$$\begin{aligned} \text{minimize} \quad & \lambda^T f_0(x) = \lambda_1 F_1(x) + \cdots + \lambda_q F_q(x) \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & h_i(x) = 0 \text{ for } 1 \le i \le p. \end{aligned}$$

(We could do things like logsumexp instead of just linear combinations.) For any choice of λ (relative weights of our objectives) an optimal x for this problem yields a Pareto-optimal point for the original problem (because otherwise a dominating point would do a better job here).

Often in practice, we just set $\lambda_1 = 1$ for our primary objective and then vary the other λ_is in terms of how much we care on a relative scale. If our original problem is convex, then the scalarized problem is also convex, and it turns out we get "almost all" points on the Pareto optimal curve this way because λ is specifying the supporting (tangent) hyperplane to our set of feasible points! In practicular, λ also has the interpretation of the tradeoff rate at any Pareto optimal point.

For regularized least-squares as above, $\lambda = (1, 1)$ would yield the following point which minimizes $||Ax - b||_2^2 + ||x||_2^2$:



In this case, the only points we can't actually achieve are the boundary points which would require vertical or horizontal supporting hyperplanes. But we'll talk more about this later. And similarly in the risk-return tradeoff problem, we would now be trying to minimize $-\overline{p}^T x + \gamma x^T \Sigma x$, which is the negative of the risk-adjusted return for some risk-aversion parameter γ.

This finishes up one chunk of the course, and now we're transitioning to the "fun part" of the course where we get to start doing stuff by solving practical problems. We'll do one more theoretical topic, **duality**, which will in particular give us an explanation of Lagrange multipliers.

Definition 118

Start with a standard-form problem which is not necessarily convex of the form

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & h_i(x) = 0 \text{ for } 1 \le i \le p \end{aligned}$$

on some domain $\mathcal{D}$. The **Lagrangian** is a function $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ with domain $\mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$ of the form

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{n} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x),$$

where λ_i, ν_i are the **Lagrange multipliers**.

We may have seen this before in calculus where we were told "behaviorally" how to solve optimization problems.

Definition 119

With the notation above, the **Lagrange dual function** $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ is given by

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu) = \inf_{x \in \mathcal{D}} \left\{ f_0(x) + \sum_{i=1}^{n} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right\}.$$

Interestingly, g is **always concave** in λ, ν no matter what f_0 was, since for each fixed x the function inside the infimum is affine.

One interpretation of this is that the original problem is an economy with some regulations given by the constraints, and here we introduce a price associated to each constraint in the "free market." For instance, if we originally required the hard constraint $f_1(x) \le 0$, the Lagrangian now turns this into a soft constraint where we get penalized at rate λ_1 for violating that, and it's also actually better to have a margin as well. (Mathematically, this is kind of silly, since we're basically approximating the function which is 0 for $x \le 0$ and ∞ for $x > 0$ with some line through the origin.)

We have the useful **lower bound property** that

$$\lambda \succeq 0 \implies g(\lambda, \nu) \le p^*;$$

indeed for any feasible point $\tilde{x}$ we have $f_0(\tilde{x}) \ge L(\tilde{x}, \lambda, \nu) \ge g(\lambda, \nu)$, so if we take the infimum over all feasible points we get $p^* \ge g(\lambda, \nu)$ as well.

Example 120

Suppose we want to minimize $x^T x$ subject to $Ax = b$ (we know the solution will be $A^\dagger b$). We don't need a lower bound for this since we know how to solve it, but let's see how the Lagrangian strategy provides us with a parametrized family of lower bounds for the optimal value. First form the Lagrangian $L(x, \nu) = x^T x + \nu^T(Ax - b)$; if we minimize over x, we set the gradient equal to zero and find that $x = -\frac{1}{2}A^T\nu$. Thus

$$g(\nu) = L\left(-\frac{1}{2}A^T\nu, \nu\right) = -\frac{1}{4}\nu^T AA^T \nu - b^T\nu,$$

so for **any** ν we have $p^* \ge -\frac{1}{4}\nu^T AA^T\nu - b^T\nu$.

Example 121

Similarly for a standard form linear program, where we want to minimize $c^T x$ subject to $Ax = b$, $x \succeq 0$, the Lagrangian is

$$L(x, \lambda, \nu) = c^T x + \nu^T(Ax - b) - \lambda^T x = -b^T\nu + (c + A^T\nu - \lambda)^T x.$$

So we want to minimize an affine function, which will yield $-\infty$ unless the affine function itself is constant: thus

$$g(\lambda, \nu) = \begin{cases} -b^T\nu & \text{if } A^T\nu - \lambda + c = 0, \\ -\infty & \text{otherwise.} \end{cases}$$

This means that we get a lower bound of $-\infty$ in this framework for lots of choices of λ, ν, which is true but just not very informative.

9 January 27, 2026

Today's lecture is being given by one of the TAs, Daniel Cederberg.

Remark 122. *Before we start with today's lecture, it's worth knowing how to remember the composition rule for convexity $f(x) = h(g(x))$ without needing to remember a bunch of different cases. The idea is that we need to know the curvature and monotonicity of h, and we also need the curvature of g. One of the rules we've seen is that if h is convex and increasing and g is convex, then $h(g(x))$ is convex. But more generally, if h is increasing we need h and g to have the same curvature, and if h is decreasing we need h and g to have opposite curvature. In such a situation, h and f will have the same curvature – we can't change the curvature of the outer function only via composition rules.*

We'll continue our discussion of duality today – this is a general topic in mathematics which broadly means that we study the same object from two different perspectives. (For example, we can view a signal in the time domain, or we can apply the Fourier transform to get things in a frequency representation.) We'll see this in the context of optimization. Recall that for a standard form problem, we can define a Lagrangian $L(x, \lambda, \nu) = f_0(x) + \sum_i \lambda_i f_i(x) + \sum \nu_i h_i(x)$ (these constants λ_i, ν_i have various names, but we'll think of them here as Lagrange multipliers). We can then define the dual function $g(\lambda, \nu)$ by minimizing L over the original variable x; the key fact is that this is always concave in those new variables λ, ν (even if the original problem is not convex), and for any $\lambda \succeq 0$ we always get a lower bound $g(\lambda, \nu) \le p^*$ on the optimal value of the original problem.

Last time, we saw two examples where we took some starting problem and derived the dual function. For example, in Example 120 the Lagrangian is quadratic in x, which allows us to minimize it analytically and derive a nice expression for $g(\nu)$. (Indeed, we confirm that in this case $-\frac{1}{4}\nu^T AA^T \nu - b^T \nu$ is a concave function.) Similarly in Example 121, we formed a Lagrangian by associating a multiplier to the inequality constraint $Ax = b$ and also a multiplier to the equality constraint $x \succeq 0$ (note that we have to flip the sign to get it in standard form). In that case L was actually affine in x, so minimizing it over all x gave us either a constant or $-\infty$.

Remark 123. *Remember that when we optimize the dual function, we optimize over all $x \in \mathcal{D}$ in the problem domain (the intersection of all domains of our original functions). This is different from restricting ourselves to the feasible set*

Example 124

Suppose now that we want to perform an **equality-constrained norm minimization problem**, where we minimize $||x||$ subject to $Ax = b$. We'll do this in more detail than usual. Our Lagrangian is then

$$L(x, \nu) = ||x|| + \nu^T(b - Ax) = b^T\nu + ||x|| - (A^T\nu)^T x.$$

(For equality constraints, there is a sign ambiguity – we can either treat it as $Ax - b = 0$ or $b - Ax = 0$, but it doesn't matter. Remember that Lagrange multipliers for inequality constraints care about having $f_i(x) \le 0$ and that's why we care about $\lambda_i \ge 0$, but for $h_i(x) = 0$ we have no such condition.)

Optimizing this over x yields the dual function

$$g(\nu) = b^T\nu + \inf_x \left\{||x|| - (A^T\nu)^T x\right\}.$$

There turns out to be a deep connection between duality and conjugate functions, and we can see that by rewriting this as

$$\begin{aligned} g(\nu) &= b^T\nu - \sup_x \left\{(A^T\nu)^T x - ||x||\right\} \\ &= b^T\nu - f_0^*(A^T\nu), \end{aligned}$$

where f_0^* is the conjugate function for the norm. And in the textbook, we see that the conjugate function yields 0 on the unit ball in the dual norm and ∞ otherwise:

$$g(\nu) = \begin{cases} b^T\nu, & \text{if } ||A^T\nu||_* \le 1 \\ -\infty, & \text{otherwise.} \end{cases}$$

Notably, we managed to compute $g(\nu)$ even though we had an *arbitrary* norm and there is no analytical solution, and now the lower bound property tells us that if $||A^T\nu||_* \le 1$, then $p^* \ge b^T\nu$. (And one simple method to find to find something that satisfies this property is to take any random ν and then normalize.)

All of the problems we've seen so far are simple – they're convex and we can solve them easily. But duality also applies to non-convex problems, and some people say that this is where it's actually useful.

Example 125

In the **two-way partitioning** problem, we wish to minimize x^TWx subject to $x_i^2 = 1$ for all $1 \le i \le n$. This is a nonconvex problem with 2^n discrete points – we can think of it as dividing n people into two groups where people have preferences for whether they're in the same group as others, and W_{ij} (resp. $-W_{ij}$) is the cost of assigning to the same set (resp. different sets). We're writing the constraint like this rather than $x_i \in \{-1, 1\}$ so that it fits into the duality framework.

This is indeed not a convex problem, since equality constraints must be affine and furthermore x^TWx need not be convex. But we can still say something useful about the problem using duality:

$$\begin{aligned} g(\nu) &= \inf_x \left\{ x^TWx + \sum_i \nu_i(x_i^2 - 1) \right\} \\ &= \inf_x \left\{ x^T(W + \mathrm{diag}(\nu))x \right\} - \mathbf{1}^T\nu \\ &= \begin{cases} -\mathbf{1}^T\nu, & \text{if } W + \mathrm{diag}(\nu) \succeq 0, \\ -\infty & \text{otherwise.} \end{cases} \end{aligned}$$

So we get a lower bound on the original problem of $-\mathbf{1}^T\nu$ as long as $W + \mathrm{diag}(\nu) \succeq 0$. Notice that the dual function is often only finite on some constrained set (which is convex, since g has to be a concave function.)

Remark 126. *Notice that we could rewrite explicit constraints instead as implicit constraints: we could have just asked to minimize $F_0(x)$, which is f_0 but with restrictions for f_i, h_i baked into it. But then there are no Lagrange multipliers to assign, so duality wouldn't give us something useful.*

Proposition 127

Returning now to the comment about conjugate functions from earlier in the lecture, now consider a general minimization of $f_0(x)$ subject to *affine* constraints $Ax \preceq b$, $Cx = d$. The dual function in such a situation then takes the form

$$\begin{aligned} g(\lambda, \nu) &= \inf_{x \in \mathrm{dom}\, f_0} \left\{ f_0(x) + (A^T\lambda + C^T\nu)^Tx - b^T\lambda - d^T\nu \right\} \\ &= -f_0^*(-A^T\lambda - C^T\nu) - b^T\lambda - d^T\nu. \end{aligned}$$

So if the conjugate of f_0 is known, this can simplify our calculations: for example in entropy maximization in information theory, $f_0(x) = \sum_{i=1}^n x_i \log x_i$ has conjugate $f_0^*(y) = \sum_{i=1}^n e^{y_i - 1}$.

We'll now talk about the **Lagrange dual problem** – we've already touched on how the dual gives us lower bounds on the original problem, and because we want the best possible lower bound we care about solving

$$\begin{aligned} \text{maximize} \quad & g(\lambda, \nu) \\ \text{subject to} \quad & \lambda \succeq 0. \end{aligned}$$

This is **always a convex optimization problem** because the constraints are linear and we're maximizing a concave function. The dual optimal value (the best lower bound on p^* obtained from the dual problem) is often denoted d^*.

Unfortunately, when we have a non-convex problem, the dual doesn't tell us that much – it gives us a lower bound, but it might be very bad compared to the actual answer. On the other hand, we get a lot of information if the original problem is convex, and we'll see that soon.

Fact 128

Usually our dual function g will have some implicit constraint because it's $-\infty$ except on some smaller set. It will be best to usually encode those constraints explicitly when setting up our problem.

Example 129

Let's compute the dual problem of the standard form LP in Example 121; we want to maximize $g(\lambda, \nu)$ subject to $\lambda \succeq 0$. Turning the implicit constraint in g into an explicit one, and then eliminating λ (because saying that $A^T\nu - \lambda + c = 0$ and $\lambda \succeq 0$ is the same as just requiring $A^T\nu + c \succeq 0$), the dual problem we wish to solve is

$$\begin{aligned} \text{maximize} \quad & -b^T\nu \\ \text{subject to} \quad & A^T\nu + c \succeq 0. \end{aligned}$$

As a sidenote, if we apply this whole procedure and compute the dual problem of this dual problem, we'll get something equivalent to the original one. This is like how applying the Fourier transform twice gets us back something similar to our original function.

Proposition 130

We have the **weak duality** property $d^* \le p^*$ (the best lower bound is still a lower bound for the original problem).

This always holds for all convex and nonconvex problems, and it can be used to find nontrivial lower bounds like for the two-way partitioning problem in Example 125. Indeed,

$$\begin{aligned} \text{maximize} \quad & -\mathbf{1}^T\nu \\ \text{subject to} \quad & W + \operatorname{diag}(\nu) \succeq 0 \end{aligned}$$

is a semidefinite program whose solution gives an interesting lower bound.

On the other hand, the strong duality property $d^* = p^*$ **does not hold in general**, but it does almost always hold for convex problems. There are some necessary technical conditions that guarantee strong duality in convex problems, and those are typically called **constraint qualifications**. For historical purposes, it's worth spending a bit of time on this:

Proposition 131 (Slater's constraint qualification)
Suppose we have a convex problem

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & Ax = b \end{aligned}$$

which is **strictly feasible**, meaning that some point x in the interior of the domain satisfies $f_i(x) < 0$ for all x. Then strong duality holds, and the dual optimum is attained as long as $p^* > -\infty$.

We can strengthen this inequality further – for example, we can replace "interior" with "relative interior," and affine inequalities don't need to be strictly satisfied. And of course, there are many other types of constraint qualifications as well, but they're typically not that useful. In practice, basically all of our problems aren't "on the brink of being infeasible," so there should be some points that are not right on the boundary of our inequality constraints.

Example 132
Applying Slater's condition to a linear program where we want to minimize $c^T x$ subject to $Ax \preceq b$, the dual problem asks us to minimize $-b^T \lambda$ subject to $A^T \lambda + c = 0$ and $\lambda \succeq 0$. But everything here is an affine inequality, so from the sharpened version we actually get that $p^* = d^*$ if the primal problem is feasible, and in fact $p^* = d^*$ unless both problems are infeasible.

Example 133
Next, suppose we have a quadratic program where we wish to minimize $x^T P x$ (for some positive definite P) subject to $Ax \preceq b$. The dual problem then asks us to maximize a different quadratic, $-\frac{1}{4}\lambda^T A P^{-1} A^T \lambda - b^T \lambda$, subject to $\lambda \succeq 0$. (Perhaps it's not surprising that the dual of a QP is another QP.) Strengthened Slater's condition again automatically applies, so $p^* = d^*$ if the primal problem is feasible; actually $p^* = d^*$ always holds.

In practice, when we use CVXPY, we ask it to solve some original problem, but the solver that it calls solves both the primal and dual problems. So CVXPY automatically gives us the optimal dual variables, which is useful for other purposes (which we will see later on).

10 January 27, 2026 (Problem Session)

The start of today's problem session will be an **introduction to CVXPY** by Brea Swartwood.

Convex optimization software usually consists of two components: **domain-specific languages** describe problems in a "close-to-mathematical" way, and then **solvers** actually compute the solutions to the optimization problems. Each solver expects things in a low-level standard form, which might be different from solver to solver, so there's usually a lot of transformations required to take care of the translation.

The point is that CVXPY uses **disciplined convex programming** to verify problem convexity, since DCP has a modeling grammar that guarantees convexity by construction and thus CVXPY can reject problems that don't follow those rules. (As a sidenote, CVXPY converts things into **conic standard form**

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax + s = b, \\ & s \in K \end{aligned}$$

where K is a convex cone and s is a variable.)

Fact 134

In DCP convex and concave functions are formed as expressions from **variables, constants, and atomic functions**. Atomic functions are basically certain functions with known curvature and monotonicity, and DCP uses constructive convex analysis to track curvature of an expression using the composition rule. More specifically, each atom has a curvature attribute, and it also has a monotonicity attribute **for each argument**.

If we want to look up atoms, we can take a look at the website https://www.cvxpy.org/tutorial/functions/index.html for the meaning, domain, and DCP properties of each of the common atoms. And then we can just use our composition rule to detect that things like `cp.exp(cp.square(x))` are convex (where we've imported `cvxpy` as `cp` and defined `x` as a variable).

However, note that if the composition rule is violated, the curvature will be labeled as "unknown" even if it's equivalent to a convex function. For example, `cp.square(x)` is DCP-compliant but `x * x` is not (the former will return `False` when we call the `is_dcp()` function), and similarly `-cp.entr(x)` is DCP-compliant but `x * cp.log(x)` is not. The point here is that if we have two non-constant expressions, DCP does not recognize their product (since the product of two convex functions is sometimes convex and sometimes not).

Fact 135

The concept of **sign-dependent monotonicity** is important to keep in mind too; for example, `square` is decreasing for $x \le 0$ but increasing for $x \ge 0$. And CVXPY can detect the sign of expressions and use that for the composition rule – for example, the function $f(x, y) = (x^2 + y^2)^2$ can be recognized as convex if we write it as `cp.square(cp.square(x) + cp.square(y))`.

In terms of best practices for CVXPY formulations, it's highly recommended to first write down the problem in math language first and then translate to code (essentially line-by-line). One thing to be careful about is that we have to think about dimensions when initializing vectors, since numpy broadcasts zero-dimensional vectors as row vectors but CVXPY does not. This is because `x = c.Variable(3)` is treated as a row vector of shape `(1, 3)` even if we mean it as a column vector, and thus we should be explicit and use `x = cp.Variable((3, 1))` if that's what we actually mean.

Also, we often might be trying to solve the same problem multiple times but with different data; in such situations it's best to use **parameters**, which are symbolic constants that change value between solves. We can specify the sign of those parameters for DCP analysis, and the advantage is that it can actually be fast for many solves of small problems since we only need to convert to standard form once. (For example, if we want to solve least-squares "lasso regression," meaning we want to minimize $||Ax - b||_2^2 + \lambda ||x||_1$ in the variable x, we can define a set of lambda values and declare lambda as `cp.Parameter(nonneg=True)`. And furthermore, this lets us clean up our code because we don't have to put our entire declaration in the for loop – we just need to update the `.value` of the `Parameter`.

We'll turn back to solving some problems now:

Problem 136

Suppose we have a bunch of continuous functions $f_0, f_1, \cdots, f_n : \mathbb{R} \to \mathbb{R}$, and our goal is to approximate f_0 using a linear combination of $f_1, \cdots, f_n$. For any $x_1, \cdots, x_n$, the approximation to f_0, denoted f, is the linear combination

$$f(t) = x_1 f_1(t) + \cdots + x_n f_n(t),$$

and we allow some ε tolerance of the approximation error, meaning that we want $|f(t) - f_0(t)| \le \varepsilon$ to hold over the entire interval $[0, T]$. Thus for each $(x_1, \cdots, x_n)$, we define

$$W(x) = \sup \{T > 0 : |f(t) - f_0(t)| < \varepsilon \text{ for all } t \in [0, T]\}$$

to be the longest possible interval (largest approximation width) for which f is within tolerance. We want to show that W is quasiconcave.

Our goal is to show that the superlevel sets $S_\alpha = \{x \in \mathbb{R}^n : W(x) \ge \alpha\}$ are all convex. For any $\alpha \ge 0$, the condition that $W(x) \ge \alpha$ is saying that $f(t) = x_1 f_1(t) + \cdots + x_n f_n(t) = a(t)^T x$ (for $a(t) = (f_1(t), \cdots, f_n(t))$ must be close to $f_0(t)$ for all $t \in [0, \alpha]$. Indeed, we can define the **instantaneous constraint set**

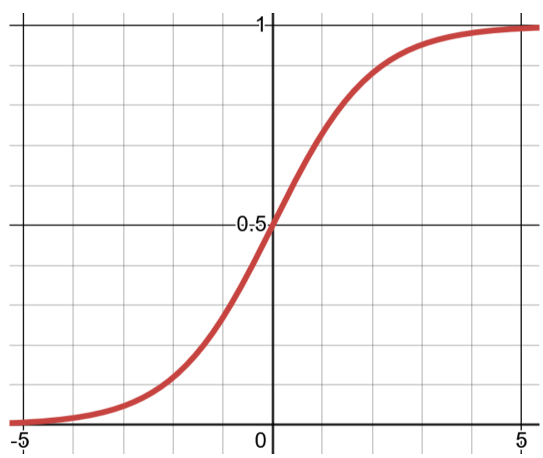
$$C_t = \{x \in \mathbb{R}^n : |a(t)^T x - f_0(t)| \le \varepsilon\}$$

(that is, we only need to be within tolerance at time t). The superlevel set S_α is then the intersection of C_t for all $t \in [0, \alpha]$, since $W(x) \ge \alpha$ must require the constraint inequality to hold up until α (and possibly past that time as well).

But each C_t is convex because it is a slab (it's asking for an affine function of x, $a^T(t)x$, to be between $f_0(t) - \varepsilon$ and $f_0(t) + \varepsilon$). So because the intersection of an arbitrary number of convex sets is convex, S_α is indeed convex, as desired.

Problem 137

Next, we'll show that the logistic function $f(x) = \frac{e^x}{1+e^x}$ is log-concave on $\mathbb{R}$. A graph is shown below (it has horizontal asymptotes at 0 and 1):



There's a whole universe behind the logistic function and the contexts in which it arises (such as population dynamics, neural network classification, and so on) – for example, if $N(t)$ is the population of a species at time t, then we can correct the "dumb model" of exponential growth $\dot{N}(t) = rN(t) \implies N(t) = N_0 e^{rt}$ (which doesn't account for the fact that there might be some maximum possible population that can be supported) by letting the rate of growth

r decay down to zero at some **carrying capacity** C. Approximating this with a straight line then yields

$$\dot{N}(t) = rN(t)\left(1 - \frac{N(t)}{C}\right) \implies N(t) = \frac{Ce^{rt}}{e^{rt} - \left(1 - \frac{C}{N_0}\right)},$$

and the solution to this equation is exactly a scaled version of the logistic curve. (If we want to read more about this, we can take a look at the book "Nonlinear Dynamics and Chaos" by Steven Strogatz.)

Turning back to the actual problem, if we want to show that a function is log-concave, that's saying that we must show that f is strictly positive and that $\log f(x)$ is concave. It's true that f is positive because the exponential function is always positive, and

$$\begin{aligned}\log f(x) &= \log(e^x) - \log(1 + e^x) \\ &= x - \log(1 + e^x).\end{aligned}$$

To show that this is concave, the first term is linear (affine), and we can observe that $\log(1 + e^x)$ is convex because it's $\text{logsumexp}(0, x))$. So since we're subtracting a convex function from an affine function, we indeed have a concave function.

Problem 138

Similarly, we can perform the same analysis for the harmonic mean $f(x) = \frac{1}{\frac{1}{x_1} + \cdots + \frac{1}{x_n}}$ on $\mathbb{R}^n_{++}$.

Indeed, if all x_is are positive, the denominator is positive and thus the whole quantity is positive. Thus we look at $\log f(x)$ and compute that

$$\begin{aligned}\log f(x) &= -\log\left(\frac{1}{x_1} + \cdots + \frac{1}{x_n}\right) \\ &= -\log\left(e^{-\log x_1} + \cdots + e^{-\log x_n}\right).\end{aligned}$$

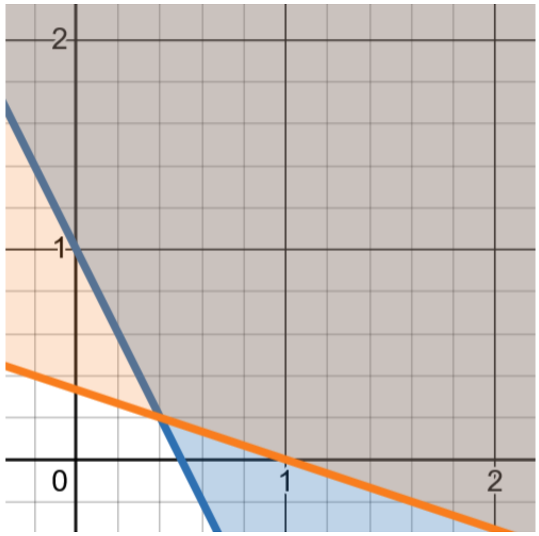
So in fact $\log f$ is the negative of a logsumexp, and logsumexp is increasing in each of its arguments and each $-\log x_i$ is convex in x_i; thus the composition rule tells us $\log f$ is the negative of a convex function, hence concave.

Problem 139

Next, we'll do an explicit example of a convex optimization problem: consider

$$\begin{aligned}\text{minimize} \quad & x_1 + x_2 \\ \text{subject to} \quad & 2x_1 + x_2 \geq 1, \\ & x_1 + 3x_2 \geq 1, \\ & x_1 \geq 0, \\ & x_2 \geq 0.\end{aligned}$$

The feasible set here is a subset of the first quadrant above two specified lines, yielding the common shaded region shown below (restricted to the first quadrant):



If our goal is to minimize $x_1 + x_2$, we can imagine levelsets $x_1 + x_2 = c$ (which are all parallel lines of slope -1). Our goal is to minimize the function, so we should pick the levelset with smallest c that still intersects the set. In this case, the optimal point will be the intersection of the two marked lines in the diagram.

We can imagine variants of this problem with different lines. But if we wanted to minimize $-x_1 - x_2$ instead, then the problem is unbounded, and if we wanted to minimize x_1, then any point on the y-axis is in the feasible set. (And more generally, the functions we want to minimize will be more complicated, but we can geometrically think of them as decreasing c in the levelsets until we are just barely tangent to our feasible set.)

11 February 3, 2026

We'll almost finish the "theory" part of the course today and move to things that will pull together what we've learned so far.

What we've been discussing in the last few classes is that duality is an **automated way to create parameterized lower bounds** on the optimal value of our optimization problem. These parameters, called Lagrange multipliers or dual variables, can be tuned, and we're often curious what the best possible lower bound for p^* is. This "dual optimization" problem is always convex and so we can solve for d^*, and the important thing is that $d^* \leq p^*$. Often in convex problems we actually have equality, and even if not we still have a way to tell how far off our answer could possibly be.

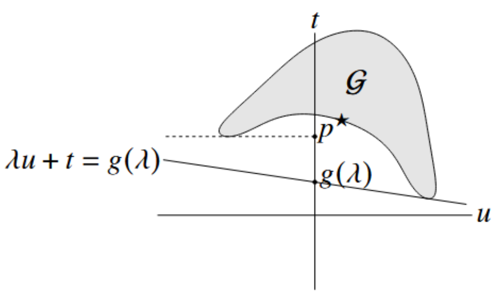
Remark 140. *Notice that the problem "minimize f_0 subject to $f_1 \leq 0$" is essentially asking us to minimize $f_0(x) + I_+(f_1(x))$, where I_+ is the indicator function which returns $+\infty$ if its argument is positive and 0 otherwise. And so it's somehow interesting that the right variational thing to do is to try to minimize $f_0(x) + \lambda f_1(x)$ instead for a positive number λ.*

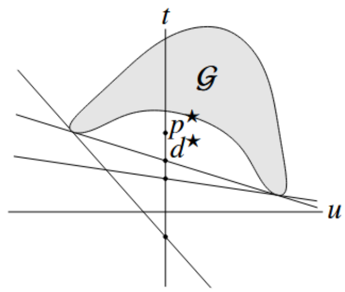
We also mentioned **Slater's constraint qualification** last time, which is a condition which guarantees $d^* = p^*$ for convex problems as long as all of the non-affine constraints can be simultaneously strictly satisfied. And it's important to remember that in all practical problems, strong duality does indeed hold.

Remark 141. *After we do a few of these problems, we start getting used to the form of the answer we'll get and we start seeing the connections to conjugates. For example, the dual of a linear program is another linear program, and same with quadratic programs, with the exact same data rearranged in some way.*

Fact 142

For a geometric interpretation, suppose we have a single constraint f_1 and we want to think about the set of possible constraint-objective pairs $\mathcal{G} = \{f_1(x), f_0(x) : x \in \mathcal{D}\}$. If we plot the set $\mathcal{G}$ in a two-dimensional plane (with f_1 on the horizontal axis and f_0 on the vertical axis), we're basically asking for the lowest possible point on the left half of the plane (because the right half is infeasible).





Looking at the left image here, the level curves of the Lagrangian are lines with a downward slope, and so for any given λ we can visualize $g(\lambda)$ as follows: pick the lowest such line of a fixed slope which still intersects $\mathcal{G}$. Then $g(\lambda)$ is the y-intercept we get for that line, because by definition

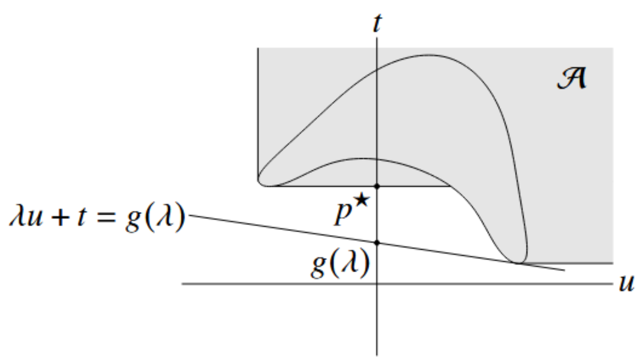
$$g(\lambda) = \inf_{(u,t)\in\mathcal{G}} \{t + \lambda u\}$$

and the quantity $t + \lambda u$ is constant on such lines. But notice that because this set $\mathcal{G}$ is not convex (more precisely, because the epigraph of it is not), the best possible $g(\lambda)$ we can get over all slopes will be worse than the optimal value p^*. If our set were convex, the supporting hyperplane theorem actually tells us that (assuming constraint qualifications hold) we can actually get $g(\lambda)$ to achieve that optimal value.

We can also do the same thing with the epigraph variation of the problem where we consider

$$\mathcal{A} = \{(u, t) : f_1(x) \leq u, f_0(x) \leq t \text{ for } x \in \mathcal{D}\}$$

In this perspective, strong duality holds if we have some non-vertical supporting hyperplane to $\mathcal{A}$ at $(0, p^*)$. And Slater's condition tells us that if there's a point on the left side here, then the supporting hyperplane is non-vertical and thus strong duality does make sense.



We'll next talk about **optimality conditions** for convex problems. If we assume that strong duality holds, and x^*

is optimal for the original problem and (λ^*, ν^*) for the dual problem, then

$$\begin{aligned} f_0(x^*) &= g(\lambda^*, \nu^*) \\ &= \inf_x \left\{ f_0(x) + \sum_{i=1}^m \lambda_i f_i^*(x) + \sum_{i=1}^p \nu_i^* h(x) \right\} \\ &\le f_0(x^*) + \sum_{i=1}^m \lambda_i^* f_i(x^*) + \sum_{i=1}^p \nu_i^* h_i(x^*) \\ &\le f_0(x^*), \end{aligned}$$

where the first inequality just comes from noticing that x^* is a particular value, so plugging it into the expression infimum must be at least as big as the actual infimum; the second inequality is then coming from $f_i(x^*) \le 0, h_i(x^*) = 0$ and $\lambda_i^* \ge 0$. But this is a chain of inequalities which is actually an equality, so everything is equality and thus x^* is actually the minimum for $L(x, \lambda^*, \nu^*)$. This means that our second inequality actually becomes **complementary slackness**: we have $\lambda_i^* f_i^* = 0$, meaning that

$$\lambda_i^* > 0 \implies f_i(x^*) = 0 \quad \text{and} \quad f_i(x^*) < 0 \implies \lambda_i^* = 0.$$

Thinking about things mechanically, it might not be surprising that when we minimize the potential energy of a configuration, a "tight cable" at its limit means there will be tension, and in fact the Lagrange multipliers are exactly the tensions. (And in a robotics problem, if we have a robot holding an object then the λs give us the contact forces.)

Proposition 143 (Karush-Kuhn-Tucker (KKT) conditions)

Suppose f_i, h_i are differentiable. Then the KKT conditions are the following:

1. (Primal constraints) $f_i(x) \le 0$ for all i and $h_j(x) = 0$ for all j.
2. (Dual constraints) The Lagrange multipliers satisfy $\lambda \succeq 0$.
3. (Complementary slackness) $\lambda_i f_i(x) = 0$ for all i.
4. The gradient of the Lagrangian with respect to x vanishes, meaning that

$$\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) + \sum_{i=1}^p \nu_i \nabla h_i(x) = 0.$$

(This is exactly what we do when solving constrained Lagrange multiplier problems in calculus.)

If strong duality holds, the KKT conditions are sufficient for optimality, since they guarantee that $f_0(x) = g(\lambda, \nu)$. That is, if x, λ, ν satisfy KKT, then they must be optimal. And it turns out that **if Slater's condition is satisfied, a converse is true**: x is optimal if and only if there are λ, ν that satisfy the KKT conditions.

So far, we've only seen one practical use for duality, which is finding some lower bound or certifying optimality. But there's actually something more useful, which is to consider perturbation or sensitivity analysis.

Example 144

Consider the perturbed problem

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \le u_i \text{ for } 1 \le i \le m, \\ & h_i(x) = v_i \text{ for } 1 \le i \le p. \end{aligned}$$

The original problem just had all u_i, v_is equal to zero, and this is a shift of that. If $u_i < 0$, we say that we 'tighten the constraint," and if $u_i > 0$, we say that we "loosen the constraint." Sometimes we can interpret the v_is as well (for example if the constraint is some conservation of power at a node in a grid, then perhaps there is some load there that's not part of our system).

The dual of this problem also shifts in an easily describable way: we go from solving

$$\begin{aligned} \text{maximize} \quad & g(\lambda, \nu) \\ \text{subject to} \quad & \lambda \succeq 0 \end{aligned}$$

to instead solving (because the Lagrangian gains an additional factor of $\sum_i \lambda_i(-u_i) + \sum_i \nu_i(-v_i)$, which doesn't depend on x)

$$\begin{aligned} \text{maximize} \quad & g(\lambda, \nu) - u^T \lambda - v^T \nu \\ \text{subject to} \quad & \lambda \succeq 0. \end{aligned}$$

Letting $p^*(u, v)$ denote the optimal value of the original problem as a function of u, v, this is a partial minimization of a convex function in x, u, v over x (specifically, $f_0(x) + I_+(f_i(x) - u_i) + I_0(h_i(x) - v_i)$ where I_0 is $+\infty$ unless its argument is zero), hence convex. So by weak duality for this perturbed problem at $(0, 0)$, we get that (basically we can use any λ^*, ν^* as a lower bound for our perturbed problem, but we're going to use the optimal dual variables from the original one)

$$\begin{aligned} p^*(u, v) &\ge g(\lambda^*, \nu^*) - u^T \lambda^* - v^T \nu^* \\ &= p^*(0, 0) - u^T \lambda^* - v^T \nu^*. \end{aligned}$$

That means the slope for our optimal Lagrange multiplier λ_i^* tells us a **lower bound for how p^* will change** via a Taylor expansion: if we loosen constraint i and λ_i^* is small, then p^* won't decrease much. But also keep in mind that all of these things are asymmetric in the sign: for example, a large Lagrange multiplier implies that tightening the corresponding constraint **guarantees** that the objective goes up a lot, but we can't guarantee that loosening it will make the objective go down a lot.

However, that's a global statement, and we can say things pretty accurately locally because Taylor approximations are good near their center:

Proposition 145 (Local sensitivity)

Suppose in addition that $p^*(u, v)$ is differentiable at $(0, 0)$. Then

$$\lambda_i^* = -\frac{\partial p^*(0,0)}{\partial u_i}, \quad \nu_i^* = -\frac{\partial p^*(0,0)}{\partial v_i}.$$

That is, suppose we minimize the cost of production in an energy grid, and our constraints require that the sum of the powers in and out of a distribution point is zero. Then the corresponding Lagrange multipliers are called **locational marginal prices** – it tells us what approximately happens to the cost when we have to re-solve the **whole problem** again if we just suck out a megawatt from a particular point. So it's not that surprising that these dual variables come up in economics all the time!

Remark 146. *Notice that this also gives us a sense in which "some inequality constraints are tighter than others" even if it doesn't make sense from a margin sense: Lagrange multipliers tell us how much varying each of those constraints would change the optimal value, and the higher rate is the "tighter one" in some sense.*

We'll now mention some things about how we can reformulate problems and how that affects duality. CVXPY relies on these kinds of transformations, and the point is that they can be useful when the dual is difficult to derive because even **equivalent formulations end up with very different duals** – if $\mathcal{P}$ and $\tilde{\mathcal{P}}$ are equivalent, and $\mathcal{D}$ is the dual of $\mathcal{P}$ and $\tilde{\mathcal{D}}$ is the dual of $\tilde{\mathcal{P}}$, it's not true that $\mathcal{D}$ and $\tilde{\mathcal{D}}$ are always going to be equivalent.

Here's one thing we can do:

- Start with an unconstrained problem where we just minimize a single objective $f_0(Ax + b)$. (In this case, the dual function is actually a constant function, since the Lagrangian is just f_0 and thus g is always p^*.) So we have strong duality but the dual problem is completely useless.
- Now un-eliminate constraints by introducing a new variable y and setting up the constraint $y = Ax + b$, so now we want to solve

$$\begin{aligned} \text{minimize} \quad & f_0(y) \\ \text{subject to} \quad & Ax + b - y = 0. \end{aligned}$$

  The dual of this reformulated problem is now interesting: it's asking

$$\begin{aligned} \text{maximize} \quad & b^T\nu - f_0^*(\nu) \\ \text{subject to} \quad & A^T\nu = 0. \end{aligned}$$

  So if the conjugate f_0^* is easy to express, this is actually a nontrivial useful dual.

Example 147

Suppose we have a general norm and want to minimize $||Ax - b||$ for some A, b. The conjugate of a norm is 0 inside the unit ball of the dual norm and ∞ otherwise, so the dual of this reformulated problem is (by moving the ∞ around into the constraint)

$$\begin{aligned} \text{maximize} \quad & b^T\nu \\ \text{subject to} \quad & A^T\nu = 0, \\ & ||\nu||_* \leq 1. \end{aligned}$$

Finally, we'll discuss **theorems of alternatives**, which is a way to apply duality to feasibility problems (which are to "minimize 0 subject to the constraints," which either outputs 0 or $+\infty$).

Definition 148

Consider two systems of inequality and equality constraints. If they can't both be feasible, call them **weak alternatives**, and if exactly one of them is always feasible, call them **strong alternatives**.

For example $x > a$ and $x \le a$ are strong alternatives, and $x > a$ and $x \le a - 1$ are weak alternatives. A **theorem of alternatives** is a statement that two inequality systems are indeed weak or strong, and we can think of this as providing a version of duality for feasibility because we essentially care about whether certain quantities are positive or not.

We'll start next time by seeing what happens when we take the dual of a feasibility problem and how to interpret things in this context! It will turn out that feasibility of an inequality system and its dual are actually alternatives.

12 February 3, 2026 (Problem Session)

We'll have a CVXPY component led by one of the TAs today, Nika Zahedi, to discuss some problems, more common mistakes and best practices.

Problem 149

Consider a simple numerical perturbation analysis, which is one way in which we can use dual variables. Suppose we want to minimize the quadratic function $x_1^2+2x_2^2-x_1x_2-x_1$ subject to linear constraints $x_1+2x_2 \le u_1$, $x_1-4x_2 \le u_2$, $5x_1 + 76x_2 \le 1$.

The first thing we should do is put this in **matrix form**: we write the objective function as $x^TQx + f^Tx$, where the terms of degree 2 end up in the x^TQx part and the linear term ends up in the f^Tx part, so that $Q = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 2 \end{bmatrix}$ (we want this matrix to be symmetric, which is why we split the $-x_1x_2$ across the two off-diagonal entries) and $f = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$. Similarly, the constraints can also be encoded as a single vector identity $Ax - b \le 0$, where $A = \begin{bmatrix} 1 & 2 \\ 1 & -4 \\ 5 & 76 \end{bmatrix}$ and $b = \begin{bmatrix} u_1 \\ u_2 \\ 1 \end{bmatrix}$.

Once we do this, there isn't much we need to do to convert to CVXPY code: we've reduced our problem to a quadratic objective with a linear constraint, so all that's left is to ask CVXPY to give us the optimal points. After writing out the objective and constraint, we essentially just need to set up the problem `cp.Problem(cp.Minimize(cp.quad_form(x, Q) + f @ x), [A @ x <= b])`. After we call `.solve` on this problem, we can get the `values` of our variables, as well as the `.dual_values`. (Notice that there should be 2 values of our variables, but 3 values of the dual variables, since there are three constraints.)

If we want to check the KKT conditions now, we need four things to hold: **primal feasibility** (the optimal point satisfies all of the inequality and equality constraints), **dual feasibility** (all of the λ_is are nonnegative) **complementary slackness** (either $f_i = 0$ or $\lambda_i = 0$), and **the gradient of the Lagrangian vanishes**. All of these conditions can be checked here: we find in this case with a particular choice of u_1, u_2 that $Ax - b$ is actually exactly the zero vector up to rounding errors, so in fact complementary slackness is automatically satisfied. For the gradient, we have to do a bit more computation:

$$L(x, \lambda) = x^T Q x + f^T x + \lambda^T (Ax - b) \implies \nabla_x L = 2Qx + f + A^T \lambda,$$

where we use that $\lambda^T(Ax) = x^T A^T \lambda$. (If Q were not symmetric we would have $(Q + Q^T)$ instead of $2Q$, so we should be careful.) And plugging in our optimal x and λ indeed shows that $\nabla_x L$ is the zero vector up to numerical errors.

Next, if we perform perturbation analysis by adjusting the vector values u_1, u_2 by a small change δ, recall that we estimate an optimal value in terms of the unperturbed problem

$$p^*_{\text{pred}} = p^* - \lambda^T \delta,$$

and also that this estimate is **always a lower bound** (this is the "global sensitivity" concept). So now we can just compare p^*_{pred} to the true new p^* when we set up the (slightly) different constraints, and we see that we always get a lower bound but that sometimes it is not very tight.

Problem 150

Taking a more practical problem as an application now, suppose we want to think about tradeoffs of energy storage where we charge a battery to save in terms of varying electricity costs. We have time periods 1 through T, in which we have electricity prices p_t (positive) and desired consumption u_t (nonnegative). We then have the problem variables that we get to control: we have units of energy stored in the battery q_t (always nonnegative), and amount of charging we do c_t (no condition on signs).

By energy conservation we have $q_{t+1} = q_t + c_t$, and we also require that we finish with the same battery charge as we began with so that $q_1 = q_T + c_T$. But there are also some extra constraints to think about: the net consumption in each period has to be nonnegative, so $u_t + c_t$ should be nonnegative for all t. And we have a maximum battery capacity Q (so $q_t \le Q$ for all t) and a maximum battery charge or discharge rate, meaning $-D \le c_t \le C$.

Our goal is then to minimize the total cost $p^T(u + c)$. But now that we've written everything out, CVXPY can solve for the optimal vectors q and c if we encode our objective and constraints properly. The main thing to keep in mind is to initialize the variables as `cp.Variable(shape = (T, 1))` rather than `cp.Variable(shape = T)`, and also remember that constraints like $q_{t+1} = q_t + c_t$ can be encoded in matrix form by taking subarrays and saying that `q[1:] == q[:T-1] + c[:T-1]`.

In a situation like this, we have dual variables for each c_t and each q_t. It will turn out that there are periods of time when we are charging at the maximum allowed rate, and since there is zero slack in the constraint there it is possible for the dual variables to be nonzero. But in all other cases where we aren't at the limit, we will have $\lambda_t = 0$. And in this case, because the function we are trying to optimize is essentially $p^T c$ (plus a constant not in our control), the **dual variables corresponding to the charge conservation constraints will essentially trace out the prices**. Indeed, the Lagrangian for this problem is

$$\begin{aligned} L = p^T c + \lambda_1^T(-c - D\mathbf{1}) + \lambda_2^T(c - C\mathbf{1}) + \lambda_3^T(-q) + \lambda_4^T(q - Q\mathbf{1}) \\ + \lambda_5^T(-u - c) + \sum_{i=1}^{T-1} \nu_i (q_{i+1} - q_i - c_i) + \nu_T(q_1 - q_T - C_T), \end{aligned}$$

and we're interested in the the dual variables for the last constraints, which are the ν_is. The KKT conditions tell us that $\nabla_{c,q} L = 0$: for c this gives us

$$0 = \nabla_c L = p - \lambda_1 + \lambda_2 - \lambda_5 - \nu.$$

Therefore our dual variables should actually be $\nu = p - \lambda_1 + \lambda_2 - \lambda_5$, and these additional factors are zero if we're not at capacity (that is, we'll be exactly equal to the price unless we're at the max charge or discharge rate, or if we have zero power consumption). And the reason this derivative takes such a nice form is basically because all of our functions of interest in this problem are affine.

13 February 5, 2026

We'll begin by finishing up our discussion of duality by thinking about feasibility inequalities and theorems of alternatives ("this is feasible if and only if this other thing is not"). If we apply what we've thought before to the minimization problem

$$\begin{aligned} \text{minimize} \quad & 0 \\ \text{subject to} \quad & f_i(x) \le 0 \text{ for } 1 \le i \le m, \\ & h_i(x) = 0 \text{ for } 1 \le i \le p, \end{aligned}$$

which has either solution $p^* = 0$ if feasible or $p^* = \infty$ if not, then Lagrange duality yields the dual function

$$g(\lambda, \nu) = \inf_x \left\{ \sum_{i=1}^m \lambda_i f_i(x) + \sum_{i=1}^p \nu_i h_i(x) \right\},$$

and this is actually "positive homogeneous" (that is, linear when λ, ν are positive). Now we know that $g(\lambda, \nu) \le p^*$ by weak duality. But this means is that if the system

$$\lambda \succeq 0 \quad \text{and} \quad g(\lambda, \nu) > 0$$

is feasible, then the original inequality system must be infeasible because p^* must be ∞. This means we have a weak alternative to the original system (and in fact it is strong if f_i are convex and h_i are affine, and we have a constraint qualification). Note that in basically all practical applications, strict inequalities don't actually make sense due to rounding errors, and in this particular case we can actually write the alternative system as

$$\lambda \succeq 0 \quad \text{and} \quad g(\lambda, \nu) \ge 1$$

by positive homogeneity.

Example 151

Consider the system $Ax = b, x \succeq 0$ (which is asking if the solution to the linear system intersects the positive orthant). The corresponding dual function here is

$$g(\lambda, \nu) = \begin{cases} -b^T \nu, & \text{if } A^T \nu = \lambda, \\ \infty, & \text{otherwise.} \end{cases}$$

Thus we have a (strong) alternative to the original system, which is $A^T \nu \succeq 0, b^T \nu \le -1$ (where again by homogeneity we have replaced $-b^T \nu > 0$ by $b^T \nu \le -1$).

Proposition 152 (Farkas' lemma)
The two inequality systems

$$\left\{Ax \preceq 0 \text{ and } c^T x < 0\right\}, \quad \left\{A^T y + c = 0 \text{ and } y \succeq 0\right\}$$

are strong alternatives. This can be shown by using strong duality for the feasible LP "minimize $c^T x$ subject to $Ax \preceq 0$," and this is used for the basic fundamental idea of **arbitrage** in economics.

To elaborate a bit more on this, suppose we have n assets with prices $p_1, \cdots, p_n$, and we invest x_i in the ith one so that our initial cost is $p^T x$. Suppose there are a finite number m of total final outcomes (like in a horse race), and V is a matrix where V_{ij} is the payoff (final value) of asset j in outcome i. Assuming that the first asset is just cash, so it is risk-free, we have $p_1 = 1$ and $V_{i1} = 1$ for all i (so no matter what the outcome is, nothing happens to our money).

The concept of **arbitrage** is that there is some x with $p^T x < 0$ (meaning we get money when we invest at the beginning), but $Vx \succeq 0$ (so no matter what happens, we cannot lose money). And in economics the standard assumption is that prices will be set so that arbitrage is not possible.

So by Farkas' lemma, saying that there is no arbitrage is the same thing as saying that there is some $y \in \mathbb{R}^m_+$ with $V^T y = p$. The first column of V is corresponding to the risk-free asset, so it is just $\mathbf{1}$, and therefore $\mathbf{1}^T y = 1$; that is, y can be interpreted as a **risk-neutral probability** on the set of possible outcomes. Then $V^T y = p$ basically means that "in the absence of arbitrage, asset prices equal their expected payoff under some risk-neutral probability," and the "risk-neutral" part means that if we don't care about risk (as opposed to being risk-averse or risk-loving), the expected profit for each asset is at most the price we initially put into it.

Remark 153. *This is already kind of cool and weird: we had no description of probability or which outcomes are more likely than others, but out of these duality calculations we automatically have a distribution of outcomes coming from prices. And that's essentially the fundamental idea of prediction markets – we can assume the pricing goes towards some arbitrage-free thing and then just solve an LP. And this happened roughly because "the conjugate of the max function is the indicator function of the unit simplex."*

Example 154
Suppose $V = \begin{bmatrix} 1 & 0.5 & 0 \\ 1 & 0.8 & 0 \\ 1 & 1 & 1 \\ 1 & 1.3 & 4 \end{bmatrix}$. This says that the first asset is just cash, the second has varying outcomes, and the third is more like a horse race, and there are four different possible final outcomes.

With prices $p = \begin{bmatrix} 1 \\ 0.9 \\ 0.3 \end{bmatrix}$, there is indeed an arbitrage: choosing $x = \begin{bmatrix} 6.2 \\ -7.7 \\ 1.5 \end{bmatrix}$ would give us a negative initial cost, but then the matrix Vx has all nonnegative entries, meaning that we can't lose money at all. On the other hand, the prices $p = \begin{bmatrix} 1 \\ 0.8 \\ 0.7 \end{bmatrix}$ yield a risk-neutral probability vector $y = \begin{bmatrix} 0.36 \\ 0.27 \\ 0.26 \\ 0.11 \end{bmatrix}$, which is the "probabilities of the outcomes that the prices are telling us to expect." Indeed, $V^T y$ in this case is exactly the price vector.

With that, we're now ready to start the part of the course which is **just applications and nothing more**. We'll organize into various topics and fields, but the point is just to go through problems and look at why they will be useful. (We'll go through simple examples in class and also do some in our homework.)

The point as always is that "if we write something as a convex problem, it is completely tractable," and we can always add additional (convex) constraints and so on if we want to modify the problem.

Fact 155

We'll begin with **approximation**. A generic approximation problem asks us to minimize the "residuals" $||Ax - b||$, where A is a matrix giving us predictions, b is some observed data, $m \geq n$, and $|| \cdot ||$ is any generic norm.

This has lots of names and interpretations: for example, solving this is giving us the best approximation of the vector b by linear combinations of the columns of A, or alternatively it's the point in a subspace closest to b, or it's the design x^* that best approximates our desired result, or it's the most plausible point x^* in a linear measurement or physics model with noise or error that leads to our final measurement b.

In some examples, like Euclidean approximation, the solution is exact and pretty simple. But if we instead use $|| \cdot ||_\infty$ (Chebyshev approximation) or $|| \cdot ||_1$ (sum-of-absolute residuals approximation), the solution is basically just an LP. So they're all kind of the same in spirit, but we're measuring closeness in different ways and that's quite important. In mathematical terms "all norms are equivalent in $\mathbb{R}^n$" so it doesn't matter in topology where we just care about the norm going to zero, but for applications it does actually matter.

Example 156

We'll generalize those examples a little bit and now consider a problem where we minimize a function of the residuals: let $\phi : \mathbb{R} \to \mathbb{R}$ be a **convex penalty function** and define

$$\begin{aligned} \text{minimize} \quad & \phi(r_1) + \cdots + \phi(r_m) \\ \text{subject to} \quad & r = Ax - b. \end{aligned}$$

This gives us the Euclidean norm for $\phi(u) = u^2$, but the point is just that we can have various choices of 'how much penalties matter to us;" in the absolute-value ℓ^1 case with $\phi(u) = |u|$, small residuals irritate us much more than in the quadratic one.

Two other interesting (useful) examples are the **deadzone-linear function** with width a

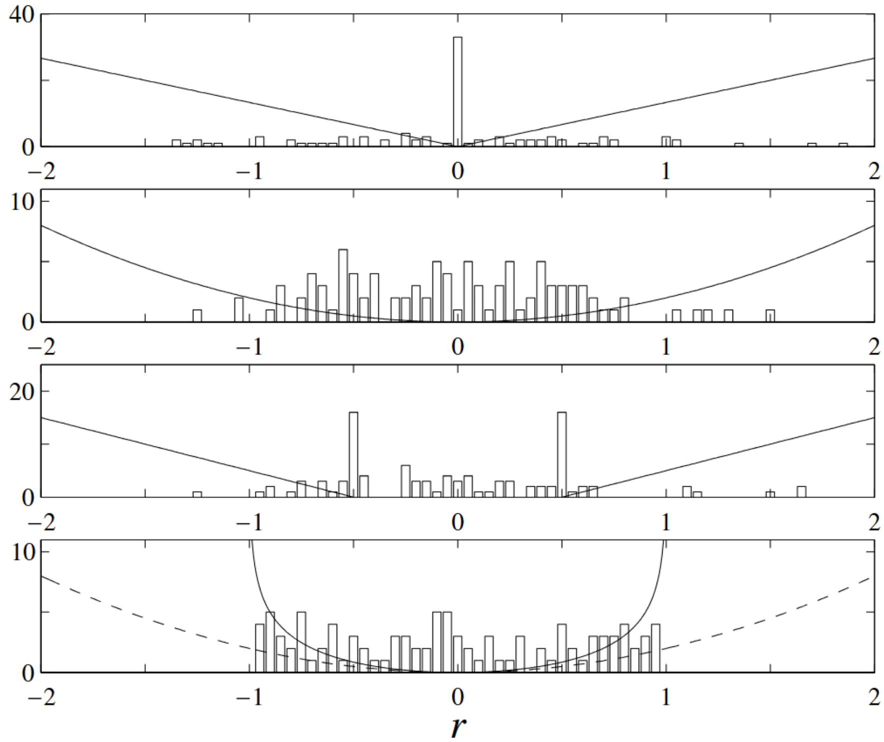
$$\phi(u) = \max\{0, |u| - a\},$$

which means that we don't care about a small residual at all and then care linearly outside of that, and the **log-barrier** with limit a

$$\phi(u) = \begin{cases} -a^2 \log\left(1 - \left(\frac{u}{a}\right)^2\right), & \text{if } |u| < a, \\ \infty, & \text{otherwise,} \end{cases}$$

which is like quadratic for small residuals but then really does not allow residuals bigger than size a. (And then we can have things like the **pinball loss**, which biases towards overestimating or underestimating more than the other one.)

The shape of the penalty function will affect the distribution of residuals. For a particular 100×30 matrix, below are the residuals we get for each of the penalties absolute-value, square, deadzone, and log-barrier after choosing the optimal x with respect to that function:

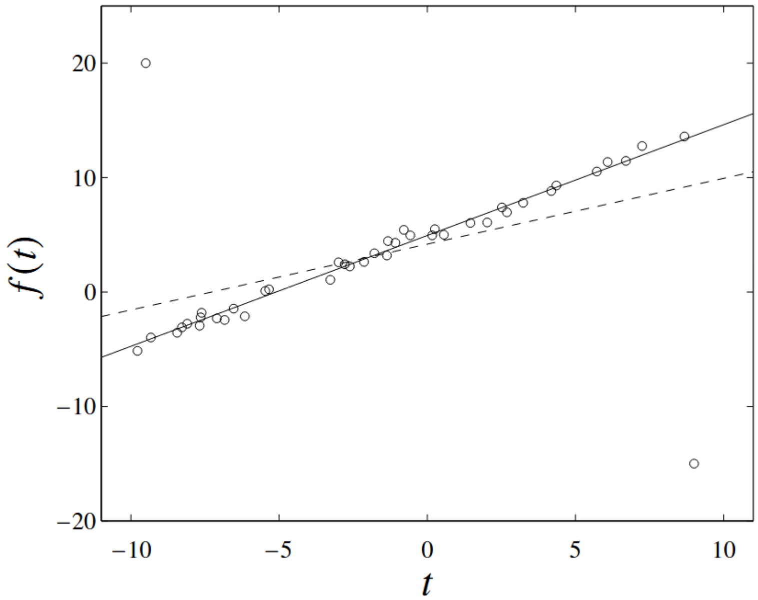


In the first case many of the residuals are exactly zero (this is like saying ℓ^1 "sparsifies"). In the second case this does not happen because "it doesn't really matter to make small residuals smaller," but in contrast notice that the largest residuals are now smaller. The third case now has a lot of residuals exactly bunched up at $-a$ and a (because that's the point at which we stop caring about making the residuals any smaller), and finally the fourth case has no residuals outside the barrier at all. So even though they're all making Ax close to b, there are still qualitative differences. The same interpretation will come up in some statistical applications later.

One other famous function is the **Huber penalty function**

$$\phi_{\text{Hub}}(u) = \begin{cases} u^2, & \text{if } |u| \leq M, \\ M(2|u| - M), & \text{if } |u| > M. \end{cases}$$

The idea is that we care about residuals like least-squares, but it's also pretty chill if there are some big ones. This is called a **robust penalty** because the approximation will not be quite as sensitive to outliers as least-squares. (There's also a "BerHu" penalty where the roles of ℓ^1 and ℓ^2 are swapped, but this is pretty silly.) For example, if we have 2 out of 42 points in a dataset that are extreme outliers as shown below, the dashed line shows what least-squares gives us, and the solid line shows the Huber penalty line.



Mechanically, we can imagine connecting a vertical spring between each observed data point and the line; ordinary strings have quadratic potential energy and so the dashed-line is what we get when we minimize the energy of the configuration. And the Huber function is like a spring whose force stops going up like Hooke's law after some point. But it's a sign that maybe Huber should be used instead of least-squares in applications where some fraction of the data points are just wrong.

Example 157

Now consider a **least-norm problem**

$$\begin{aligned} \text{minimize} \quad & ||x|| \\ \text{subject to} \quad & Ax = b, \end{aligned}$$

which has the geometric interpretation that we have an affine solution set and want to find the smallest-norm solution.

For some interpretations of this, we can assume we have perfect measurements of x via Ax but don't actually have x itself; if the norm represents implausibility (for example in tomography) then we are choosing the most plausible estimate. Alternatively, perhaps we have some design variables x as inputs (e.g. rocket thruster lengths) and some required results b; we can thus use this to find the most efficient design that satisfies our requirements.

Again least-Euclidean-norm has a closed-form analytical solution, and the least sum of absolute values can be solved via an LP. And because of the discussion we had before, the latter tends to have sparse x^*s because we continue to have equal incentives to make the errors down all the way down to zero.

Example 158

In most problems we actually have a bi-objective: we actually want to minimize $||Ax - b||$ and $||x||$ simultaneously, and this leads us to the concept of **regularized approximation**. The interpretation is that we want to find a good approximation also with small x.

For example in a linear measurement model $y = Ax + v$, perhaps our model is only accurate for small x because it's linearized and not the real truth. So in those cases we don't want to minimize $||Ax - b||$ with a really big x where the linear model is not accurate, we might just be kidding ourselves. So the second term can be thought of as a "trust penalty." And we'll also see the concept of **robust approximation** many times throughout this class: sometimes A isn't what we thought it was, and a good approximation with small x is actually less sensitive to errors in A than a good approximation with large x.

We can turn this into a scalarized problem, as we discussed earlier in the course: we get an optimal tradeoff curve γ while minimizing $||Ax - b|| + \gamma||x||$ over $\gamma > 0$. It's also common to minimize $||Ax - b||^2 + \delta||x||^2$, which turns out to trace out the same tradeoff curve. For the 2-norm this is actually called **Tikhonov regularization** or **ridge regression** (we're minimizing the sum of the squared residuals, plus some regularization term that asks our parameters to be small); this can then be solved with an exact analytical solution.

Example 159

Suppose we have a linear dynamical (convolution) system with impulse-response (convolution kernel) h, meaning that

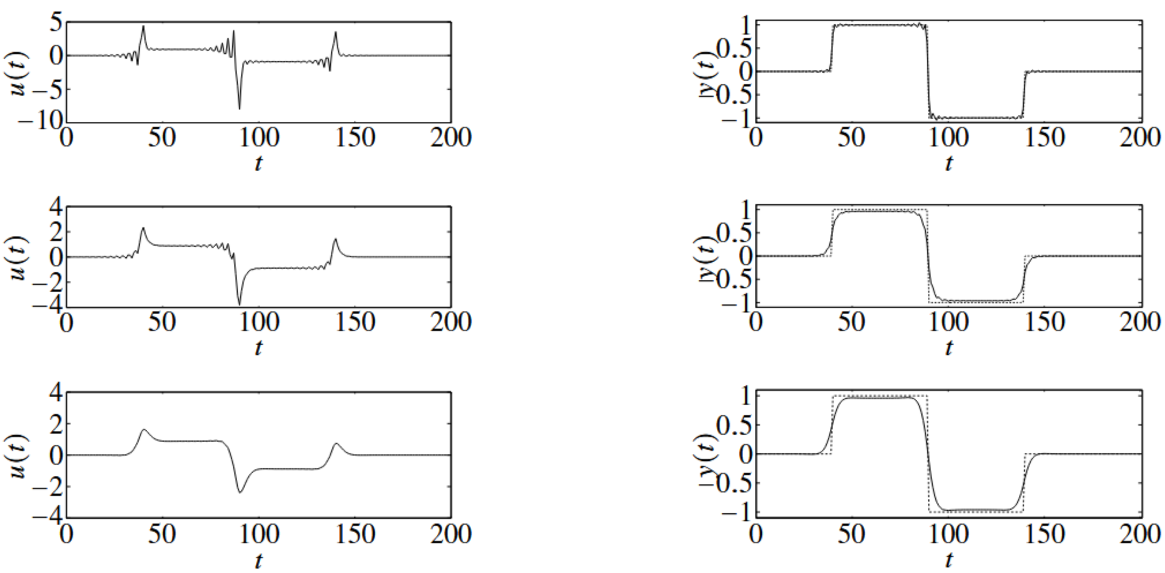
$$y(t) = \sum_{\tau=0}^{t} h(\tau)u(t - \tau).$$

For example, u might be heat we put into a building and y gives us the temperature in some region inside. Then the input design problem is a multi-criterion problem with three objectives:

- We have some desired y_{des} which may not be realistic, and we want to minimize the tracking error $J_{\text{track}} = \sum_{t=0}^{N}(y(t) - y_{\text{des}}(t))^2$ of what we actually achieve.
- We want to have minimal input variation, meaning we want to reduce $J_{\text{der}} = \sum_{t=0}^{N-1}(u(t+1) - u(t))^2$.
- We also want minimal input magnitude, meaning we want to reduce $J_{\text{mag}} = \sum_{t=0}^{N} u(t)^2$.

We can formulate this as a regularized least-squares formulation by minimizing $J_{\text{track}} + \delta J_{\text{der}} + \eta J_{\text{mag}}$.

The six plots below show the resulting inputs $u(t)$ and outputs $y(t)$ that we get out of certain choices. The first row here is $\delta = 0$ and small η; the next is $\delta = 0$ and larger η; the last is large δ. Notice that the first two cases don't penalize "wiggly-ness," and the last two do penalize having very large us because of the magnitude penalty.

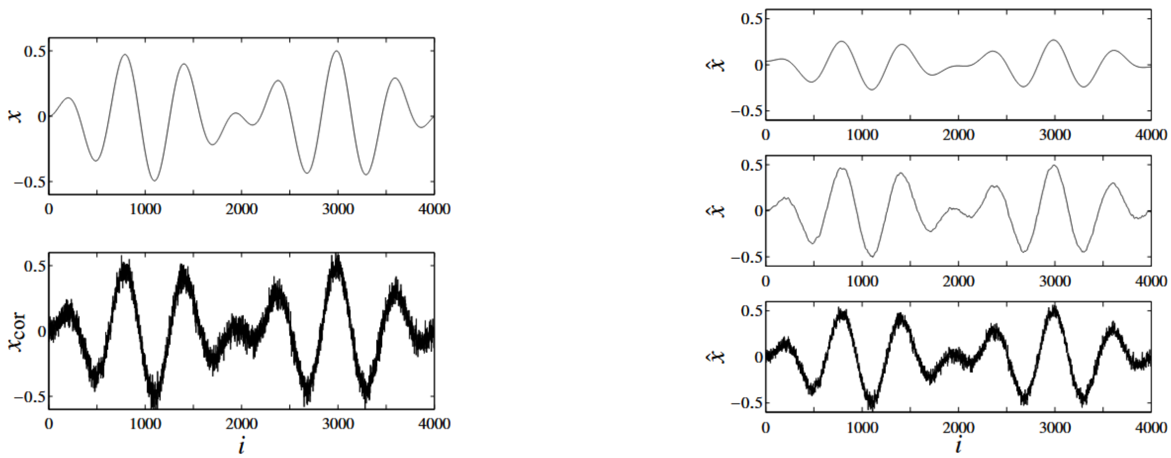


The way we choose δ, η is basically to guess and then fiddle with them until we see something we like. (And sometimes the answer is "cross-validation" in some fields, but honestly in most engineering it's just to guess and adjust.)

Example 160

In **signal reconstruction**, we have some unknown signal x and we read a corrupted noisy version $x_{\text{cor}} = x + v$. We thus want to minimize both $||\hat{x} - x_{\text{cor}}||_2$ and some smoothing objective (regularization function) $\phi(\hat{x})$, using perhaps quadratic $\sum_i (\hat{x}_{i+1} - \hat{x}_i)^2$ or total variation smoothing $\sum_i |\hat{x}_{i+1} - \hat{x}_i|$.

On the left, the original signal and the noisy one are shown, and on the right we see three different solutions on the tradeoff curve between the error and the quadratic smoothing objective.



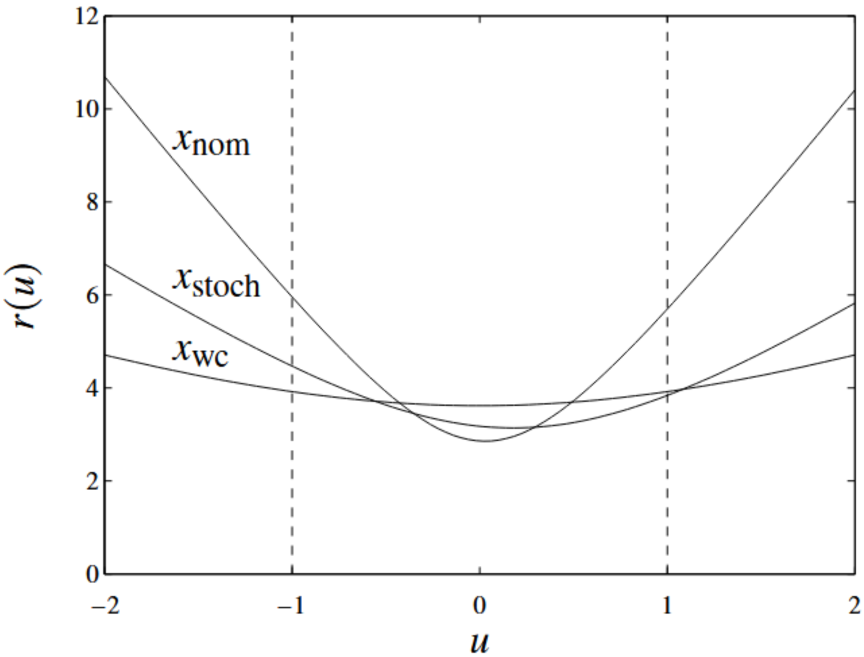
This is essentially "taking a high-frequency signal and putting it through a low-pass filter." Choosing a big penalty gives us something very smooth but not very accurate, and a small penalty just means we still have something that's pretty noisy. In this particular case, the top solution is a bit too inaccurate (it misses some of the original features), while the bottom solution is not regularized enough.

One big thing to remember is that whenever we see an ℓ^1 norm, we should expect sparse things in the arguments and so basically $\hat{x}$ should be piecewise constant. (And if we put in the "second finite differences" in place of the first finite differences, we get piecewise linear functions.) This also means that **using total-variation smoothing will preserve sharp transitions in the signal** much more than quadratic smoothing, for example if there is a discontinuous jump in x.

Example 161

Finally, let's go back to the idea of robust approximation where we want to minimize $||Ax - b||$ with uncertain A. We can take a stochastic approach where we minimize $\mathbb{E}[||Ax - b||]$, or a worst-case approach where we minimize $\sup_A ||Ax - b||$. Despite both of these being convex problems, the problem is only really tractable in special cases (for certain norms, distributions, sets, and so on).

For example, suppose A only varies on a line segment so that it is of the form $A_0 + uA_1$ for $u \in [-1, 1]$. Then if we just optimize ignoring uncertainty (think of this as choosing the "nominal" x), we're the most accurate at $u = 0$ but can have big error $r(u) = ||A(u)x - b||_2$ away from that. A "stochastic" approach where we assume u is uniform on $[-1, 1]$ is then on average more accurate (things get less worse as A varies), and a "worst-case" approach makes the biggest possible $r(u)$ smaller and so we get something as robust as possible at the cost of it being inaccurate near $u = 0$. This is indeed what we see:



In the case of least-squares with a specific known form for the randomness, we can actually analytically solve and find that the solution is equivalent to ridge regression if the error has centered, uncorrelated entries. We'll talk about this more next time!

14 February 10, 2026

We'll finish up robust approximation and then move on to **statistical estimation** today. Remember that in each case, we're looking at some simplified generic problems to get used to the terminology and main ideas. (On our homework, we're doing some real practical examples from various fields, and we should feel free to use code generation but only after we've really figured out the problem ourself so that we actually learn.)

In the setting of last class, suppose we want to solve a least-squares problem but we acknowledge that we don't know the matrix A.

Example 162

Suppose $A = \overline{A} + U$ for U a random perturbation matrix with $\mathbb{E}[U] = 0$ and $\mathbb{E}[U^T U = P]$. The stochastic least-squares problem then asks us to minimize $\mathbb{E}\left[||(\overline{A} + U)x - b||_2^2\right]$, and if we expand everything out we see that the explicit expression we want to minimize is the objective

$$||\overline{A}x - b||_2^2 + x^T P x,$$

which is like the nominal objective with an extra penalty in the directions where A varies a lot. So we get the usual Tikhonov regularization (ridge regression) if P is a constant times the identity matrix, for example if the errors of the "features" in a statistical model are iid centered Gaussian.

Example 163

In **worst-case robust squares**, suppose we know that A lies in some ellipsoid

$$\mathcal{A} = \{\overline{A} + u_1 A_1 + \cdots + u_p A_p\}$$

and so we want to minimize $\sup_{||u||_2 \le 1} ||P(x)u + q(x)||_2^2$, where $q(x) = \overline{A}x - b$ and $P(x) = \begin{bmatrix} A_1 x & A_2 x & \cdots & A_p x \end{bmatrix}$.

This problem is not convex, since we're maximizing something (the function $||Pu + q||_2^2$) that is convex. But it turns out strong duality actually holds in this case: we end up being able to solve the equivalent semidefinite program

$$\begin{aligned} \text{minimize} \quad & t + \lambda \\ \text{subject to} \quad & \begin{bmatrix} I & P & q \\ P^T & \lambda I & 0 \\ q^T & 0 & t \end{bmatrix} \succeq 0. \end{aligned}$$

This is a "minimax problem," and the completely generic trick to keep in mind is the following: if we want to minimize $\min \max$, then we can **look at the** $\max$ **problem and replace it by its dual**, which will be a minimization problem (of a convex function). But this is a $\min \min$ problem, which means we can turn things into just a min problem.

Fact 164

There are lots of isolated non-convex problem with zero duality gap, and they are usually pretty weird and involve some advanced stuff. But there is a big general family of them, which is **any problem involving two quadratic functions**. This is only convex if we want to minimize a convex quadratic or something like that, but all of the other ones also have zero duality gap.

In general, the point is that in robust optimization, ignoring the uncertainty often gives you a much larger range of objective values when you actually sample including the uncertainty (so the norm we get can sometimes be much bigger). So sometimes we want to give up a bit of the nominal cost while not making things go up too much due to randomness.

Remark 165. *A "trade secret" for handling robustness, though, is that there are always things we don't know and one good way to handle it is to just take five or ten versions (scenarios) and minimax over those different versions. That already gives us maybe 80 percent of the benefit, since just telling the optimization problem we don't know things for sure is already a big help. (Statisticians would say you need many more orders of magnitude to provably be close, but in practice this really isn't necessary.)*

We'll now do a whirlwind tour through our next topic of **statistical estimation**.

Definition 166

In **maximum likelihood estimation (MLE)**, we have a bunch of different densities parameterized by some parameter x (or usually θ in actual applications). Let these densities be written $p_x(y) = 0$ (and let $p_x = 0$ if x is invalid; for example if we have a covariance matrix $N(0, \Sigma)$, x is Σ and it has to be positive semidefinite). We call p_x, **as a function of** x, the **likelihood function**, and $\ell(x) = \log p_x(y)$, as a function of x, is called the **log-likelihood function**. In an MLE problem, we wish to choose x that maximizes p_x (or equivalently ℓ).

As long as $\log p_x(y)$ is concave in x for fixed y, this is a convex optimization problem. (Note that this is **not the same** as having a log-concave density – that would require $\log p_x(y)$ to be concave in y for fixed x. They are the same if we have an exponential family, though, since the density is then proportional to the exponential of an inner product.)

Example 167

In a linear measurement model $y_i = a_i^T x + v_i$ where we have m measurements, x is a parameter we want to estimate, a_i is like the measurement model or the "physics prediction," and v_i is some iid noise with density $p(z)$. Thus the density of a particular measurement $y \in \mathbb{R}^m$ that we get will be

$$p_x(y) = \prod_{i=1}^{m} p(y_i - a_i^T x),$$

and thus our goal is to maximize the log-likelihood $\ell(x) = \sum_{i=1}^m \log p(y_i - a_i^T x)$.

For a few examples, Gaussian noise would just turn this into

$$\ell(x) = -\frac{m}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(a_i^T x - y_i)^2,$$

so we're just doing least-squares regression (and it's the same solution regardless of σ). Similarly, Laplacian noise would ask us to minimize the ℓ^1-norm instead, meaning that we're much less unhappy about outliers than in the Gaussian case. (This makes sense, because the tails of a Gaussian decay much faster and so we should be very surprised to see a 5σ deviation.)

Remark 168. *If we try doing Huber fitting, which is basically like least-squares for modest residuals and then "more chill" for big residuals, the maximum-likelihood equivalent of this to have a piecewise function which is Gaussian up until some point (which we can tune as a hyperparameter) and then Laplacian (exponentially decaying) outside of that. So that's saying that our measurement errors are centrally Gaussian, but there's a 3 percent probability of actually being quite a bit bigger; in such situations we want to be pretty robust to outliers.*

Example 169

In logistic regression, we have a random variable which lies in $\{0, 1\}$ with distribution

$$p = \mathbb{P}(y = 1) = \frac{\exp(a^T u + b)}{1 + \exp(a^T u + b)}$$

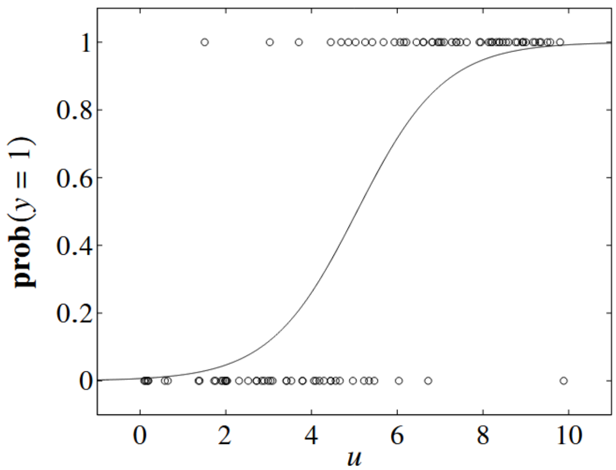
for parameters a, b, where u are some explanatory variables. Our goal is then to estimates a, b given some observations (u_i, y_i).

In this case, the log-likelihood can be written out: if $y_1 = \cdots = y_k = 1$ and $y_{k+1} = \cdots + y_m = 0$, then we work out

$$\ell(a, b) = \sum_{i=1}^{k}(a^T u_i + b) - \sum_{i=1}^{m}\log(1 + \exp(a^T u_i + b))),$$

which is indeed concave in the variables a, b of interest.

For a visual example, suppose we have just one explanatory variable and we see that some values of u give $y = 0$ and others give $y = 1$. Then the logistic fit will be the curve shown:



This is of course a bit silly, since it's really useful to do this when u is very high-dimensional and we can't visualize things. But it's generically doable, and it's an example of a "generalized linear model" which can be turned into convex optimization.

Example 170

Next, suppose we have a Gaussian covariance estimation problem where we want to fit a $N(0, \Sigma)$ to observed data $y_1, \cdots, y_N$. (We can do the mean and variance simultaneously as well.) The log-likelihood here works out to

$$\begin{aligned} \ell(\Sigma) &= \frac{1}{2} \sum_{k=1}^{N} (-2\pi n - \log\det \Sigma - y_k^T \Sigma^{-1} y_k) \\ &= \frac{N}{2}(-2\pi n - \log\det \Sigma - \operatorname{tr} \Sigma^{-1} Y), \end{aligned}$$

where $Y = \frac{1}{N} \sum y_k y_k^T$ is the empirical covariance.

$y^T \Sigma^{-1} y$ is convex in Σ (similarly to how $\frac{1}{x}$ is convex in x for the scalar case), so the last term gives us something concave. Unfortunately, $\log\det \Sigma$ is concave and so that's actually a convex contribution to our problem! So this is not a convex optimization problem as stated. But we can still solve it and the MLE is the empirical covariance, and now we'll see what the trick is. If we change variables to $S = \Sigma^{-1}$ instead (this is sometimes called the "natural parameter" in an exponential family description), then the log-likelihood becomes

$$\ell(S) = \frac{N}{2}(-2\pi n + \log\det S - \operatorname{tr}(SY)),$$

and now everything is of the right curvature. This has an analytical solution $S = Y^{-1}$ with no other constraints (but we can still solve it with more constraints, too), and a similar trick works for the nonzero mean case.

Remark 171. *There's a small group of transformations that come up over and over – one of them is this, and another is to take logs of positive variables (much like how we often take logs of power in various scientific fields). Basically the point is that in many problems, there are natural parametrizations that are exactly the things that turn problems into convex problems.*

If we now think about **sparsifying** S, the story becomes more interesting. We call S the **precision** or **information** matrix; $S_{ij} = 0$ means that conditioned on all other ys, y_i and y_j are **independent**. So a sparse inverse covariance matrix means that many components are conditionally independent, meaning that we can describe things via a sparse

Bayes network. So if our goal is now to fit data and we want S to be sparse, we can instead minimize the convex function

$$-\log\det S + \operatorname{tr}(SY) + \lambda \sum_{i\neq j} |S_{ij}|$$

using a convex sparsifier (and where λ is a hyperparameter). Since many true precision matrices are actually sparse, our regularized estimation errors tend to be much better this way than just using Y^{-1}.

Example 172

We'll next turn to (a simple version of) **hypothesis testing**. Suppose we have a finite number of outcomes for our observation n, and we want to tell whether it comes from hypothesis 1 (a distribution p) or hypothesis 2 (a distribution q). Of course we can't tell if $p = q$ and we can really tell if p, q have disjoint support, but in general it's more difficult to say. What we can do is construct a nonnegative matrix $T \in \mathbb{R}^{2\times n}$ such that $1^T T = 1^T$, so each column tells us the probabilities of deciding hypothesis 1 versus 2 (so if the column is $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ we always think it's hypothesis 2).

We then get a "confusion matrix"

$$D = \begin{bmatrix} Tp & Tq \end{bmatrix} = \begin{bmatrix} 1 - P_{\text{fp}} & P_{\text{fn}} \\ P_{\text{fp}} & 1 - P_{\text{fn}} \end{bmatrix},$$

where "fp" and "fn" are the probabilities of "false positive" (select 2 if it's actually 1) and "false negative" (vice versa). So the multi-objective formulation of this problem is to minimize $(P_{\text{fp}}, P_{\text{fn}})$ subject to constraints on the detector matrix (nonnegative entries with columns summing to 1).

We can scalarize with weight λ, and this is an LP with a simple analytical solution; it turns out to literally just be

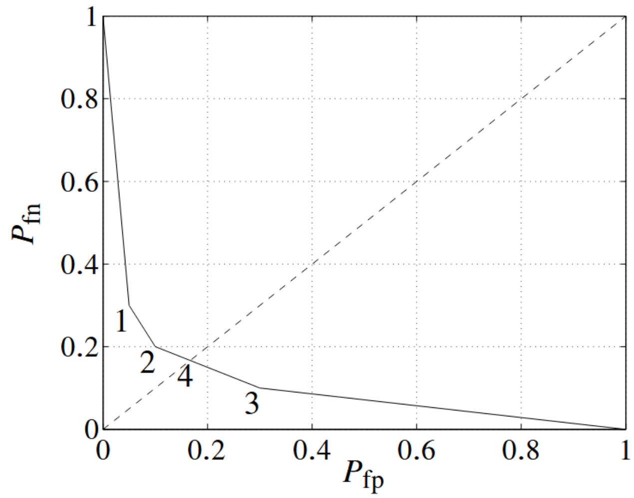
$$(t_{1k}, t_{2k}) = \begin{cases} (1, 0) & \text{if } p_k \geq \lambda q_k, \\ (0, 1) & \text{if } p_k < \lambda q_k. \end{cases}$$

This is called a **likelihood ratio test**; if $\lambda = 1$ this is just minimizing the total error and this actually becomes "maximum likelihood" (we see whether the event was more likely under p or q). So actually our randomized detector is just a deterministic detector.

On the other hand, if we want to minimize $\max(P_{\text{fp}}, P_{\text{fn}})$ instead of the sum or a linear combination, the solution is no longer deterministic:

Example 173

Suppose we have two hypotheses and four outcomes encoded in the matrix $\begin{bmatrix} 0.70 & 0.10 \\ 0.20 & 0.10 \\ 0.05 & 0.70 \\ 0.05 & 0.10 \end{bmatrix}$. Intuitively, we should be guessing p if we observe outcome 1, and we should be guessing q if we observe 3. But for example outcome 4 is a bit less clear and it depends which kind of error we care more about. The Pareto curve (see below) actually turns out to be piecewise linear. The corners of the curve (labeled 1, 2, 3) are deterministic, meaning that they are either always returning p or returning q. But point 4 is the minimax detector (it's where the curve intersects $P_{\text{fn}} = P_{\text{fp}}$) and it is in fact randomized.



So the detector would return nonzero probabilities of both distributions for some outcomes, which might feel weird at first. But this kind of idea of "mixed strategies" comes up in various mathematical "games."

Example 174

Finally, consider the setting of **experiment design**. Suppose we have m linear measurements $y_i = a_i^T x + w_i$ where w_i are iid standard normal, but the a_is are to be chosen. The least-squares (ML) estimate is that $\hat{x} = \left(\sum_{i=1}^m a_i a_i^T\right)^{-1} \sum_{i=1}^M y_i a_i$. The error then has zero mean and has covariance matrix

$$E = \mathbb{E}[ee^T] = \left(\sum_{i=1}^m a_l a_i^T\right)^{-1}.$$

So the setting is that we want the "confidence ellipsoid" E to be small in an experiment, and we want to **choose test vectors from some finite set** $(v_1, \cdots, v_p)$ so that we are pretty sure (maximally informative) about our value x.

First of all, the order of measurements shouldn't matter since we're just adding errors, so it just matters how many of each type we have which we can encode as a vector optimization problem. Our variables are the integer counts m_k (which must be nonnegative and sum to m) and we want to somehow minimize the matrix $E = \left(\sum_{k=1}^p m_k v_k v_k^T\right)^{-1}$. We have to first scalarize so this makes sense; we can for example minimize $\log\det E$ (aka make the volume small) or $\operatorname{tr} E$ (so somehow minimizing errors in the sum of the different directions) or $\lambda_{\max}(E)$.

So the only thing keeping this from being convex is the integer constraint. (It's interesting because if $p = 3$ and $m = 5$, we can just enumerate all possibilities, and if $p = 3$ and $m = 20000$ then we can just pretend they're floats and round and we don't lose anything since we're in some kind of "thermodynamic limit." But there's an area in the middle where it's hard to enumerate but we can't quite do that approximation.) Especially in the case $m \gg p$ though, it's pretty good to use $\lambda_k = \frac{m_k}{m}$ as the continuous real variable and do simple rounding for a heuristic. The optimal value for this relaxed problem is then a lower bound on the optimal value for the integer experiment design problem, just like in our boolean LP example earlier in the course, and rounding gives us the corresponding upper bound.

15 February 10, 2026 (Problem Session)

We'll begin with a practical example:

Example 175

Suppose we have an optimal control problem with the discrete "state evolution equation" $x_{t+1} = Ax_t + Bu_t$ for all $t = 0, \cdots, N-1$, where we specify $x_0 = x^{\text{init}}$ and require a final target state $x_N = 0$. (Think of $x_t \in \mathbb{R}^n$ as a summary of the state of the system at a certain time, which is enough to figure out what happens going forward.) We then wish to choose some inputs $u_0, \cdots, u_{N-1} \in \mathbb{R}^m$ (we can call these **control actions**) satisfying those constraints and optimizing some particular objective.

For example, suppose we have a mass in 3D space; to predict what happens in ε seconds from now, we need its position and velocity coordinates, and those would constitute x_t. We should always make sure to "type-check" all of the quantities of interest: we're given A, B, and N, and in this case A (the "dynamics matrix") will be $n \times n$ and B (the "input matrix") will be $n \times m$.

Remark 176. *In the textbook, there's a version of this equation which looks more intimidating, where A, B themselves have a dependence on time. Those matrices represent the parameters of our system; in our case we have a* **linear, time-invariant** *state space model, but there can be variations where the parameters do vary and that can be much harder to analyze.*

Just to get a sense of how this system is evolving, we can iterate this for a few steps: we have

$$x_1 = Ax_0 + Bu_0, \quad x_2 = A(Ax_0 + Bu_0) + Bu_1 = A^2x_0 + ABu_0 + Bu_1,$$

$$x_3 = Ax_2 + Bu_2 = A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2.$$

We can now start to see some patterns: there is always a component coming just from the initial state $A^t x_0$, and then each of the other terms comes from applying some number of As to Bu_t terms. And in fact this lets us write out that in general (by inductively plugging in the previous term)

$$x_t = A^t x_0 + \sum_{k=0}^{t-1} A^{t-1-k} B u_k.$$

The first term is usually called "ZIR" (zero-input response; this is what happens when we have no inputs u_t at all) and the rest is called "ZSR" (zero-state response; this naming means we zero out the initial x_0 and only take into account the inputs or control commands that were given.)

We still need to massage this a bit to get it into matrix-vector form: if we stack all of the powers of A as a single matrix and also all of the other control terms, we can write

$$x_t = A^t x_0 + \begin{bmatrix} B & AB & \cdots & A^{t-1}B \end{bmatrix} \begin{bmatrix} u_{t-1} \\ u^{t-2} \\ \vdots \\ u_0 \end{bmatrix}.$$

Writing this whole thing in condensed form as $x_t = A^t x_0 + \mathcal{A}_t v_t$, our constraint at the final time is that

$$\begin{aligned} 0 = x_N &= A^N x_0 + \mathcal{A}_N v_N \\ \implies \mathcal{A}_N v_N &= -A^n x_0. \end{aligned}$$

In control theory the matrix $\mathcal{A}_N = \begin{bmatrix} B & AB & \cdots & A^{N-1}B \end{bmatrix}$ is called the **controllability matrix**, since it tells us whether it's even possible to make a system go to a given target or not. And the vector $v_N = \begin{bmatrix} u_{N-1} \\ u_{N-2} \\ \vdots \\ u_0 \end{bmatrix}$ is the total set of possible inputs we can put in. Again doing dimensional analysis, each block $A^k B$ is $n \times m$, so the controllability matrix $\mathcal{A}_N$ is $n \times Nm$ and the set of inputs is an Nm-dimensional vector (so everything checks out).

For a sense of scale, the data file being considered has $n = 4$, $m = 2$, $N = 100$, so the controllability matrix is very short and wide and thus it should be very possible to reach any given final state (we have an underconstrained system). If we had more equations than unknowns, we would have an overconstrained system and perhaps the right solution is to do least-squares to get as close as possible. But instead, here the point is that we have some vector that we can control, v_N, and we will optimize it over the feasible set $\mathcal{A}_N v_N = -A^n x_0$.

In various subparts of our problem, our objective may differ; for example, perhaps we want to minimize the sum of the squares $\sum_{k=0}^{N-1} ||u_k||^2$. But we've packaged all of our u_ks into a single v, and in fact the sum of squares of norms is exactly $||v_N||_2^2$. So that makes it easy for us to translate this all to code: with the notation above, we want to solve (over v)

$$\begin{aligned} \text{minimize} \quad & v^T v \\ \text{subject to} \quad & \mathcal{A}_N v = -A^N x_0. \end{aligned}$$

We can also compute the Lagrangian for this problem, which would be (we use μ for the equality constraint)

$$\begin{aligned} L(v, \mu) &= v^T v + \mu^T (\mathcal{A}_N v - A^N x_0) \\ &= v^T v + \mu^T \mathcal{A}_N v - \mu^T A^N x_0 \\ &= v^T v + (\mathcal{A}_N^T \mu)^T v - (A^N x_0)^T \mu \end{aligned}$$

with domain $(v, \mu) \in \mathbb{R}^{Nm} \times \mathbb{R}^n$. This is a quadratic of positive curvature, hence easy to minimize over v: we have

$$\nabla_v L(v, \mu) = 2v + \mathcal{A}_N^T \mu$$

(here using that the derivative of a quadratic form is $\frac{\partial}{\partial z}(z^T A z) = (A + A^T) z$), and so setting this equal to zero yields $v = -\frac{1}{2} \mathcal{A}_N^T \mu$. Thus plugging back in yields

$$g(\mu) = \inf_v L(v, \mu) = -\frac{1}{4} \mu^T \mathcal{A}_N \mathcal{A}_N^T \mu - (A^N x_0)^T \mu,$$

which is always a concave function (a quadratic with a negative curvature).

We can now determine μ in various ways: one of them is that we can apply the equality constraints and write

$$\mathcal{A}_N v = -A^N x_0 \implies -\frac{1}{2} \mathcal{A}_N \mathcal{A}_N^T \mu = -A^N x_0.$$

This matrix $\mathcal{A}_N \mathcal{A}_N^T$ is an $n \times n$ matrix, and it's invertible **as long as $\mathcal{A}_N$ is of full row rank** (which it is in our example). This tells us the value of the dual variable to plug in:

$$\mu = 2(\mathcal{A}_N \mathcal{A}_N^T)^{-1} A^N x_0,$$

and CVXPY will compute for us the optimal primal variable

$$\boxed{v = -\mathcal{A}_N^T (\mathcal{A} \mathcal{A}_N^T)^{-1} A^N x_0};$$

that is, there is a valid analytical solution in this case, and the $\mathcal{A}_N^T(\mathcal{A}\mathcal{A}_N^T)^{-1}$ part is the **right pseudo-inverse**.

Fact 177

Now that we've seen how to solve the problem analytically, the nice thing about CVXPY is that we can just type in the problem in the initial form and it'll give us the optimal solution too.

We can encode the values of $x_0, \cdots, x_N$ as a `Variable((n, N+1))` and the values of $u_0, \cdots, u_{N-1}$ as a `Variable((m, N))`, and then we can append constraints to our list of constraints as well (initial state $x_0 = x^{\text{init}}$, final state $x_N = 0$). And we can then type in any objective we want; it's just important to keep in mind that because the xs are encoded in a single matrix, we may have to write the objective in some interesting ways. For example, if we want to minimum the sum of the 2-norms, we may first apply `norm` over the zeroth axis and then apply `sum` to the result.

We will notice that the configuration of inputs looks rather different depending on the objective. In particular, optimizing something involving 1-norms tends to create sparsity, but we'll see different kinds of sparsity depending on the specific objective as well:

- If we want to minimize $\sum ||u_t||_2$, then only some of the vectors will be nonzero because we're taking the "ℓ^1 norm of those ℓ^2 norms," and if we want to minimize $\sum ||u_t||_1$, then furthermore only some components of those vectors will be nonzero (because even the u_ts themselves should be sparse). And it's also good to remember that ℓ^2 error penalizes outliers heavily, so we tend to expect things to be more spread-out for the sake of bringing down the largest values.
- Suppose we want to minimize the function $\max ||u_t||_2$ instead. Since there is no penalty for anything besides a single maximum value, the optimization will often try to make the values of u_t as uniform as possible so that all of the contributions can come in without any penalty. (It's useful to keep in mind though that the objective is not to make things uniform – that's often just a natural consequence of minimizing the peak.)
- Finally, we can use the Huber norm (with varying thresholds M) at which our quadratic objective turns into a linear one. If M is extremely small, then we basically expect the optimization to look like the ℓ^1 case (very sparse), and if M is extremely big, we expect it to be like ℓ^2 (avoiding big outliers by spreading things out; generally more "continuous"-looking plots). And in between, we see plots that morph from one description to the other.

16 February 12, 2026

We'll move on to a broad area of problems involving geometric programs today, and that'll wrap up the "overview of applications" section for a while. But first, let's discuss experiment design a bit more: recall that the idea is that we get to make some number of measurements, and we get to pick from a palette of choices and wish to minimize the covariance error matrix (via some scalarization, which is a monotone mapping from $S^n_{++} \to \mathbb{R}$ in the sense of the positive definite ordering). And we usually solve the integer-constrained problem by approximating with a relaxation, and then rounding to a valid integer solution (this is called "relax-and-round") and maybe doing a local search afterward (this is sometimes called "polishing") by seeing whether the objective improves if we swap one experiment from one choice to another.

If we scalarize via the log determinant, we do what's called "D-optimal experiment design" (we're minimizing the volume of the confidence ellipsoid). Often when we solve a problem, we also work out the dual, not necessarily for a computational reason but for useful interpretation. In this case, the dual problem is

$$\begin{aligned} \text{maximize} \quad & \log\det W + n\log n \\ \text{subject to} \quad & v_k^T W v_k \le 1 \text{ for } 1 \le k \le p. \end{aligned}$$

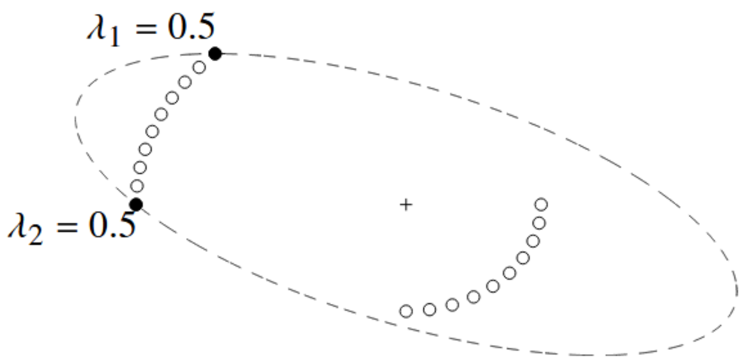
The interpretation of what's happening here is that $\{x : x^T W x \le 1\}$ is an ellipsoid centered at the origin, and we ask for it to have minimum volume given that it includes all of our test vectors v_k. Complementary slackness in particular tells us that if λ_k and W are the primal and dual optimal variables, then

$$\lambda_k(1 - v_k^T W v_k) = 0 \text{ for } 1 \le k \le p;$$

that is, the vectors are typically on the boundary of that minimum-volume ellipsoid. (Notice also that this means we will never one of the v_is that's strictly in the convex hull of some others – that makes sense because those points inside will have a lower signal-to-noise ratio.)

Remark 178. *We might ask why it's not best to just repeatedly use the single point v_i which gives us the highest signal-to-noise ratio. But for D-optimal design, this will give us a confidence ellipsoid that's degenerate – we'll do super well in one direction v_i, but we'll know nothing in the other directions and that volume will be infinite. So the point is to take points that are both relatively uncorrelated and also big.*

In the example below, there are 20 total possible experiment points chosen, but the best design in this case happens to be to only use two of them and to use them each 10 times.



We'll move now to the generic category of **geometric problems**. Of course, there isn't a strict cutoff between this topic and the previous one, but it's mostly a matter of formulation. We'll begin with some simple examples of geometric problems, not even complicated enough to put on the slides:

- If we want to find the closest point in a set to another given point, where the set was described as a polyhedron, we can just solve the resulting QP.
- If we want code to see whether two polyhedra intersect, we can just set up the constraints together and solve the feasibility problem.

Example 179

Suppose we want to find the minimum-volume ellipsoid that covers a set C (not necessarily convex), which we call the **Löwner-John ellipsoid** $\mathcal{E}$ of C. Generically, finding it is a convex optimization problem.

We can parametrize an ellipsoid as $\{v : ||Av + b||_2 \le 1\}$ for some positive definite A and some vector b. (We can constrain A to be positive definite instead of just invertible, because this set is $(A, b) \mapsto (QA, Qb)$ for any orthogonal matrix Q, so we can use the SVD and turn $U\Sigma V^T$ into $V\Sigma V^T$.) Note however that it's a bit subtle how we choose the right parametrization depending on the problem we're solving; sometimes one of them works and another doesn't. For this case, though, we now want to minimize $\log\det A^{-1}$ (which is indeed convex), subject to the constraint $\sup_{v\in C} ||Av+b||_2 \le 1$. This is indeed convex because $||Av+b||_2$ is convex in A, b, and then a general supremum of convex functions is convex.

The issue, though, is that if C is a general set, then evaluating this constraint can be quite hard. But the case where $C = \{x_1, \cdots, x_m\}$ is finite, and so it's completely tractable to set up the convex optimization problem with m constraints. And in fact, this also gives us the Löwner-John ellipsoid for the polyhedron $\text{conv}(x_1, \cdots, x_m)$ as well.

Remark 180. *Sometimes we want to remove suspected outliers from a dataset in a way that doesn't depend on our sense of scale. We can do this by fitting a minimum-volume ellipsoid, removing the points on the boundary, and seeing whether that improves our performance. This goes under the name of* **ellipsoidal peeling**. *And if we now want to remove them one-by-one, we can take a look at the dual variables and see which ones are largest.*

Example 181

Now for something slightly different, suppose we have a convex set C and want to find the maximum-volume ellipsoid $\mathcal{E}$ that lies entirely enside C. For example, this might make it easier to encode a feasible region with a simpler object if a robot needs to repeatedly check whether we're inside some safe zone every millisecond.

The difference here is that we will now parametrize via the forward image and write $\mathcal{E} = \{Bu + d : ||u||_2 \le 1\}$. Again we can assume B is positive definite, and this time the problem we want to solve would be (over B, d)

$$\begin{aligned} \text{maximize} \quad & \log\det B \\ \text{subject to} \quad & \sup_{||u||_2\le 1} I_c(Bu+d) \le 0, \end{aligned}$$

where I_C is the indicator function of lying in C (so 0 if we're in it, ∞ otherwise). This is indeed a convex optimization problem, since each I_C is convex so their maximum is as well, and $\log\det B$ is concave. This time, the simple case where we can actually evaluate the constraints tractably is if C is a polyhedron $\{a_i^T x \le b_i \text{ for } 1 \le i \le m\}$: indeed, since $\sup_{||u||_2\le 1} a_i^T(Bu+d) = ||Ba_i||_2 + a_i^T d$, the problem reduces in this case to

$$\begin{aligned} \text{maximize} \quad & \log\det B \\ \text{subject to} \quad & ||Ba_i||_2 + a_i^T d \le b_i \text{ for } 1 \le i \le m. \end{aligned}$$

So we can calculate the maximum volume ellipsoid inside a polyhedron.

Fact 182

We might ask whether we can flip the problem around and find the minimum-volume ellipsoid covering a region bounded by some number of linear inequalities, or the maximum-volume ellipsoid that lies inside the convex hull of some set of vertices. But there's actually an asymmetry here, and both of those problems are actually NP-hard! (The issue is basically that converting from inequality constraints to number of vertices might give us exponentially many things.)

The tractable versions of the problems come with nice efficiency bounds, though. For example, if C is convex and bounded with nonempty interior in $\mathbb{R}^n$, and we calculate the minimum-volume ellipsoid that includes it, then **shrinking by a factor of n guarantees that we lie inside** C, and in fact if the set is symmetric we improve this to $\sqrt{n}$ and this is exactly a tight bound. (For example, visualizing an equilateral triangle, its circumcircle shrunk by a factor of 2 ends up being the incircle.) So in words, "we can universally approximate any convex set with an ellipsoid within a factor of $\sqrt{n}$" by splitting the difference. The same goes in reverse too: the maximum-volume inscribed ellipsoid expanded by a factor of n contains the set.

For another perspective, this tells us that any unit norm ball can be approximated by the Euclidean ball up to a factor of $\sqrt{n}$ and so "all norms in finite-dimensional Euclidean space are equivalent" and so all of our optimization problems can kind of been approximated by quadratic optimization problems.

Remark 183. *Note that the operations we're doing here commute with affine transformations of $\mathbb{R}^n$. So we can figure out the minimum-volume ellipsoid of any triangle by just doing an affine shift of the equilateral triangle, and similarly the set of points on boundary of a minimum covering ellipsoid also remain the same under such transformations (though the Lagrange multipliers may change when we do this).*

We'll now think about **centering** problems; that is, choosing a baseline value that is as resistant to error as possible by picking something "deep within a set." There are many possible definitions of the center of a convex set, even in $\mathbb{R}^2$, and there are many different perfectly good ones:

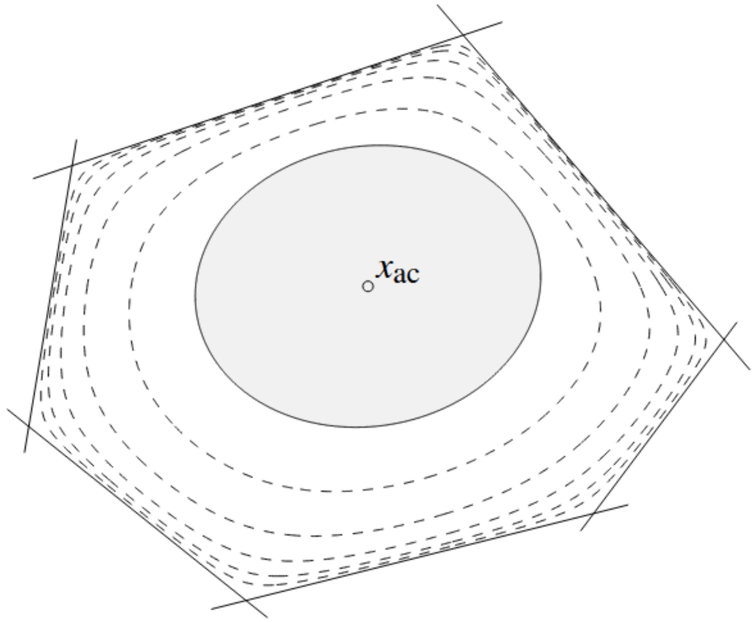
- The **Chebyshev center** (which we saw earlier on in the course) is the center of the largest inscribed ball; that is, it is the point "deepest in the set." (This can be found via linear programming for a polyhedron.) This is unfortunately not unique, for example if we imagine a long rectangle.
- Alternatively, we can take the **center of the MVE** (maximum-volume inscribed ellipsoid) as we've been discussing above. This one yields a unique solution because the convex optimization problem we set up for it is actually strictly convex, and it's of course invariant under affine coordinate transformations (unlike the Chebyshev center).
- The **analytic center** of some set of convex inequalities $f_i(x) \leq 0$ and linear equations $Fx = g$ is the solution of the problem

$$\begin{aligned} \text{minimize} \quad & -\sum_{i=1}^{m} \log(-f_i(x)) \\ \text{subject to} \quad & Fx = g; \end{aligned}$$

  this objective function is convex, called the **log-barrier** for the inequalities, and it basically maximizes the geometric mean of the margins of our inequalities.

  The point is that the analytic center is easy to compute compared to the MVE or Chebyshev center, but we should be careful because a set can be described in multiple ways and yield different analytic centers. (For example, a redundant constraint doesn't change the set but it does bias the log-barrier.) That is, the analytic center is description-dependent.

Below is a visual representation of the level curves of $\phi(x) = -\sum_{i=1}^{m} \log(b_i - a_i^T x)$, where we have m total linear constraints bounding a polyhedron. As we move towards any boundary, one of the inequalities becomes tight and thus ϕ approaches $+\infty$. So we can think of the curves as either being near-approximations of the polyhedron or converging towards the analytic center.



All smooth functions look quadratic near their minimum, so we'll get something that looks like an ellipsoid near the analytic center which kind of gives us the general shape. For example, when we're minimizing log-likelihood, we can think about points near that optimal point, and the corresponding ellipsoid is exactly the Fisher information, which tells us about how well we can distinguish two such points with a hypothesis test.

It turns out that our polyhedron is bounded between the ellipsoids (here x_{ac} denotes the analytic center)

$$\{x : (x - x_{\text{ac}})^T \nabla^2 \phi(x_{\text{ac}})(x - x_{\text{ac}}) \leq 1\} \leq \text{polyhedron} \leq \{x : (x - x_{\text{ac}})^T \nabla^2 \phi(x_{\text{ac}})(x - x_{\text{ac}}) \leq m(m-1)\}.$$

Note that this is a shrinkage factor of m rather than n, but because $m \geq n$ this is always worse; it's just easier to compute.

Example 184

Next, consider linear discrimination, in which we have two sets of points (for example, the feature vectors associated with negative and positive outcomes) and want to separate them by a hyperplane. This means we want to find $a \in \mathbb{R}^n$, $b \in \mathbb{R}$ such that $a^T x_i + b > 0$ and $a^T y_i + b < 0$ (that is, everything in one set lies on one side, and everything in the other lies on the other.)

Keep in mind that our variables to optimize over are a and b; the problem is homogeneous so we can also equivalently require that $a^T x_i + b \geq 1$ and $a^T y_i + b \leq -1$. (This is an example where the difference between strict and non-strict inequalities matters; if we just made the original inequalities non-strict then $a = 0, b = 0$ would give us a valid solution, but that's not useful.) So now we have a problem which is a set of linear inequalities in a and b, which means we have an LP feasibility problem.

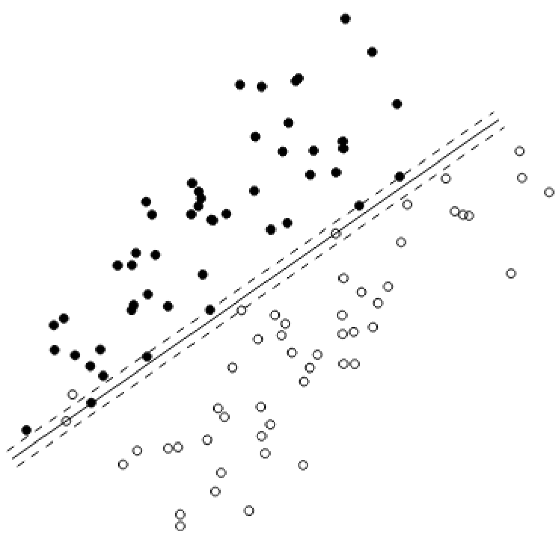
But we can now modify our problem: we can ask to separate two sets of points not just by a hyperplane but by a slab, or equivalently pick the separation which has the maximum margin. The Euclidean distance between the hyperplanes $\{a^T z + b = 1\}$ and $\{a^T z + b = -1\}$ is $\frac{2}{||a||_2}$, so we can recast the "robust linear discrimination" problem as a QP

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}||a||_2^2 \\ \text{subject to} \quad & a^T x_i + b \geq 1 \text{ for } 1 \leq i \leq N, \\ & a^T y_i + b \leq -1 \text{ for } 1 \leq i \leq M. \end{aligned}$$

Similarly, since it might not even be possible to completely linearly separate our data (which definitely happens all the time in machine learning or if we're fitting predictors to medical records), we can do **approximate linear separation** instead and solve

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^T u + \mathbf{1}^T v \\ \text{subject to} \quad & a^T x_i + b \geq 1 - u_i \text{ for } 1 \leq i \leq N, \\ & a^T y_i + b \leq -1 + v_i \text{ for } 1 \leq i \leq M, \\ & u \succeq 0, \\ & v \succeq 0. \end{aligned}$$

This is kind of like minimizing the number of incorrect classifications, but unfortunately that problem isn't convex so this heuristic is better – we're minimizing the sum of violations of the original inequalities. (So $u_i \in [0, 1]$ means we're on the right side but without much margin, and $u_i > 1$ means we're on the wrong side.) An example with some data is shown below – this is another example where the convex problem actually works better in practice than what we may have thought we wanted to solve before.



(And if we also want to maximize the slab of that margin, we can get a **support vector classifier** which trades off between the margin and the classification error.)

Example 185

Turning now to nonlinear discriminators, we can now ask for a nonlinear function $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(x_i) > 0$ and $f(y_i) < 0$. We can do this by parametrizing some **linear** family of functions $f(z) = \theta^T F(z)$ for some basis functions $F = (F_1, \cdots, F_k)$; we then want to solve the set of linear inequalities

$$\theta^T F(x_i) \geq 1 \quad \text{and} \quad \theta^T F(y_i) \leq -1.$$

For example, quadratic discrimination can be done by parametrizing via $f(z) = z^T Pz + q^T z + r$ for variables P, q, r, and polynomial discrimination works by letting our basis $F(z)$ be the set of all monomials up to degree d. Extending this latter example a bit more, if we want to solve the problem of "minimum degree that separates some sets of points," then we're doing a quasiconvex optimization problem and this can be solved via bisection search.

Example 186

Finally, consider the problem of **placement and facility location** (usually in $\mathbb{R}^2$ up to maybe $\mathbb{R}^4$ if we care about both space and time). Suppose we have N points $x_1, \cdots, x_N$, where some of them are given and others are variables (for example, some external pins are fixed in an integrated circuit, and the other positions are other gates). We have some cost function $f_{ij}(x_i, x_j)$ (in warehouse placement it might be 0 if two nodes should be connected, and otherwise it's the driving distance). We wish to minimize the total cost $\sum_{i \neq j} f_{ij}(x_i, x_j)$. In other words, we want things that are connected to each other to be close.

The point is that we get a bunch of convex problems and the optimal placements can depend a lot on our cost function: for example, trying to minimize $\sum_{i,j} ||x_i - x_j||_2$ versus $\sum_{i,j} ||x_i - x_j||_2^2$ versus $\sum_{i,j} ||x_i - x_j||_2^4$ spreads out the free points in different ways, since we're articulating in the latter cases that we really do not want any pair of connected points to be very far apart.

17 February 17, 2026 (Problem Session)

Remark 187. *I could not attend this session in person, so these notes are transcribed from the posted recording.*

We'll start with a problem involving probability and do some background context as well:

Problem 188

Suppose we have binary random variables X_1, X_2, X_3, X_4 (so kind of like flipping coins, except the variables are not necessarily independent) and we're given some information about the outcomes

$$\mathbb{P}(X_1 = 1) = 0.9, \quad \mathbb{P}(X_2 = 1) = 0.9, \quad \mathbb{P}(X_3 = 1) = 0.1,$$

$$\mathbb{P}(X_1 = 1, X_4 = 0 | X_3 = 1) = 0.7, \quad \mathbb{P}(X_4 = 1 | X_2 = 1, X_3 = 0) = 0.6.$$

We're then curious about the minimum and maximum possible values of $\mathbb{P}(X_4 = 1)$.

As a reminder, we have a sample space Ω of all possible outcomes (expressed as finely / exhaustively as possible as mutually exclusively possibilities), which in this case is $\{0, 1\}^4$, which has 16 total possibilities. Events A are just collections of outcomes, and we can think of events A, B as interacting via a Venn diagram. If probabilities are encoded by areas, then the area of A is just its probability. But if we know that B has happened, then we have to restrict to only the circle for event B, and so $\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$ is the **fraction of area** where A happens "within the new universe / conditional sample space B." (We have to do this normalization so that $\mathbb{P}(B|B) = 1$ and we still have a valid probability measure.) We call B the "conditioning event."

In our problem, the variable we are optimizing over is the **probability vector** p of 16 different possibilities $(0000, 0001, 0010,$ and so on, representing the values of $X_1, X_2, X_3, X_4)$. If we wanted to draw this out visually, we could use the x-axis for both X_1 and X_2 (with columns for $00, 01, 10, 11$) and similarly the y-axis for X_3 and X_4. Out of those 16 possibilities, eight of them correspond to $X_4 = 1$, so the objective we are trying to minimize or maximize is $p_{0001}+p_{0011}+p_{0101}+p_{0111}+p_{1001}+p_{1011}+p_{1101}+p_{1111}$, which can also be written in the form a^Tp for some constant vector a. (The important concept here is that $X_4 = 1$ is an event, rather than a single sample point.) Since this is a linear function, our problem still has hopes of being convex whether we are minimizing or maximizing this objective.

From here, we just need to formulate the constraints. Since p has to be a probability vector, we must have $p \succeq 0$ and $\mathbf{1}^T p = 1$. Then the first three constraints work out very similarly to how $\mathbb{P}(X_4 = 1)$ is defined, except we now set those expressions equal to certain values (so the constraints look like $a_1^T p = 0.9$, $a_2^T p = 0.9$, $a_3^T p = 0.1$ for some vectors a_1, a_2, a_3). For the first conditional probability, notice that because we specify $\mathbb{P}(X_3 = 1) = 0.1$, we can rewrite that condition as

$$0.7 = \mathbb{P}(X_1 = 1, X_4 = 0|X_3 = 1) = \frac{\mathbb{P}(X_1 = 1, X_3 = 1, X_4 = 0)}{\mathbb{P}(X_3 = 1)} \implies \boxed{\mathbb{P}(X_1 = 1, X_3 = 1, X_4 = 0) = 0.7\mathbb{P}(X_3 = 1) = 0.07},$$

which we can rewrite as $p_{1010} + p_{1110} = 0.07$, an affine constraint. And similarly, even though we don't know the probability $\mathbb{P}(X_2 = 1, X_3 = 0)$, we can still rewrite

$$0.6 = \mathbb{P}(X_4 = 1|X_2 = 1, X_3 = 0) = \frac{\mathbb{P}(X_2 = 1, X_3 = 0, X_4 = 1)}{\mathbb{P}(X_2 = 1, X_3 = 0).} \implies \boxed{\mathbb{P}(X_2 = 1, X_3 = 0, X_4 = 1) = 0.6\mathbb{P}(X_2 = 1, X_3 = 0)}.$$

And now this constraint is linear in p as well, since both probabilities in this boxed equation are linear in p. So putting everything together, we actually see that our problem (either minimizing or maximizing the objective) is an LP and so can be solved efficiently.

Example 189

Next, we'll talk a bit about maximum likelihood estimation and how to set up one such problem as a convex optimization problem.

The idea of maximum likelihood expectation (MLE) is that we have random variables $Y_1, \cdots, Y_m$ (which map the m outcomes of a random experiment to numbers). If these random variables are continuous, then we have some probability distribution function f, and we'll assume in this case that the density is specified by some set of parameters $\theta = (\theta_1, \cdots, \theta_\ell)$. (For example for Gaussian random variables, the parameters are the mean μ and the standard deviation σ.) We can then define the **likelihood function**

$$\mathcal{L}(y; \theta) = f_{Y_1,\cdots,Y_m}(y_1, \cdots, y_m; \theta_1, \cdots, \theta_\ell)$$

(here $Y_1, \cdots, Y_m$ denote the random variables, $y_1, \cdots, y_m$ are particular values of those variables, and $\theta_1, \cdots, \theta_\ell$ are the parameters). For example, perhaps we have a coin that we can flip but we don't know the probability p of heads. So then we can collect data and let the Y_is record the outcomes of those independent flips, and the likelihood function is then $p^{\#\text{ heads}}(1-p)^{\#\text{ tails}}$. The idea of MLE is then to ask **what values of the parameters make the result that we saw as likely as possible** (so in this coin-flipping example, what p maximizes this expression). More formally, $\hat{\theta}_{\text{ML}}$ is exactly the value of θ that maximizes $\mathcal{L}(y; \theta)$, and so importantly we think about this problem as optimizing over θs, not ys.

Since log is monotonically increasing, we can also perform the maximization on $\log \mathcal{L}(y; \theta)$ instead of $\mathcal{L}(y; \theta)$ itself. This is good because log is concave and so we have hopes of making the problem convex if we have a log-concave density (which is true for important real-world distributions like Gaussians or Exponentials). We can now state the problem of interest:

Problem 190

Suppose we have a linear measurement model $y_i = a_i^T x + v_i$, where we observe the noise-corrupted y_i and get to specify a_i but the deterministic parameter x is unknown. We assume that we know the normalized probability density function of the errors v_i but not the mean and standard deviation, and so the density of v is given by

$$f(v) = \frac{1}{\sigma} f\left(\frac{v-\mu}{\sigma}\right)$$

and the errors are independent and identically distributed. (In other words, we know the parent shape of the errors but not the exact location and spread.) We wish to show that the MLE estimates for x, μ, σ can be determined by solving a convex optimization problem **as long as** f is log-concave.

(Notice that the normalized probability density function corresponds to a random variable U with mean 0 and standard deviation 1, and we are just stretching that to the random variable $\mu + \sigma U$.) First of all, we are told that the log-likelihood function of interest is

$$\sum_{i=1}^m \log p(y_i - a_i^T x) = -m \log \sigma + \sum_{i=1}^m \log f\left(\frac{y_i - a_i^T x - \mu}{\sigma}\right),$$

and we will derive this now. The likelihood function is

$$\begin{aligned} \mathcal{L}(y; x, \mu, \sigma) &= f_{Y_1, \cdots, Y_m}(y_1, \cdots, y_m; x, \mu, \sigma) \\ &= \prod_{i=1}^m f_{Y_i}(y_i; x, \mu, \sigma) \end{aligned}$$

by independence of the errors. It makes sense to use the log version of MLE to turn this product into a sum, so that

$$\log \mathcal{L}(y; x, \mu, \sigma) = \sum_{i=1}^m \log f_{Y_i}(y_i; x, \mu, \sigma).$$

Now we can relate these f_{Y_i}s to the known log-concave density f – it will turn out the f_{Y_i}s are also log-concave, so that the log-likelihood is concave and we can indeed maximize it to get a convex optimization problem. To calculate this probability density, a standard trick is to look at the cumulative distribution function:

$$F_{Y_i}(y_i) = \mathbb{P}(Y_i \le y_i) \implies f_{Y_i}(y_i) = \frac{d}{dy_i} F_{Y_i}(y_i).$$

In our case, we have

$$F_{Y_i}(y_i) = \mathbb{P}(a_i^T x + V_i \le y_i) = \mathbb{P}(V_i \le y_i - a_i^T x) = F_{V_i}(y_i - a_i^T x)$$

so differentiating both sides in y_i yields that

$$f_{Y_i}(y_i) = f_{V_i}(y_i - a_i^T x).$$

And now we can do a similar transformation to relate V to the normalized U (whose PDF and CDF we denote with just f and F): if $V = \sigma U + \mu$, then

$$\mathbb{P}(V \le v) = \mathbb{P}(\sigma U + \mu \le v) = \mathbb{P}\left(U \le \frac{v-\mu}{\sigma}\right)$$

(the direction of the inequality doesn't change since $\sigma > 0$). Thus we have $F_V(v) = F\left(\frac{v-\mu}{\sigma}\right)$, and taking the derivative of both sides in v yields

$$f_V(v) = f\left(\frac{v-\mu}{\sigma}\right) \cdot \frac{1}{\sigma}$$

by the chain rule. Therefore we actually have

$$f_{Y_i}(y_i) = f_{V_i}(y_i - a_i^T x) = \frac{1}{\sigma} f\left(\frac{(y_i - a_i^T x) - \mu}{\sigma}\right),$$

and plugging this back into the log-likelihood function yields

$$\begin{aligned}
\log \mathcal{L}(y; x, \mu, \sigma) &= \sum_{i=1}^m \log\left(\frac{1}{\sigma} f\left(\frac{y_i - a_i^T x - \mu}{\sigma}\right)\right) \\
&= \sum_{i=1}^m -\log\sigma + \log\left(f\left(\frac{y_i - a_i^T x - \mu}{\sigma}\right)\right) \\
&= -m\log\sigma + \sum_{i=1}^m \log f\left(\frac{y_i - a_i^T x - \mu}{\sigma}\right),
\end{aligned}$$

which is exactly what the problem claims. So now we are optimizing over x, μ, σ simultaneously, and this might seem a bit complicated especially with σ in the denominator. But what we can do now is come up with a change of variables so that the problem is indeed convex in those variables: defining $\tau = \frac{1}{\sigma}$, $\zeta = \frac{x}{\sigma}$, and $\rho = \frac{\mu}{\sigma}$, we now want to optimize

$$-m\log\frac{1}{\tau} + \sum_{i=1}^m \log f\left(y_i\tau - a_i^T\zeta - \rho\right)$$

in terms of τ, ζ, ρ, from which we can indeed recover our original σ, x, μ (and so it's sufficient to solve this transformed problem). And the transformed problem is indeed convex because the objective is concave – the first term is $m\log\tau$, which is concave, and $\log f$ is concave and we are plugging in an affine function of our parameters into its argument. That means we can indeed solve for (τ, ζ, ρ) efficiently and then recover the optimal parameters via $\sigma = \frac{1}{\tau}$, $x = \frac{\zeta}{\tau}$, and $\mu = \frac{\rho}{\tau}$.

18 February 19, 2026

One last comment from the material from last time: we were discussing the optimal placement problem, where we want to minimize some objective in terms of the distance. Observe that if we made the objective a dead-zone penalty (that is, 0 penalty up to some point, at which point it becomes large), we would have a whole bunch of distances right at that cutoff.

Today, we'll start the next part of the course: "**how do we actually solve these convex optimization problems**." It's important to note that very few of us will write algorithms that actually do this solving. (It's important that those small number of people do actually do that, though.) But it's just good for us to demystify things and be able to implement in simple cases.

But today's material is something every educated person should know about – usually in ICME they'd cover this in multiple quarters, but we'll just do a quick background lecture of **numerical linear algebra**.

Definition 191

A **flop** (floating-point operation) is an addition, subtraction, multiplication, or division of floating-point numbers.

(Some people also count things like exponentials or square roots.) The point is to get "gross" estimates of running time and complexity: we express the number of flops approximately as a (usually polynomial) function of the problem dimensions, such as the number of rows or columns of our matrix, and we just look at the leading order behavior. We can kind of then predict how long it takes by comparing the number of flops needed to how many flops our computer can do.

This is really not an accurate predictor of computation time on modern computers (it used to be more accurate back when floating-point arithmetic had to actually be done off-chip), but it's still useful as a rough estimate of complexity.

Example 192

In the 1960s, "basic linear algebra subroutines" were classified into different levels. At level 1 we have vector-vector operations like inner products, sums, scalar multiplications, or distances. All of these operations require $O(n)$ flops (for example, inner products take $n-1$ additions and n multiplications). Level 2 would be matrix-vector products like Ax; if A is an $m \times n$ matrix this takes $O(mn)$ flops, but it can be faster if A is sparse, banded (that is, only nonzero near the diagonal) or given in a particular low-rank factorization. At level 3, we get matrix-matrix multiplication, which can take $O(mnp)$ flops for $m \times n$ by $n \times p$ but again less if we have sparse matrices or other special structures. And in the world of AI, there are now "higher levels" of complexity as well.

As a sidenote, we might ask whether there's anything faster than $O(n^3)$ for multiplying two $n \times n$ matrices. In theory it turns out we can do things in something like $O(n^{2.37})$ using the idea of "blocking" (though there's no case so far where we'd actually use this algorithm to multiply faster). The idea is to block conformally into 8×8 blocks, write the matrix product in that block form, and then recursively perform this. And this block idea actually contributes to the way that matrix multiplication actually works on hardware to make it possible to distribute the computation to different cores. But the actual theory is to recurse down to 2×2 matrices and it turns out we can do it cleverly in 7 flops instead of 8.

The implementation of BLAS and variants (like for sparse matrices) is pretty good now; for single-thread CPUs we can usually do something like 10^9 flops per second, and GPUs can usually do maybe 10^{12} flops per second. And what those GPUs do best is multiplying large dense matrices for exactly this blocking reason.

Example 193

Now let's think about how we can use this to solve linear equations $Ax = b$; that is, compute $A^{-1}b$. There are very few cases where we actually compute the inverse matrix and then multiply it by b (and that's not what the "solve" method in a numerical linear algebra package would do). In general solving takes $O(n^3)$ flops, though if A has various structures it can be much faster, and so it's extremely useful to **recognize matrix structure** that can be exploited.

- Of course, if A is diagonal, then it's easy to solve the system in n flops by dividing entry-wise. Similarly, if A is lower triangular, then we can solve via **forward substitution** (solve for $x_1 = \frac{b_1}{A_{11}}$, then plug into the second equation to solve for $x_2 = \frac{b_2 - A_{21}x_1}{A_{22}}$, and so on), or if A is upper triangular we can do **backward substitution** (solve for x_n, then x_{n-1}, and so on). But this isn't very parallelizable, so again some amount of blocking is used in practice.
- It's also easy to solve $Ax = b$ if A is orthogonal, since $A^T = A^{-1}$ so we just need to compute $A^T b$ (which takes roughly $2n^2$ flops). In the special case where $A = I - 2uu^T$ is a reflection through one particular line, the matrix is completely dense but we can still find the solution $x = A^T b = b - 2(u^T b)u$ in a linear number of flops. So if we can write orthogonal matrices as a product of such reflections we can also invert easily.

- If A is a permutation matrix, then we don't actually have to do any floating-point operations at all because we just rearrange the values of b. So back in the 1960s this would have been very fast relative to floating-point operations, but not really anymore.

All of these are examples where the matrix is "easily inverted," and what that really means is that we can easily get $A^{-1}b$, not that we can get A^{-1} (for example, because the latter might take up a lot of space).

Proposition 194

The **factor-solve method** is a generic way of solving $Ax = b$. What we do is factor A as a product of simple matrices (somewhere between 2 and 5), each of which is easily inverted. Then if $A = A_1 \cdots A_k$, and we want to solve $Ax = b$, then $x = A_k^{-1} \cdots A_1^{-1} b$ by solving a series of k systems of equations $A_1 x_1 = b$, then $A_2 x_2 = x_1$, and so on, rather than computing the inverses.

What ends up happening is that the factorization step is usually the dominant cost by some factor of a problem dimension. For example, the factor for completely generic $n \times n$ matrices takes n^3 but the solve takes just n^2 flops. But then we get some helpful improvements if we want to solve m different equations $Ax_1 = b_1, Ax_2 = b_2, \cdots, Ax_m = b_m$ with the same A – we do the factorization once and cache it. Then the total cost would be one factorization plus m solves, and that's effectively the same if factorization cost is the dominant term. So we can solve a modest number of equations in the same amount of time as one, which is really nice!

Proposition 195 (LU factorization)

For any nonsingular matrix A, we can write $A = PLU$, where P is a permutation matrix, L is lower triangular, and U is upper triangular. The factorization cost turns out to be something like $\frac{2}{3}n^3$ flops, and this is basically done with Gaussian elimination. Once we have this factorization, the inverting of P, L, U each take at most n^2 flops, so the total cost is indeed dominated by that Gaussian elimination part.

Proposition 196 (Sparse LU factorization)

Now suppose A is invertible and **sparse**. Then we can factor as $P_1 LUP_2$ where P_1, P_2 are permutation matrices, chosen in a way so that L, U are sparser. (This basically means that we can permute the rows and columns so that the LU factorization is sparse.) There are a huge array of methods to **heuristically** choose P_1, P_2 during the factorization process, and it works unbelievably well in practice even if there aren't algorithmic guarantees. (But choosing P_1, P_2 to actually provably minimize the number of "fill-ins" in L and U is NP-hard.)

Proposition 197 (Cholesky factorization)

Specializing to the case of a positive definite matrices A, we can now factor as $A = LL^T$ for L lower triangular with positive diagonal entries. The factorization cost ends up being about half of what we need in an ordinary LU factorization, so about $\frac{1}{3}n^3$ flops. And the L is actually unique for these matrices.

Weirdly, sparse Cholesky is different from sparse LU: we now ask for $A = PLL^T P^T$, and we heuristically choose P. What's cool is that the choice of P will only depend on the sparsity pattern of A (where the nonzeros are), so we can determine P from the start (the "symbolic factorization" part) and then do L afterward (the "numerical factorization" part).

We'll apply this in cases like where A is a Hessian of a strictly convex smooth function.

Example 198

If A is sparse with an upper arrow sparsity pattern (meaning it's only nonzero in the first row and column and on the diagonal), then L would end up being a full lower triangular matrix with $O(n^2)$ nonzero entries. But if we permute it so that the first row and column are now at the end, we get a lower arrow sparsity pattern and it turns out L only has nonzeros on the diagonal and the last row, which has $O(n)$ nonzeros (there is zero fill-in).

Proposition 199

Slightly generalizing from the positive definite case, any nonsingular symmetric matrix A can be written as $PLDL^TP^T$ where D is block diagonal with 1×1 or 2×2 blocks. The factorization is again $\frac{1}{3}n^3$ and again we can sparsify with a good choice of P.

We can now think about how we can do structured blocking: suppose we want to write our system $Ax = b$ as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

where $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$. If A_{11} is nonsingular, we can then solve $x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$ in terms of x_2, and then plug that back into the other equation to get

$$(A_{22} - A_{21}A_{11}^{-1}A_{12})x_2 = b_2 - A_{21}A_{11}^{-1}b_1.$$

The matrix factor on the left-hand side here is called the **Schur complement**, and it gives us a way to solve: we form the Schur complement and the right-hand side, and then solve this smaller linear system by again taking Schur complements. More precisely, we form $A_{11}^{-1}A_{12}$ and $A_{11}^{-1}b_1$ (that is, solve the linear systems with each column of A_{12} and with b_1), use them to form the Schur complement and the right-hand side, then determine x_2, then determine x_1. By the time we get to S we probably don't have any useful structure for speeding up computations, but we might be able to factor A_{11} quickly depending on its structure; if so, the entire thing is linear in n_1. More precisely, the dominant terms in the flop count are

$$f + n_2 s + 2n_2^2 n_1 + \frac{2}{3}n_2^3$$

where f is the cost of factoring A_{11}, s is the cost of the solve step (which we have to repeat n_2 times to form $A_{11}^{-1}A_{12}$), and then the last two terms are the cost of matrix multiplication and of actually inverting the Schur complement. So in situations where we have a very dense block in the corner of an otherwise banded matrix, we can do much faster than we would naively expect (and we should think of wanting to make A_{11} as big as possible while still having these nice properties).

If we don't exploit the sparsity at all, then the number of flops needed is just $\frac{2}{3}(n_1 + n_2)^3$, which means we save nothing. But if A_{11} is diagonal, it ends up just taking something like $2n_2^2 n_1 + \frac{2}{3}n_2^3$ steps, which can be much better if $n_1 \gg n_2$.

Example 200

Turning back to structure of solving, suppose we have a "structure plus low-rank" problem $(A+BC)x = b$, where A has structure so it's easy to invert, and $B \in \mathbb{R}^{n\times p}$ and $C \in \mathbb{R}^{p\times n}$ for $p < n$. Adding an extra variable actually ends up helping us in this case: define $Cx = y$, so that we can solve the "uneliminated system"

$$\begin{bmatrix} A & B \\ C & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

This is now a great situation to use block elimination – we can eliminate I of course, but that's bad because that gets us the original equation. Instead, we can eliminate A and then the Schur complement reads

$$(I + CA^{-1}B)y = CA^{-1}b,$$

at which point we can solve $Ax = b - By$. This gives us the following useful fact:

Proposition 201 (Matrix inversion lemma)

Suppose $A, A+BC$ are nonsingular. Then

$$(A+BC)^{-1} = A^{-1} - A^{-1}B(I + CA^{-1}B)^{-1}CA^{-1}.$$

For example, if we want to do sequential least-squares and do one more measurement, then A is the Gram matrix and we add a rank-one perturbation. Re-solving least-squares would take n^3 steps, but this kind of rank-one update can be much faster. These things used to be very important, but it's much less important now because of how fast our computers are.

A special case of this is the **diagonal plus low rank** form; for example for covariance matrices this would be a **factor model** in which we have a small number of underlying factors and the diagonal part is the "idiosyncratic component" (at least in finance dialect). If we actually form $D = A + BC$ where A is diagonal and B is $n \times p$, and then try to solve $Dx = b$, then again we essentially ignore the structure and have to pay n^3. On the other hand, if we use the matrix inversion lemma to first solve $(I + CA^{-1}B)y = CA^{-1}b$ and then compute $A^{-1}b - A^{-1}By$, that actually ends up being linear in n (the dominant terms are $2p^2n + \frac{2}{3}p^3$). The point is that we should never actually be forming this matrix $A + BC$ for both memory and runtime reasons.

19 February 24, 2026

We'll start **numerical methods for optimization** today; last time we just did a whirlwind tour of numerical linear algebra (how to solve sparse or banded systems, block elimination, etc.) and now we will apply that.

Example 202

We begin with **unconstrained optimization**. That is, we just want to solve the problem "minimize $f(x)$" with no other constraints, and we'll assume that f is convex and smooth (or at least twice continuously differentiable) and that the optimal value is attained.

The optimality condition we then care about is $\nabla f(x) = 0$; thus we have n equations and n unknowns. These will only be n *linear* equations if our function f is quadratic, but more generally it is going to be harder. Everytihng is built on the quadratic case though: if $f(x) = \frac{1}{2}x^T Px + q^T x + r$ for $P \succeq 0$, we need to solve exactly the system $\nabla f(x) = Px + q$ (and sometimes we know how to solve it very fast). We'll come back to this much more later.

For non-quadratic functions, the typical idea is to use **iterative methods**. That is, we generate some sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \cdots \in \text{dom } f$, from some starting point, and our goal is that $f(x^{(k)}) \to p^*$ and $\nabla f(x^{(k)}) \to 0$. We don't have a good way of observing whether we're close to p^*, but we **can** tell when the gradient is small and we often build our stopping criteria around that.

Fact 203

The algorithms we discuss here require a starting point $x^(0)$ in the domain, such that the sublevel set $S = \{x : f(x) \le f(x^{(0)})\}$ is closed. This condition is actually hard to verify in general unless all sublevel sets are closed; this is true if the epigraph is closed. Examples of this are if we have a barrier (where $f(x) \to \infty$ as x approaches the boundary of dom f).

Some examples of differentiable functions whose sublevel sets are all closed are logsumexp (whose domain is everything) and $-\sum_{i=1}^m \log(b_i - a_i^T x)$ (where we approach ∞ as any argument of any of the logs approaches 0).

Definition 204

A function f is **strongly convex** on S if there is some $m > 0$ such that $\nabla^2 f(x) \succeq mI$ for all $x \in S$.

In other words, subtracting a quadratic from f to get $f(x) - \frac{m}{2}||x||_2^2$ is still convex. In such a case, we get the Taylor expansion

$$f(y) \ge f(x) + \nabla f(x)^T (y - x) + \frac{m}{2}||x - y||_2^2;$$

in particular because the right-hand side is a convex quadratic in y and strictly positive definite, actually the sublevel sets S are bounded. So this means $p^* > -\infty$ (the problem is feasible) and

$$f(x) - p^* \le \frac{1}{2m}||\nabla f(x)||_2^2,$$

"justifying that the gradient being small is a good stopping criterion." Unfortunately, in applications, we don't really know the minimum curvature m (unless we are doing something like minimizing a loss function plus some sum-of-squares regularizer which provides the m).

Example 205

In a **descent method**, each iterate looks like $x^{(k+1)} = ax^{(k)} + t^{(k)}\Delta x^{(k)}$, where $\Delta x^{(k)}$ is called the **step** or **search direction** and $t^{(k)}$ is called the **step size** or **step length** (even if it's not the actual length); we choose it so that $f(x^{(k+1)}) < f(x^{(k)})$ for all k. The idea is that for f to be decreasing at each step, it requires $\nabla f(x)^T \Delta x < 0$. Thus, to specify a descent method, we **determine a descent direction and choose a step size**, then update $x \mapsto x + t\Delta x$.

It's useful to compare the following two step size ideas:

- In an **exact line search**, after choosing Δx we restrict the function to the line $x + t\Delta x$ and then minimize over t. This is useful if for some reason we can solve the one-dimensional problem exactly.
- In a **backtracking line search**, choose $0 < \beta < 1$ and $0 < \alpha < \frac{1}{2}$. We start by trying $t = 1$, and then repeatedly reduce the step size $t \mapsto \beta t$ until $f(x + t\Delta x) < f(x) + \alpha t \nabla f(x)^T \Delta x$. The idea is that we're okay with any

point which gives us at least, say, 10% of the reduction that we'd expect if we'd followed the tangent line, and we keep trying geometrically smaller and smaller ts until the step we take is okay.

Definition 206

In **gradient descent**, we choose the search direction to be $\Delta x = -\nabla f(x)$ and then perform some line search (exact or backtracking). We repeat this until some stopping criterion is satisfied, usually of the form $||\nabla f(x)||_2 \le \varepsilon$.

Proposition 207

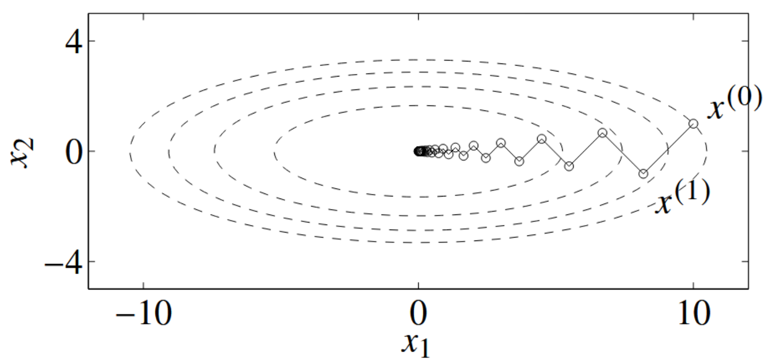
For strongly convex f, we have the convergence result

$$f(x^{(k)}) - p^* \le c^k(f(x^{(0)}) - p^*)$$

for some constant $c < 1$ which depends on m, the initial point, and the backtracking parameters. (So we get geometric decay, but it may take very long to decay.)

Example 208

Suppose we have $f(x) = \frac{1}{2}(x_1^2 + \gamma x_2^2)$ with $\gamma > 0$ (so if $\gamma = 1$ it's a perfect bowl, but if $\gamma \gg 1$ or $\gamma \ll 1$ it's much more lopsided). Exact line search started from the starting point $(\gamma, 1)$ is an outward normal to the level set, and so the diagram below shows a zig-zagging behavior for $\gamma = 10$ (because when we move tangent to our lopsided ellipsoid, we're moving at quite an angle compared to the ideal angle).

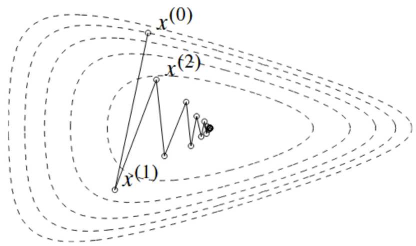


The idea is that the convergence heavily depends on the value of γ; we have

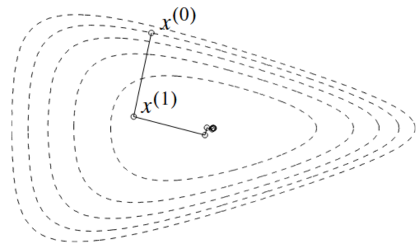
$$x_1^{(k)} = \gamma \left(\frac{\gamma - 1}{\gamma + 1}\right)^k, \quad x_2^{(k)} = \left(-\frac{\gamma - 1}{\gamma + 1}\right)^k.$$

So if γ is 1 or close to 1, it only takes a few steps for the coordinates to approach $(0, 0)$, but otherwise it takes longer.

On the other hand, for a certain nonquadratic function $e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} + e^{-x_1-0.1}$, backtracking line search initially looks like it does a lot more zig-zagging than exact line search and might be much more inefficient:

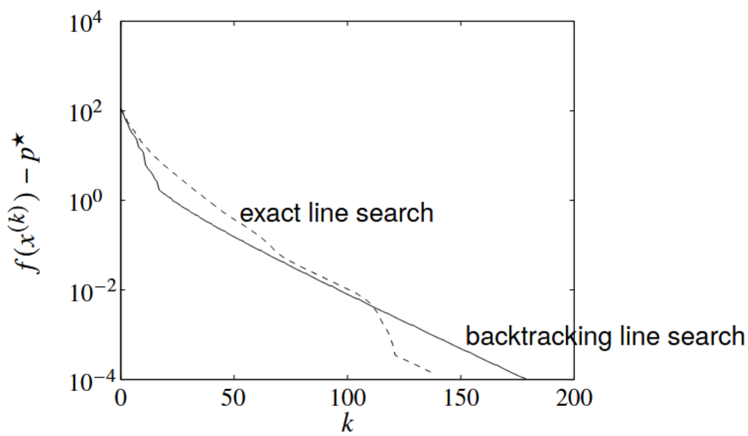


backtracking line search



exact line search

But something is quite weird here – it's actually possible for backtracking line search to beat exact line search, even though it seems like backtracking line search is meant to be an unideal approximation. Below is an example of that for a high-dimensional problem:



In practice, it actually doesn't matter at all what the parameters α, β are. The main feature of the plot is that on a semilog plot (meaning the distance to optimality is log-scaled), the we have roughly linear behavior, so we actually are getting geometric reduction in the gap and we call this **linear convergence**.

So we see that gradient descent can be very slow, and it works well when the sublevel sets look spherical. And spherical means that "every direction we look, things are the same." Thus we might ask for something different even if $-\nabla f(x)$ seems like the most natural choice. The point is that $-\nabla f(x)$ is **only the steepest descent in the Euclidean norm**, since we're inherently dividing by how far we moved.

Definition 209

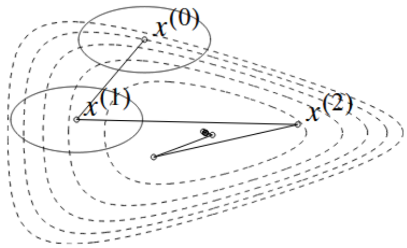
In the steepest descent method, we pick a norm $||\cdot||$ and choose the normalized steepest descent direction

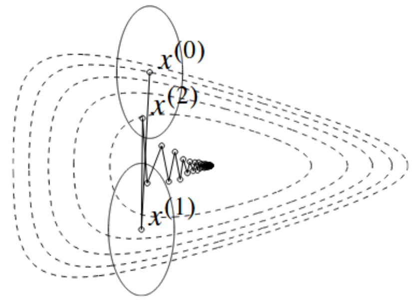
$$\Delta x_{\text{nsd}} = \operatorname{argmin}\left\{\nabla f(x)^T v : ||v|| = 1\right\}.$$

That is, the direction is the unit-norm step with the most negative directional derivative.

The **steepest descent method** (using descent with the normalized steepest descent direction) is then gradient descent if we use the Euclidean norm. If we instead use the quadratic norm $(x^T P x)^{1/2}$, then $\Delta x_{\text{sd}} = -P^{-1}\nabla f(x)$ (slightly skewed away from the gradient). If we instead use the ℓ^1 norm, we get **coordinate descent** with $\Delta x_{\text{sd}} = -\left(\frac{\partial f(x)}{\partial x_i}\right) e_i$, where i is the coordinate with the largest magnitude of ∇f. And if we use the ℓ^∞ norm, all that matters is whether each coordinate is $+1$ or -1 depending on the sign of ∇f.

The point is that we can improve the coefficient for convergence if we choose a better norm. **If we choose our norm sublevel sets to look like the shape of the function's sublevel sets**, then we'll converge much faster than if we choose something poorly aligned, as shown below.





Another interpretation of this for the quadratic case is that steepest descent with P is like transforming the problem by $P^{1/2}$. So we could do something like pick a bunch of random Ps and see which one converges fastest, but unfortunately there's far more bad Ps in general and it's hard to find the good ones. Thus we'll focus on yet another idea which lets us essentially dynamically pick P:

Definition 210
The **Newton step** uses the search direction

$$\Delta x_{\text{nt}} = -\nabla^2 f(x)^{-1} \nabla f(x).$$

(This indeed has a negative inner product with $\nabla f(x)$.) The interpretation of this is that it minimizes the local second-order approximation $\hat{f}(x+\nu) = f(x) + \nabla f(x)^T + \frac{1}{2}\nu^T \nabla^2 f(x)\nu$, and near the minimum of any function it's quadratic. So because the sublevel sets will be almost ellipsoids near the optimum, it's a reasonable approximation (attempt) to **dynamically update the metric in which we do steepest descent** to match what we hope is the actual curvature of our function. Yet another interpretation is that $x + \Delta x_{\text{nt}}$ is exactly satisfying the linearized optimality condition $\tilde{\nabla} f(x+\nu) = 0$ (where $\tilde{\nabla}$ is the first-order approximation for the gradient). And because minimizing a quadratic is just solving linear equations, we can actually do this step pretty efficiently.

Remark 211. *This last interpretation is exactly the "Newton's method" we might be familiar with for approximating the solution to an equation $f(x) = 0$, where we repeatedly take the tangent line and see where it intersects the x-axis.*

Newton's method was actually abandoned back in the 1960s, partially because computers were far less powerful and this method requires storing the Hessian (which means we need $O(n^2)$ space already), and partially because people didn't really know numerical linear algebra and how to exploit the special structure of the Hessian in particular cases (to get a good Cholesky factorization, for example). And we could also just compute the Hessian every 10 steps or so to amortize the effort of computing it. We'll say more about all this soon.

Definition 212
The **Newton decrement** is defined to be

$$\lambda(x) = (\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x))^{1/2}$$

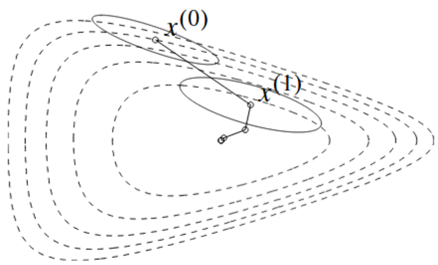
The idea is that this is a second-order approximation of what p^* is, and it gives us an approximation of the gap $f(x) - p^*$ via the quadratic approximation $\hat{f}$:

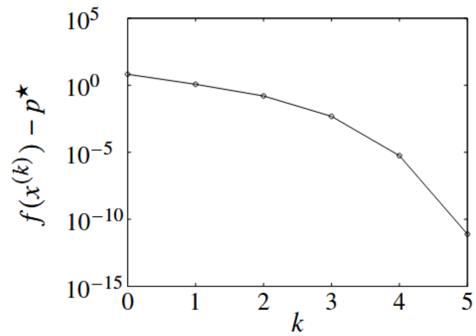
$$f(x) - \inf_y \hat{f}(y) = \frac{1}{2}\lambda(x)^2.$$

Thus our stopping criterion in Newton's method is typically to quit if $\frac{\lambda^2}{2} \le \varepsilon$.

An affine change of coordinates in a gradient method greatly affects the convergence rate (since it's like using a different P). But **Newton's method is invariant under changes of affine coordinates** – using $f(Ty)$ instead just means we get $y^{(k)} = T^{-1}x^{(k)}$ for all k – and in particular this means it gives us a more coordinate-independent measure of how close we are to x^* than $||\nabla f(x)||_2$.

Taking the nonquadratic example from before, Newton's method, using backtracking parameters $\alpha = 0.1, \beta = 0.7$, actually gives us **quadratic local convergence** (accelerating convergence rather than linear on a log plot) to machine precision in just 5 steps. Intuitively, this may make sense because the sublevel sets get closer and closer to ellipsoidal as we approach the optimum, so we'll get really good estimates. (Real applications really only need maybe two or three iterations before we're good enough.)





And for our $\mathbb{R}^{100}$ example, we see that backtracking line search and exact line search are pretty close even with the simplification in backtracking search, and backtracking actually eventually just accepts $t = 1$ every single time so that we just use the Newton step every time. Even in high dimensions, it's very interesting that the behavior is very similar – with sparse coefficients, our minimization converges in maybe 20 steps (it only needed that many queries for directions) even in tens of thousands of variables.

Example 213

We'll now think a bit about classical convergence analysis for Newton's method. The point is that if the second derivative of our function changes slowly (so if the third derivative is small), Newton's method works well. Of course, the third derivative is a weird object to think about and we probably don't want to do that too much – we just think about Lipschitz constants on the Hessian instead.

We'll assume that f is strongly convex with some constant m, and that we have the Lipschitz condition

$$||\nabla^2 f(x) - \nabla^2 f(y)||_2 \le L||x - y||_2.$$

The point is that there are usually two phases: the **damped Newton phase** where $||\nabla f(x)||_2 \ge \eta$ and the **quadratically convergent phase** where $||\nabla f(x)||_2 < \eta$. In the former, the function value will decrease by at least $-\gamma$ at each step, so we are significantly improving our optimization. That can only happen so many times before we get to p^*, and once we reach the latter phase we have that $||\nabla f(x^{(k+1)})||_2$ is less than a constant times $||\nabla f(x^{(k)})||_2$ (the number of accurate digits basically doubles every time, instead of just getting a constant number more accurate digits each time!).

It turns out that in the damped Newton phase, most iterations will require backtracking steps, but in the quadratically converging phase, basically all step sizes will be $t = 1$ and we're essentially two or three steps away.

Proposition 214

The number of iterations of Newton's method until $f(x) - p^* \le \varepsilon$ is bounded by $\frac{f(x^{(0)} - p^*}{\gamma} + \log\log(\varepsilon_0/\varepsilon)$, where γ, ε_0 are constants depending on $m, L, x^{(0)}$.

In particular, the last term is essentially constant for every practical purpose ever, and this seems like a great result. But unfortunately in practice, p^* isn't known, and furthermore the constants γ, ε_0 are usually unknown (because they depend on the Lipschitz constant and strong convexity). So this gives us qualitative insight but not actual quantitative bounds, and even if we have those bounds they are usually extremely far from expectations – we have to make some very strong assumptions and get very vague results. Furthermore, **the bound we get out of the analysis is actually not affinely invariant**, since m and L do depend on affine changes. So the numerical implementation is more elegant than the mathematical analysis!

Luckily, we will now see convergence analysis that is actually useful:

Example 215

We'll now see an analysis of Newton's method for an interesting concept from the 80s called **self-concordant functions**. This is work of Nesterov and Nemirovski which does not depend on unknown constants and gives an affine-invariant bound, though it only applies to a specific class of convex functions. This will also come up in the interior-point methods we will study later on in the course.

Definition 216

A convex function $f : \mathbb{R} \to \mathbb{R}$ is **self-concordant** if $|f'''(x)| \le 2f''(x)^{3/2}$ for all $x \in \operatorname{dom} f$ (The classical analysis would have just used $|f'''(x)| < L$, but that's not affine-invariant.) A convex function $f : \mathbb{R}^n \to \mathbb{R}$ is self-concordant if $g(t) = f(x + tv)$ is self-concordant for all $x \in \operatorname{dom} f$ and $v \in \mathbb{R}^n$.

(We have to use a comparison to f'' instead of to f', because for convex functions we know f'' is actually positive.) Some examples of self-concordant functions on $\mathbb{R}$ are linear and quadratics, $-\log x$, and $x \log x - \log x$, and it's nice to know that the property is invariant under certain transformations, like pre-compositions with affine functions. In particular, $f(X) = -\log\det X$ on S^n_{++} is self-concordant (and we should not think too hard about what the third derivative of this function actually is, because it has to have six indices).

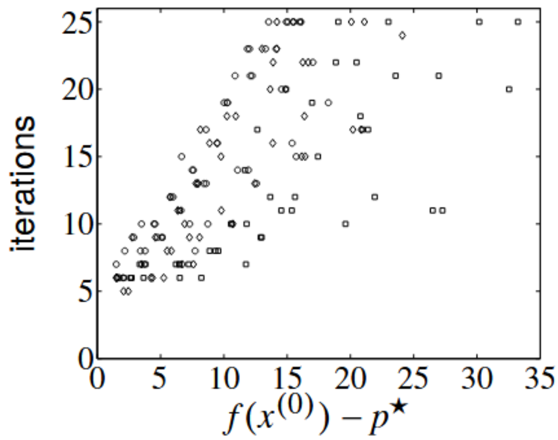
Proposition 217

For self-concordant functions, there exist constants $\eta \in (0, \frac{1}{4}]$, $\gamma > 0$ such that the number of Newton iterations is bounded by $\frac{f(x^{(0)}) - p^*}{\gamma} + \log_2 \log_2 \frac{1}{\varepsilon}$, where γ can be computed explicitly.

For example, if $\alpha = 0.1, \beta = 0.8, \varepsilon = 10^{-10}$, this bound actually evaluates to $375(f(x^{(0)}) - p^*) + 6$, and that constant can be improved with more work. And even if we don't know p^*, we can still just get a lower bound on it via duality, so really nothing is missing here!

Example 218

In the plot below, randomly generated instances of functions of the form $f(x) = -\sum_{i=1}^M \log(b_i - a_i^T x)$ were generated, and the number of actual steps it took (for various values of m, n) are plotted. So it looks like empirically we can really take something like $c = 2$ and still get a bound of the form $c(f(x^{(0)} - p^*) + 6$.



Fact 219

Finally, we'll discuss how to actually implement all of this. The main cost of this whole thing is that we need to evaluate derivatives to calculate the Hessian and then solve the Newton system $H\Delta x = -g$ for $H = \nabla^2 f(x)$ and $g = \nabla f(x)$. We usually do this via Cholesky factorization and then solve via substitution; this costs $\frac{1}{3}n^3$ flops if the system is unstructured, but far less if H is sparse or banded or has other structure.

Notice that H being banded (meaning that it's zero outside of a narrow strip around the diagonal) means there are only local interactions; this happens a lot for example in control systems where outputs only depend on the previous output and input.

For example if our objective is separable plus some low-rank thing, meaning that $f(x) = \sum_{i=1}^n \psi_i(x_i) + \psi_0(Ax+b)$ for $A \in \mathbb{R}^{p \times n}$ for $p \ll n$, then we can make use of this structure since the Hessian is low-rank plus diagonal. We can then do one iteration in far fewer than $\frac{1}{3}n^3$ flops by factoring $H_0 = L_0 L_0^T$ and then solving the resulting system.

So the point is to make some links in our head between "structure on the Hessian" and "efficient numerical linear algebra," and we'll be doing much more of that throughout this course.

Next time, we'll start considering the case of **equality-constrained minimization** where we minimize $f(x)$ subject to $Ax = b$ for some $A \in \mathbb{R}^{p \times n}$. We'll specifically be trying to satisfy the optimality condition $\nabla f(x^*) + A^T \nu^* = 0$, $Ax^* = b$, rather than just $\nabla f(x^*) = 0$, and we've already seen that these dual variables ν^* might actually be interesting to us too.

20 February 24, 2026 (Problem Session)

We'll continue to work through some applications of convex optimization in useful contexts:

Problem 220

Suppose we have m data points $u_1, \cdots, u_m \in \mathbb{R}^n$, and we want to fit the data to a sphere $\{x \in \mathbb{R}^n : ||x - c|| = r\}$ (where we get to pick the center c, as well as the radius r). The cost function (criterion) will be to minimize the square of the difference of squares

$$\sum_{i=1}^m \left(||u_i - c||^2 - r^2\right)^2.$$

The picture we should keep in mind is that in two dimensions, we have lots of points in the xy-plane, and we want to fit a circle of an appropriate radius so that all points are of norm roughly r away from the center. But the penalty function is a bit funny, since we're comparing the *squared* distance to r^2 and then further squaring that difference.

Defining the error vector to have components $\mathcal{E}_i = r^2 - ||u_i - c||^2$, we are trying to minimize $||\mathcal{E}||^2$. We can write this squared norm out so that

$$\begin{aligned}\mathcal{E}_i &= r^2 - \langle u_i - c, u_i - c\rangle \\ &= r^2 - ||u_i||^2 + 2\langle u_i, c\rangle - ||c||^2.\end{aligned}$$

To simplify some of the annoyances, we can change variables $t = r^2 - ||c||^2$ and rewrite $b_i = ||u_i||^2$, so that

$$\mathcal{E}_i = 2u_i^T c + t - b_i.$$

Now we want to optimize over (c, t) instead of (c, r), and this trick is only allowed if at the very end, we can indeed recover the optimal r^* from c^* and t^*. Since $r^2 = t + ||c||^2$, the recovery must be that $r = \sqrt{t + ||c||^2}$, and thus we require that $t^* + ||c^*||^2 \geq 0$. The reason that we make this transformation is that the original function is not convex in the variables (c, r), but this new function is actually convex in (c, t). One way to see this is to write the error vector in matrix form as

$$\mathcal{E} = 2Uc + t\mathbf{1} - b = \begin{bmatrix} 2U^T & \mathbf{1} \end{bmatrix} \begin{bmatrix} c \\ t \end{bmatrix} - b,$$

where the columns of U are the fitted data points u_i. The point is that $\mathcal{E}$ is affine in our optimization variables, and thus $||\mathcal{E}||^2$, our objective, is indeed convex. That means we can find c^*, t^* via convex optimization.

The only thing left to do (theoretically) is to verify that condition $t^* + ||c^*||^2 \geq 0$. One way we can do this is that we have a standard least-squares problem where we want to minimize $||Ax - b||^2$ (for $A = \begin{bmatrix} 2U^T & \mathbf{1} \end{bmatrix}$), and thus actually we know that $x^* = (A^T A)^{-1} A^T b$ and therefore

$$\begin{aligned}p^* &= ||Ax^* - b||^2 \\ &= \left|\left| \begin{bmatrix} 2U^T & \mathbf{1} \end{bmatrix} \begin{bmatrix} c^* \\ t^* \end{bmatrix} - b \right|\right|^2 \\ &= \left|\left| t^*\mathbf{1} - (b - 2U^T c^*) \right|\right|^2.\end{aligned}$$

But each of the vectors $t^*\mathbf{1}$ and $b - 2U^T c^*$ lives in $\mathbb{R}^m$, and t is unconstrained, so this is actually a **linear algebra projection**: $t^*\mathbf{1}$ should be exactly the orthogonal projection of $b - 2U^T c^*$ onto the all-ones subspace. Therefore

$$\begin{aligned}0 &= \langle \mathbf{1}, t^*\mathbf{1} - (b - 2U^T c^*)\rangle \\ &= mt^* - \mathbf{1}^*(b - 2U^T c^*),\end{aligned}$$

and so t^* can be written in terms of c^*; recalling the definitions of b and U,

$$\begin{aligned}t^* &= \frac{1}{m}\mathbf{1}^*(b - 2U^T c^*) \\ &= \frac{1}{m}\left(\sum_{i=1}^m ||u_i||^2 - 2\sum_{i=1}^m u_i^T c^*\right).\end{aligned}$$

So to check our condition, we can just substitute back in:

$$\begin{aligned} t^* + ||c^*||^2 &= \frac{1}{m}\left(\sum_{i=1}^m ||u_i||^2 - 2\sum_{i=1}^m u_i^T c^*\right) + ||c^*||^2 \\ &= \frac{1}{m}\left(\sum_{i=1}^m ||u_i||^2 - 2\sum_{i=1}^m u_i^T c^* + \sum_{i=1}^m ||c^*||^2\right) \\ &= \frac{1}{m}\left(\sum_{i=1}^m ||u_i - c^*||^2\right). \end{aligned}$$

Now this right-hand side is an average of squares, so it will indeed be nonnegative and we have verified the condition that we need.

Once we've set up the problem mathematically, converting it to CVXPY code is not too bad: we can use our given data points to form the data matrix $A = \begin{bmatrix} 2U^T & \mathbf{1} \end{bmatrix}$ and the vector $b = \begin{bmatrix} ||u_1||^2 \\ \vdots \\ ||u_m||^2 \end{bmatrix}$, and then we minimize `cp.norm2(Ax - b)`. The result then allows us to recover the center c and the radius r. Alternatively, we can stick with something more closely resembling our original objective $\sum_{i=1}^m (||u_i||^2 - 2u_i^T c - t)^2$ by representing $||u_i||^2$ as `cp.norm` over one of the two axes of our data matrix U. And there are no constraints here – we can't actually specify that $t + ||c||_2^2 \geq 0$ in a convex optimization problem by DCP rules, and luckily we don't need to pass it in because we've already proven that it will be satisfied for the optimal point.

Remark 221. *Notice that c^* will not be anything like the empirical mean of our data in general, and this isn't even a symptom of our specific choice of loss. For example, if we had lots of points on the boundary of the left half of a circle but none on the right, the empirical mean would skew heavily towards the left of the center c^*, even if the fit is exactly perfect.*

If we instead fit a circle only in terms of the empirical mean and covariance, we would get a different kind of fit for a covariance ellipse which has to do with how many of the points are inside the ellipse – that's meant for a totally different question for a different setting of Gaussian distributed data.

Problem 222

We'll now talk a bit about gradient descent, specifically thinking about steepest descent under different norms.

In the one-dimensional case, the core update has to do with the fact that

$$f(x + \delta) = f(x) + \frac{d}{dx} f(x)\delta,$$

where depending on the sign of d, we either move in the positive or negative direction to try to minimize our function. (If the slope is negative, then we want d to be positive to move downward, and similarly if the slope is positive we want d to be negative.)

In general if we're minimizing $f : \mathbb{R}^n \to \mathbb{R}$, our goal is to pick a vector d that makes the directional derivative go down in a steeply negative way. In ordinary gradient descent we pick $-\nabla f(x)$ itself, but because the multi-dimensional linear approximation is $f(x + \delta) = f(x) + \nabla f(x)^T \delta$, we can pick anything with $\langle \nabla f(x)^T, \delta \rangle < 0$, and in the **steepest descent** approach we pick the δ which minimizes this over all δ in the unit ball (for some norm):

$$\delta_{\text{nsd}} = \text{argmin}_\delta \left\{ (\nabla f(x))^T \delta : ||\delta|| = 1 \right\}.$$

Depending on the norm we use, we'll get different answers for the best possible d.

1. If we use the Euclidean norm, we are trying to solve the problem

$$\begin{aligned} \text{minimize} \quad & (\nabla f(x))^T \delta \\ \text{subject to} \quad & ||\delta||_2 = 1. \end{aligned}$$

   But we see an inner product here, so Cauchy-Schwarz is a natural tool: one side of the inequality tells us that

$$\begin{aligned} (\nabla f(x))^T \delta &\geq -||\nabla f(x)||_2 \cdot ||\delta||_2 \\ &= -||\nabla f(x)||_2, \end{aligned}$$

   with equality if and only if the two vectors point in exactly opposite directions. Thus we see that we should pick δ to be $-\nabla f(x)$, normalized to have norm 1.

2. Closely related to the 2-norm is the problem

$$\begin{aligned} \text{minimize} \quad & (\nabla f(x))^T \delta \\ \text{subject to} \quad & ||\delta||_P = 1, \end{aligned}$$

   where $||\delta||_P = \sqrt{d^T P d}$ for a positive definite symmetric matrix P. We can work this out ourselves, but we just end up getting a certain linear transformation of the gradient.

3. Next, consider the ℓ^1 norm, meaning that we want to instead solve

$$\begin{aligned} \text{minimize} \quad & (\nabla f(x))^T \delta \\ \text{subject to} \quad & ||\delta||_1 = 1. \end{aligned}$$

   This means we're allowed to take any convex combination of the coordinates of $\nabla f(x)$ or its negative (where δ dictates the coefficients we use), and we want the result to be as negative as possible. But for example if $\nabla f(x) = (3, 1)$, we should choose $\delta = (-1, 0)$ and put all of our combination into the largest coordinate – in particular, we shouldn't always be picking things in the direction of $-\nabla f$ this time! So δ will always be a basis vector e_i or its negative, with i chosen to be the coordinate with largest magnitude in the gradient.

4. Finally, we can consider the ℓ^∞ norm, and that's left as an exercise to us. But the logic is basically the same – we interpret the inner product in some way that lets us pick the best coordinates of δ, and the difference is that $||\delta||_\infty = 1$ means each coordinate independently just has to lie in $[-1, 1]$.

21 February 26, 2026

We'll continue with equality-constrained minimization today. Like in the last lecture, we'll assume we're minimizing a smooth convex function, subject to some linear equality constraints $Ax = b$; this is the same as solving the optimality conditions

$$\nabla f(x^*) + A^T \nu^* = 0, \quad Ax^* = b,$$

where $Ax^* = b$ is a linear equation but $\nabla f(x^*)$ is nonlinear in general and so we can't just solve directly with our linear algebra techniques.

Example 223

The special case where we minimize a quadratic $\frac{1}{2}x^T Px + q^T x + r$ (for P positive semidefinite) is simpler because ∇f is affine, and the optimality conditions reduce to a system of linear equations

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \nu^* \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}.$$

The matrix on the left-hand side is called the **KKT matrix**, and it is nonsingular if and only if $Ax = 0$ for some nonzero x implies that $x^T Px > 0$ (that is, P is positive definite on the nullspace of A). Equivalently, this requires that $P + A^T A \succ 0$ (in every direction in $\mathbb{R}^n$, we're either penalized by P or by A). So this case reduces to linear algebra, and not surprisingly, just like in last lecture, we'll solve a sequence of problems of this form by approximating the objective as a quadratic and minimizing it.

Proposition 224

The first thing we can do is to eliminate equality constraints by writing in **free-parameter form**, where the feasible set $\{x : Ax = b\}$ can be rewritten as a range $\{Fz + \hat{x} : z \in \mathbb{R}^{n-p}\}$, where $\hat{x}$ is any "particular" solution of $Ax = b$ and F is some matrix whose range is the nullspace of A. (This can be obtained by QR factorization or many other methods.)

Once we've written the problem this way, we end up with a reduced problem where we just minimize $f(Fz + \hat{x})$ over z, which is an unconstrained problem we can solve via Newton's method or something else. We can then recover x^*, ν^* from our solution z^* via

$$x^* = Fz^* + \hat{x}, \quad \nu^* = -(AA^T)^{-1} A \nabla f(x^*).$$

So we might ask why we don't just always do this; mathematically it is an option, but this turns out to not always be a good idea.

Example 225

In optimal resource allocation, suppose we allocate some resource amount x_i to the ith agent, and our goal is to minimize the total cost $\sum_i f_i(x_i)$ subject to $x_1 + \cdots + x_n = b$. Then we can eliminate the equality constraint by just using $x_n = b - x_1 - \cdots - x_{n-1}$, and we can use the particular solution $\hat{x} = be_n$ and using the matrix $F = \begin{bmatrix} I \\ -\mathbf{1}^T \end{bmatrix} \in \mathbb{R}^{n \times (n-1)}$.

Notice that in this case, the Hessian of the original objective is diagonal, so if we use some smart linear algebra we can solve things a lot faster than if we use a generic (non-sparse) solver. And the Hessian of the new reduced objective $f_1(x_1) + \cdots + f_{n-1}(x_{n-1}) + f_n(b - x_1 - \cdots - x_{n-1})$ is not necessarily sparse, but it is diagonal plus rank-one, so again our methods can do that pretty efficiently.

Looking at Newton's method now (when we don't write things in free-parameter form), suppose x is feasible, and now we're going to add something to it that keeps it feasible (hence must be in the nullspace of A). So the Newton step looks like the solution v of

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}.$$

We can call v the "primal step" and w the "dual step"; the interpretation is that the Newton step Δx_{nt} exactly solves the second-order approximation of our problem

$$\begin{aligned} \text{minimize} \quad & \hat{f}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v \\ \text{subject to} \quad & A(x+v) = b. \end{aligned}$$

So we're basically taking a quadratic we can solve and getting the linearized optimality conditions.

The Newton decrement we get out of this minimization is then

$$\lambda(x) = (\Delta x_{\text{nt}}^T \nabla^2 f(x) \Delta x_{\text{nt}})^{1/2} = (-\nabla f(x)^T \Delta x_{\text{nt}})^{1/2}.$$

Remember that in the unconstrained case, the Newton decrement has to do with how much we go down in the quadratic approximation (a good guess of how close we are to the minimum, assuming the function were quadratic). It's exactly the same here; we have

$$f(x) - \inf_{Ay=b} \hat{F}(y) = \frac{\lambda(x)^2}{2}.$$

(But note that the formula from last lecture $\lambda(x) = (\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x))^{1/2}$ is not true anymore.) So if we do the exact same Newton's method via descent, by choosing the Newton step and then choosing the step size via backtracking line search, we have an algorithm. And again, this is affine-invariant and gives us a feasible descent method.

Fact 226

It turns out that a weird thing happens here: we have two ways of solving via Newton's method, by either eliminating the equality constraints and using unconstrained Newton's method, or by doing what we just described with explicit equality constraints. Actually both methods are exactly identical and we get the same exact steps, so no additional convergence analysis is needed at all.

Example 227

We're now brought to the **infeasible start Newton method**, which we'll have a chance to try implementing ourselves. Suppose we're at some point $y = (x, \nu)$, where we think of x as the primal variable and ν as the dual variable. The optimality condition can then be written as

$$r(y) = (\nabla f(x) + A^T \nu, Ax - b) = 0.$$

(We call the two components here the **dual and primal residuals**, respectively.)

Previously, we assumed we start from a feasible point x, so the primal residual $Ax - b$ is always zero. But now we assume x is just some arbitrary point we start with, and we want something where $r(y) = 0$. What we can do is to linearize the condition $r(y) = 0$, which tells us the **infeasible** or **primal-dual Newton step** to take if we want to modify both x and ν simultaneously. For this, we approximate $r(y + \Delta y) \approx r(y) + Dr(y)\Delta y$ for Dr the Jacobian matrix, and the result is

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{\text{nt}} \\ \Delta \nu_{\text{nt}} \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + A^T \nu \\ Ax - b \end{bmatrix}.$$

Notice that the second part of this is affine, so taking step size 1 here will actually make us exactly primal feasible after just one step (because now $A(x + \Delta x_{\text{nt}}) = b$). So now this gives us the **infeasible start Newton method**: for any $x \in \text{dom } f$, we compute the primal-dual Newton step, and then we do backtracking line search. Note that this is **not a descent method** in that $f(x^{(k+1)}) > f(x^{(k)})$ is possible, since we might have started from an infeasible point so we're actually below f^* to begin with! Instead, we will be doing line search on the norm of the primal-dual residual $||r||_2$. If we ever pick $t = 1$, we're now primal feasible, and we're basically back to something like our previous Newton step algorithm but with some differences.

Also, it's important to keep in mind that we should always take steps that keep us in the domain of f; if a step would take us outside $\text{dom } f$, then we should multiply the step size by β and try again.

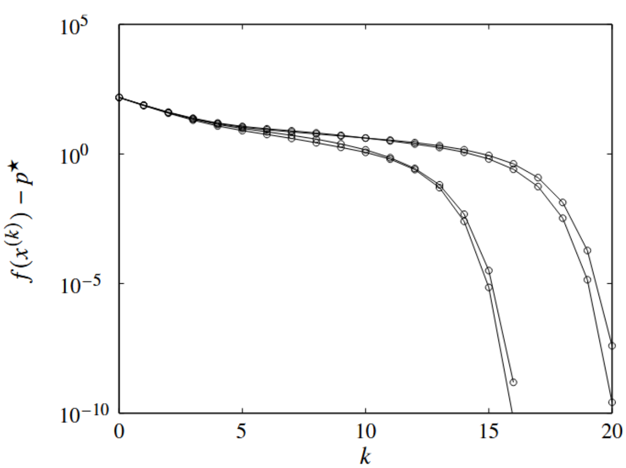
Remark 228. *Talking now about how we actually solve these systems, there's nothing wrong with just using a generic sparse LDL^T factorization; it depends on how big the 0 in our KKT system is, but usually we have not that many equality constraints. But if H is nonsingular and easily inverted (e.g. banded), we can use block elimination and do things a bit faster.*

Example 229

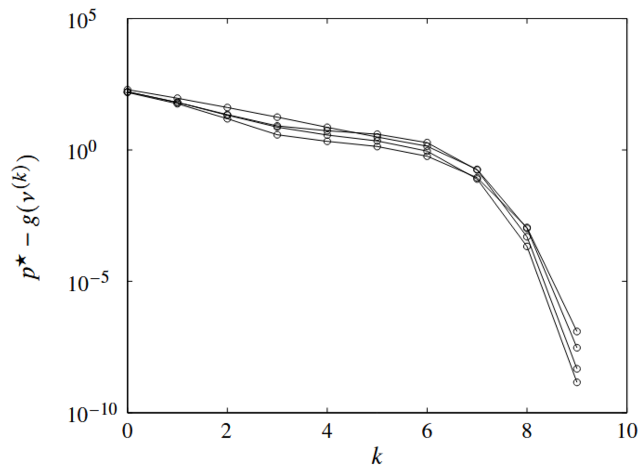
These algorithms sound very close, but when we're not feasible they're a bit different. We'll consider the following example of analytic centering, where we want to minimize $-\sum_{i=1}^n \log x_i$ subject to $Ax = b$. The dual problem then asks us to maximize $-b^T\nu + \sum_{i=1}^n \log(A^T\nu)_i + n$, and we recover x^* as $x_i^* = \frac{1}{(A^T\nu)_i}$. (Note that this dual problem might look simpler because we may have fewer variables, but that doesn't turn out to be the right conclusion.)

We can solve this in three ways: Newton's method with equality constraints, Newton's method (unconstrained) for the dual problem, and the infeasible start Newton method. These three strategies have different requirements for initialization. Feasible Newton requires some positive xs with $Ax = b$, which is actually quite hard without a linear program (which is much harder than solving this problem). Newton for the dual problem requires us to start with some ν such that $A^T\nu$ has all positive entries, but that again requires some kind of linear program. But infeasible Newton just starts with any initial point with positive entries, so we can just start from the all-ones vector.

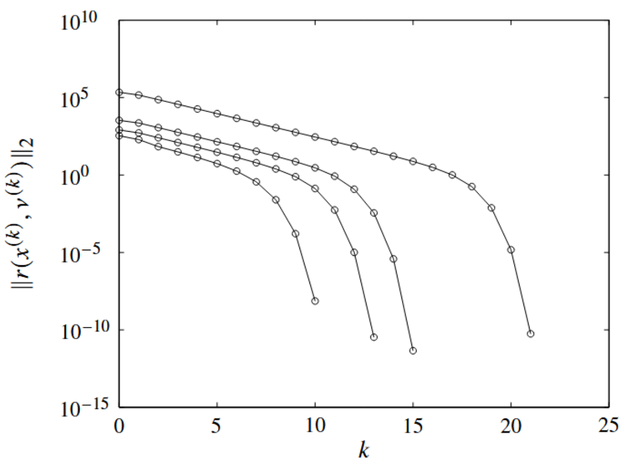
For a sense of scale, take $A \in \mathbb{R}^{100\times 500}$. It seems like we have 600 variables (primal plus dual) and so $O(n^3)$ tells us that we should try to solve the dual problem because that only has 100 variables involved, since that'll save us a factor of 6^3. But importantly this is **not** how to think about things. With four different initial conditions, we get the following trajectories for Newton's method with equality constraints:



Notice that we get quadratic convergence and it takes something like 16 to 20 steps even in 500 dimensions, which is impressive! Next, here's what we get with the dual problem:



This looks like it might be doing even better, since we only need about half as many steps as before and are solving a smaller system of equations! We'll see soon that this isn't true, though. And finally, the infeasible start Newton method looks like this, again taking something like 10 or 20 iterations. (We plot the residual here instead of the gap, but we still see the classic quadratic convergence.)



But now let's think about this through a linear algebra lens.

- In the feasible Newton method, we can first use block elimination to solve the KKT system (because the second derivative of $\log x$ is $-\frac{1}{x^2}$)
$$\begin{bmatrix} -\text{diag}(x)^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ w \end{bmatrix} = \begin{bmatrix} \text{diag}(x)^{-1}\mathbf{1} \\ 0 \end{bmatrix},$$
which is in block arrow form and thus can be done faster than naively; this reduces to just solving $A\text{diag}(x)^2 A^T w = b$, and the dominant cost is actually just the nm^2 factor (for $n \geq m$) from forming the matrix and the m^3 factor of solving becomes negligible. Luckily, a lot of modern computers are specifically good at multiplying matrices anyway.

- In the Newton method for the dual problem, we have to evaluate the Hessian, and we end up needing to solve a similar equation $A\text{diag}(A^T\nu)^{-2}A^T\Delta\nu = -b + A\text{diag}(A^T\nu)^{-1}\mathbf{1}$. So again this requires a big $O(nm^2)$ multiplication anyway, so it's not actually any faster because matrix multiplication is still dominant.

- Finally, in the infeasible start Newton method we again use block elimination, and it's almost identical to the KKT system from above.

So in all cases, even if everything looks wildly different, looking under the hood **it all ends up being equivalent if we do our linear algebra correctly**: all of them at a high level are just solving $ADA^T w = h$ for D a positive diagonal matrix, and the solve step is actually almost zero cost. So the only difference is the initialization, and in our example clearly the infeasible start method is the best thing to do.

Fact 230

We might ask what happens if our infeasible start never becomes feasible, meaning that we never take a step size of 1. In such situations, we can usually code up something that detects infeasibility and gets a certificate via duality (or in practice, we can say that the problem is infeasible if after like 50 steps, we still haven't taken a step size of 1); indeed, the proof is essentially that if we weren't feasible to begin with, we would never be able to converge. The certificate that $Ax = b$ is infeasible is to specify some w such that $w^T A = 0$ but $w^T b = 1$, which we get via dual variables.

Remark 231. *In basically all applications, we don't need more than three significant digits of accuracy in what we return because our models aren't really any better than that. But the reason it's good to deliver eight significant digits is that a user can type in a problem where things are totally different orders of magnitude in an inappropriate way, and then even after scaling back it'll still work. And luckily, quadratic convergence means we don't need to do a tradeoff with extra time taken, since we can just do two more steps and it'll be good enough.*

Example 232

For another practical example, consider the problem of network flow optimization, where we have a directed graph with $p + 1$ nodes and n arcs and there is some strictly convex flow cost function for each arc ϕ_i. Define the **incidence matrix** $\tilde{A} \in \mathbb{R}^{(p+1)\times n}$, where $\tilde{A}_{ij} = 1$ if arc j leaves node i and $\tilde{A}_{ij} = -1$ if arc j enters node i. The **reduced incidence matrix** A removes the last row of $\tilde{A}$; this matrix has rank p if the graph is connected. **Flow conservation** or **circulation** then specifies that $Ax = b$ for some (reduced) source vector b. Our goal is then to minimize the flow function $\sum_{i=1}^n \phi_i(x_i)$ subject to $Ax = b$.

In such a problem, we should immediately visualize the KKT system; the Hessian is diagonal, and every column of A has no more than two entries so things are very sparse there as well. We need to solve

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = - \begin{bmatrix} g \\ h \end{bmatrix},$$

and when we solve this via elimination we get something involving the **Laplacian matrix** of the directed graph

$$AH^{-1}A^T w = h - AH^{-1}g, \quad v = -H^{-1}(g + A^T w).$$

It turns out that $AH^{-1}A^T$ is also sparse; it only has nonzero entry (i, j) if nodes i, j are connected by an arc. And there's really a whole field dedicated to solving things involving Laplacians; we use something called "diagonally pre-conditioned CG."

Example 233
Next, consider the analytic centering problem for a linear matrix inequality, where we want to minimize $-\log\det X$ subject to $\operatorname{tr}(A_i X) = b_i$ for $1 \le i \le p$. (If X is the covariance matrix of a centered Gaussian, this is the maximum-entropy distribution subject to specifying variances in certain directions.) The optimality conditions here are that (our variable here is X)

$$X^* \succ 0, \quad -(X^*)^{-1} + \sum_{j=1}^{p} \nu_j^* A_i = 0, \quad \operatorname{tr}(A_i X^*) = b_i \text{ for } 1 \le i \le p.$$

The Newton step we end up needing to solve at a feasible X comes from the linear approximation $(X + \Delta X)^{-1} \approx X^{-1} - X^{-1}(\Delta X)X^{-1}$, and it looks like

$$X^{-1}(\Delta X)X^{-1} + \sum_{j=1}^{p} w_j A_i = X^{-1}, \quad \operatorname{tr}(A_i \Delta X) = 0 \text{ for } 1 \le i \le p.$$

So we can figure out the Newton step by solving a system of linear equations and we have $\frac{n(n+1)}{2} + p$ variables between ΔX and w. But we can simplify this by eliminating ΔX from the first equation and substituting it into the other equation (this is basically taking the Schur complement), and we get a dense (positive-definite) set of linear equations in just the variable $w \in \mathbb{R}^p$. If we work through this, we get a flop count of $pn^3 + p^2 n^2$, compared to the naive method of $(n^2 + p)^3$.

So here's where we are in the course now: next week, we'll discuss **interior-point methods**. So far, we can solve **equality-constrained smooth convex minimization problems** by reducing to solving a sequence of convex quadratic minimizations via linear algebra. But now we'll add in inequality constraints, and we'll do this by reducing the solution of an inequality constrained convex problem to a sequence of smooth minimizations with equality constraints, which we can again do by linear algebra. And actually, CVXPY is doing a lot of reductions to take our (non-smooth) problems that we give it and turning it into smooth problems which allow us to use these interior-point methods!

22 March 3, 2026

Since the final exam is coming up, it's worth noting that there will be some CVXPY questions, just to test if we know how to form problems and do basic syntax things. And in general, we'll have some problems similar to the midterm in style, each of which shouldn't be taking us significantly longer than 15 minutes.

As promised, we'll cover interior-point methods this week, and then the plan afterward is to do a "street-fighting" lecture on "convex-concave." Interior-point methods are how we solve the problem that has been gradually reduced to a simpler standard-form problem (usually a cone program); it turns out we can solve things in terms of equality-constrained smooth problems, at which point we can use Newton's method and linear algebra.

Example 234
First of all, consider inequality-constrained minimization of a smooth function $f_0(x) \le 0$ subject to $Ax = b$ along with some **smooth inequality constraints** $f_i \le 0$. We'll assume that our problem is strictly feasible and that the optimal p^* is finite and attained. Examples include linear and quadratic programs, but also QCQPs and GPs, entropy maximization with linear inequality constraints (say inequalities on expected values and tail probabilities of random variables), and so on.

(SDPs and SOCPs turn out to be better handled with generalized inequalities, so we'll handle those later.) The idea is that we can reformulate this inequality constraint using an indicator function: letting $I_-(u) = 0$ for $u \le 0$ and $I_-(u) = \infty$ otherwise, we can rewrite this as

$$\begin{aligned} \text{minimize} \quad & f_0(x) + \sum_{i=1}^m I_-(f_i(x)) \\ \text{subject to} \quad & Ax = b. \end{aligned}$$

This problem now "has no inequality constraints" because they're implicit, but we can't quite use Newton's method as is because I_- isn't smooth – the method actually won't work. But what we can do is approximate this via a **logarithmic barrier** and solve

$$\begin{aligned} \text{minimize} \quad & f_0(x) - \frac{1}{t}\sum_{i=1}^m \log(-f_i(x)) \\ \text{subject to} \quad & Ax = b. \end{aligned}$$

(The picture is that the function $-\frac{1}{t}\log(-x)$ is close to being zero if t is large, but then it goes to $+\infty$ as x approaches the boundary of feasibility.) So this gives us a smooth approximation which is Newton-ready, though this does make it so that it's "better for optimization to have some margin in the constraint." And as $t \to \infty$, our approximation gets better.

Remark 235. *We've heard something of the same structure earlier in this class before: we "approximated this same indicator" with a line λx in the Lagrange story. But this approximation is a lot more plausible because we do actually approach the function pointwise. The point though is that we should not be surprised if duality shows up, since the story started out the same way.*

Of course we could just take $t = 10^9$ now and think we've fully solved the problem for any practical purpose. But what really scares us in Newton's method is the third derivative being large (since that means rapid changes in the Hessian), and our logarithmic barrier will in fact cause problems for Newton's method actually working (numerically and also in terms of theoretical guarantees).

The point though is that the **log barrier function** for constraints $f_1, \cdots, f_m \le 0$, given by

$$\phi(x) = -\sum_{i=1}^m \log(-f_i(x)),$$

is convex from the composition rules, and we can compute its derivatives in terms of the gradients ∇f_i and the margins $-f_i$. So we can think of minimizing $tf_0(x) + \phi(x)$ instead.

Definition 236

For each $t > 0$, define the problem

$$\begin{aligned} \text{minimize} \quad & tf_0(x) + \phi(x) \\ \text{subject to} \quad & Ax = b \end{aligned}$$

and denote its solution $x^*(t)$. The set of points $\{x^*(t) : t > 0\}$ is called the **central path**.

The picture below shows an example of this for an LP; the dashed lines are the level curves for the barrier function. For $t = 0$ we get the analytic center of the inequalities, but then as t increases we care more and more about the objective over the barrier and thus approach the actual optimal value of the linear program. (And this gives us a sense of why these are called "interior-point methods:" following the curve will always give us something strictly feasible, hence in the interior.

We said earlier that we shouldn't be surprised that duality comes in, and here it is. It turns out that $x = x^*(t)$ (that is, we minimize $tf_0 + \phi$) if the KKT conditions are satisfied via

$$t\nabla f_0(x) + \sum_{i=1}^{m} \frac{1}{-f_i(x)}\nabla f_i(x) + A^T w = 0, \quad Ax = b$$

for dual variables w. In particular, this actually minimizes the Lagrangian of the original (non-barriered) problem

$$L(x, \lambda^*(t), \nu^*(t)) = f_0(x) + \sum_{i=1}^{m} \lambda_i^*(t) f_i(x) + \nu^*(t)^T(Ax - b),$$

except **at the specific value** $\lambda_i^*(t) = \frac{1}{-tf_i(x^*(t))}$ and $\nu^*(t) = \frac{w}{t}$. So if we minimize $tf_0(x) + \phi(x)$, we automatically get some dual feasible points, and so it makes sense to try evaluating that for a lower bound. When we do this, each $\lambda_i^*(t) f_i(x)$ just evaluates to $-\frac{1}{t}$, so

$$\begin{aligned} p^* &\ge g(\lambda^*(t) - \nu^*(t)) \\ &= L(x^*(t), \lambda^*(t), \nu^*(t)) \\ &= f_0(x^*) - \frac{m}{t}, \end{aligned}$$

where m is the number of inequality constraints. So solving the Newton-ready problem gives us some x^*, and out of that calculation we also get dual variables λ^*, ν^*. Evaluating those gives us $g(\lambda^*, \nu^*)$, and it turns out the gap is guaranteed to be at most $\frac{m}{t}$. So indeed as $t \to \infty$ we get a good quantitative bound on how far from optimality we are!

Fact 237

This phenomenon is usually explained in the following alternate way: on the central path, x^*, λ^*, ν^* satisfy modified KKT conditions. Specifically, we have primal and dual constraints, that the gradient of the Lagrangian with respect to x vanishes, and (this is the only difference) we have **approximate** complementary slackness $-\lambda_i f_i(x) = \frac{1}{t}$ (instead of zero).

Remark 238. *Consider such a problem where there are no equality constraints, meaning that we want to minimize* $tf_0(x) - \sum_{i=1}^{m} \log(-f_i(x))$. *This has the following physical interpretation:* $tf_0(x)$ *is the potential of a force field* $-t\nabla f_0(x)$, *and* $-\log(-f_i(x))$ *is the potential of a different force* $F_i(x) = \frac{1}{f_i(x)}\nabla f_i(x)$. *(So if we have a linear inequality constraint as part of a force field, we essentially have an force pushing us perpendicularly away from it, inversely proportional to distance.) We then want to find the point* $x^*(t)$ *at which all of these forces balance; the larger* t *is, the stronger the effect of* ∇f_0 *and thus the closer to the walls we can get.*

Interestingly in this perspective, we can have extra redundant constraints and that will actually change the central path. But we'll still end up at the same x^* *anyway.*

Remark 239. *An alternate way to solve an LP is to use the simplex method, where we start from a vertex and look among its adjacent vertices, choosing the one which minimizes the cost. But if we have* 10000 *constraints in* $\mathbb{R}^{1000}$, *we could theoretically have exponentially many vertices to consider (even if it works totally fine in practice). So it might feel more promising to "do things from the inside" instead, and in particular interior-point methods don't just* *work for LPs.*

Notice that all of this discussion didn't actually solve our problem though. Even in the 1960s, just solving the barrier with some big value like $t = 10^9$ went under the name of **UMT (unconstrained minimization technique)**. but we still haven't solved the looming issue with the huge third derivative.

Example 240

We'll now discuss the **barrier method**, often going under the name of **SUMT (sequential unconstrained minimization technique)**. UMT works perfectly well in some examples by tuning a particular good value of t, but the barrier method gives us something else to work with.

The picture to have in mind is this: pick some constant $\mu > 1$. We have some central path $x^*(t)$ approaching x^*, and we compute some particular point on this curve. Now we increase t to μt and try to compute the point $x^*(\mu t)$ on the central path, **using $x^*(t)$ as the starting point**. We then keep doing this until $\frac{m}{t} < \varepsilon$ for some $\varepsilon > 0$ that we choose.

Fact 241

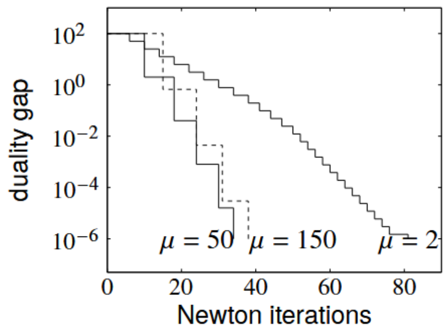
This is part of a general concept of **homotopy methods** (also called **path-following algorithms**): if we want to solve a set of equations $F(x) = 0$ but it's hard to actually do it, we can instead deform the problem. Specifically, introduce a parameter θ so that $F(x, 1) = F(x)$ and such that $F(x, 0) = 0$ is easy to solve. Then we can crank θ up from 0 to 1 gradually, and we attempt to solve $F(x, \theta + 0.01)$ from $F(x, \theta)$ using Newton's method (and if this fails, take the increment even smaller). Physically, we can imagine turning a dial up gradually and seeing how the conditions evolve over time (for example bias conditions given some supply voltage).

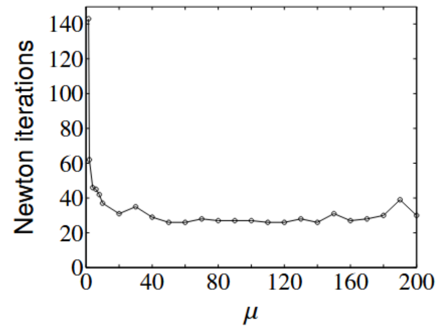
In general, homotopy methods can be weird and difficult to solve, because paths can bifurcate, start or stop, and so on. But for our convex problems, there's a single simple path, and we know it goes to the solution for sure, so everything will work out nicely! Furthermore, since we have the guarantee that $f_0(x^*(t)) - p^* \le \frac{m}{t}$, we know that we're within ε of the optimal value when we stop.

Qualitatively, we keep moving down the central path, so if μ is very close to 1 ("short-step") then Newton's method is starting close to the actual solution and we won't have to do much of a correction, since we'll probably even be in the quadratic convergence regime. Unfortunately, that means our "ignorance of optimal value" $\frac{m}{t}$ isn't going down very fast, so it will take a large number of outer iterations. On the other hand, if μ is very big ("long-step"), Newton's method will require many more steps. Usually μ is chosen to be something like 10 or 20. (And there are heuristics for how we choose the starting value $t^{(0)}$ as well.)

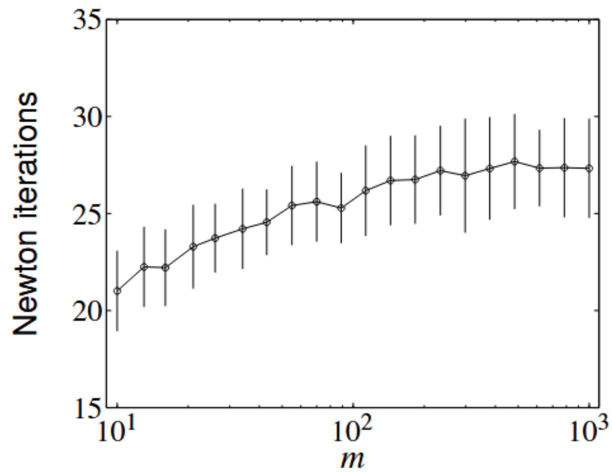
This method is completely rigorous and provides us with a near-optimal value x, along with certificates λ, ν of a particular gap through duality.

We can measure how well this method does in terms of how many total Newton steps it requires to get to some duality gap. Below is an example for an inequality form LP of 50 variables in 100 inequalities. The width of each stair here is how many iterations it takes to get to the next point in the central path, and the height is how far down we've gone in gap each time. So with small μ we have small steps but in a short amount of Newton iterations, and with large μ we have big steps but with many more Newton iterations.





Surprisingly, if we look at the right plot, we see that it doesn't actually matter very much what value of μ we take: the big reduction in duality balances out with the number of steps we have to take as long as $\mu \geq 10$. (Around $\mu = 300$ it does tend to go up again, though.) Furthermore, we get to a small duality gap in roughly 30 iterations, even though we aren't anywhere near the boundaries when we start. And weirdly all of this also works even if we're in a huge number of dimensions. If we solve an LP with constraints $Ax = b$ for $A \in \mathbb{R}^{m \times 2m}$, the number of Newton iterations grows very slowly in the dimension size:



And even with something like 10^9 dimensional problems, taking a day on a supercomputer and a terabyte of data, it still just takes 25 Newton iterations in practice (which is a wild thing which probably can't really be intuitively justified). But of course each individual one of those iterations takes much longer because the linear algebra requires lots of computation. (There are theoretical guarantees, but they are very far from the real truth anyway.)

Example 242

If we start the barrier method with an infeasible-start Newton, we don't need to worry about this next point at all. But it's worth keeping this in mind for historical context anyway. Barrier methods typically require a strictly feasible starting point, meaning some x with $f_i(x) < 0$ for all x. **Phase I methods** form optimization problems which are strictly feasible and actually find that strictly feasible point for us, if they exist. (**Phase II** then consists of actually running the barrier method starting from that point.)

The idea is to introduce a slack variable and solve the following epigraph problem over x, s:

$$\begin{aligned} \text{minimize} \quad & s \\ \text{subject to} \quad & f_i(x) \leq s \text{ for } 1 \leq i \leq m, \\ & Ax = b. \end{aligned}$$

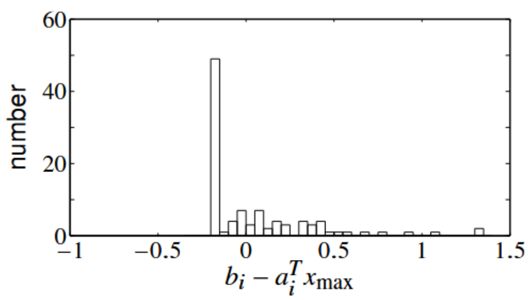
This is basically a smooth formulation of "minimize $\max(f_1, \cdots, f_m)$" which allows us to actually use Newton's method to solve it. We then get some optimal value $\overline{p}^\star$; if it's negative then we found a strictly feasible point, and if it's positive then there is none. But if $\overline{p}^\star = 0$ then this is actually inconclusive.

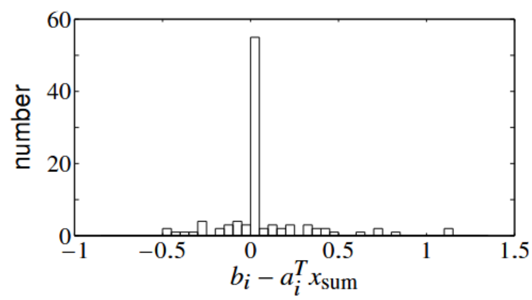
Alternatively, we can introduce a slack for each variable and minimize the sum of slacks:

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^T s \\ \text{subject to} \quad & s \succeq 0, \\ & f_i(x) \le s_i \text{ for } 1 \le i \le m, \\ & Ax = b, \end{aligned}$$

and we use a barrier method to solve this so that we actually get a strictly feasible point. This is essentially minimizing a sum of ReLu functions, and it always finds a strictly feasible point if one exists. But furthermore, the advantage is that even if the problem is infeasible, it will produce a solution that satisfies many (but not all) of those inequalities. And furthermore, we can weight the slacks in whatever way we'd like to set priorities in which constraints are preferably satisfied.

The figure below shows an example of this with an infeasible set of 100 linear inequalities in 50 variables. The left plot shows the basic phase I solution; it satisfies 39 of the inequalities and a lot of the margins are exactly equally as bad. And the right plot shows what happens if we do the sum of infeasibilities instead; it satisfies 79 inequalities and many of the margins are exactly zero.





23 March 3, 2026 (Problem Session)

We'll do some problems with a lot of setup, focusing on how to turn all of the words into convex optimization problems.

Problem 243

Suppose we have a vehicle traveling through a fixed path of n segments, meaning that we have some points $w_0, w_1, \cdots, w_n$ and the lengths $d_i = w_i - w_{i-1}$ are given. We are told that we travel at a constant speed per segment $s_i \ge 0$, and we must have $s_i^{\min} \le s_i \le s_i^{\max}$ for all i. We also are told that we start the journey at time 0, and the arrival times τ_i at each waypoint must be between $\tau_i^{\min}$ and $\tau_i^{\max}$.

The goal of the problem is to choose our speeds to minimize the fuel cost, where the rate of consumption of the vehicle (per unit time) is a positive increasing convex function $\Phi(s)$ of the speed.

Since the speed is constant per segment, it will take a time $t_i = \frac{d_i}{s_i}$ to clear segment i. And even though speed is stated to be nonnegative, it actually must be positive so that we reach all of the waypoints (otherwise the arrival time constraints will not be satisfied). And we can relate these times to the arrival times via $\tau_{i-1} + t_i = \tau_i$, or equivalently

$$\tau_i = t_1 + \cdots + t_i = \sum_{j=1}^{i} \frac{d_j}{s_j}.$$

So in terms of the speeds, our constraints look like

$$\tau_i^{\min} \le \sum_{j=1}^{i} \frac{d_j}{s_j} \le \tau_i^{\max}.$$

But as stated, this won't pass the DCP tests that we might want: our function here is indeed convex in the variables s_i (because all distances are positive), so the upper bound is a valid inequality to have in a convex optimization problem but not the lower bound. We'll address this after we write down the total fuel consumption, though: over segment i, our consumption is $t_i \Phi(s_i) = \frac{d_i}{s_i}\Phi(s_i)$, so the objective we want to minimize is

$$f_0(s) = \sum_{i=1}^{n} \frac{d_i}{s_i}\Phi(s_i).$$

And again we see the same problem: even though $\Phi(s_i)$ is convex and so is $\frac{d_i}{s_i}$, a product of convex functions isn't always convex (for example if $\Phi(s) = s^{3/2}$, then $\frac{1}{s}\Phi(s)$ would be $s^{1/2}$, which is instead concave – being able to engineer these examples is good practice for us). So what we want to do is change variables in a way that still lets us recover the values of s_i and formulates things as a convex optimization problem.

To figure out the right way to do this, notice that a two-sided constraint like $\tau_i^{\min} \le \tau_i \le \tau_i^{\max}$ can only be permitted in a convex optimization problem if τ_i is actually **affine** in our variables – think of one side as a sublevel set and the other as a superlevel set. Indeed, some simple counterexamples where superlevel sets of convex functions are not convex which are good to keep in mind are x^2 (where the 1-superlevel set is $(-\infty, -1] \cup [1, \infty)$) and, more closely related to our problem here, $\frac{1}{x} + \frac{1}{y}$ on $\mathbb{R}^2_{++}$ (which we can see visually by plotting the level curves $\frac{1}{x} + \frac{1}{y} = 1, 2$).

So for these reasons, because τ_i is the sum of the times t_i, it makes sense to **use segment travel times as our new variables**. Then $t_i = \frac{d_i}{s_i} \implies s_i = \frac{d_i}{t_i}$, so our objective can be written in terms of the t_is as

$$f_0(t) = \sum_{i=1}^{n} t_i \Phi\left(\frac{d_i}{t_i}\right),$$

and now this is indeed convex because this is the sum of perspectives of Φ (evaluated at d_i in the original variables). So that's good, and for the constraints $s_i^{\min} \le s_i \le s_i^{\max}$ become (by taking reciprocals) $\frac{d_i}{s_i^{\max}} \le t_i \le \frac{d_i}{s_i^{\min}}$ and the arrival times are now constrained via

$$\tau_i^{\min} \le \sum_{j=1}^{i} t_j \le \tau_i^{\max}.$$

We could also write these constraints in matrix form if we wanted to: for instance, the speed constraints could be written $\operatorname{diag}(s^{\max})^{-1}d \preceq t \preceq \operatorname{diag}(s^{\min})^{-1}d$, and the arrival time constraint could be written $\tau^{\min} \preceq Lt \preceq \tau^{\max}$, where L is a lower triangular "cumulative sum" matrix with all permissible entries 1. Thus everything has been formulated in a way that allows us to use convex optimization – our constraints are actually all affine – and once we find our optimal t_i^*s we can recover the optimal speeds via $s_i^* = \frac{d_i}{t_i^*}$.

Turning to CVXPY implementation details now, we have our objective and speed / time bounds in terms of t_i and can feed it into an optimization problem. In the case where $\Phi(s_i) = as_i^2 + bs_i + c$ is quadratic (for example), we can write out the objective explicitly as

$$\sum_{i=1}^{n} \left(a\frac{d_i^2}{t_i} + bd_i + ct_i \right).$$

(If we had a more general convex function for Φ, we could use the built-in perspective transform as well, but it might be a bit annoying to work with especially in terms of broadcasting to higher dimensions. So it's generally better to simplify as much as possible.) And everything else follows by directly translating our math to code, with a couple of notable details to note:

- Remember that we can specify that our variables are positive via something like `t = cp.Variable(n, pos = True)`.
- Also, if we wanted to write out the quantity $\sum_{i=1}^{n} \frac{d_i^2}{t_i}$, we should use `cp.multiply(cp.square(d), cp.inv_pos(t))` to get the right shape and satisfy DCP rules.
- For the arrival times $\sum_{j=1}^{i} t_j$, at this scale of the problem it makes sense to just explicitly write out the element-wise constraints (like `cp.sum(t[:i+1]) >= tau_min[i]`) rather than putting it big matrix form.

Problem 244

We'll now start setting up another practical problem also involving resource management. Suppose we have four types of blood (O, A, B, AB) and there are rules on what blood types can contribute to demands of what other blood types. These rules can be described via a **directed graph**, where an edge pointing from v to w means that v can contribute to w. In our case, all four nodes (for the four blood types) have a self-edge, and there are also edges from O to A, B, and AB, as well as edges from A and B to AB.

We have two blood banks of this type, each with a supply and demand of each of the four types of blood. Our goal is then to meet the demand while minimizing some combination of the price of blood consumed and the transport cost.

We can formulate this mathematically as follows: we have some given demand vector $d = \begin{bmatrix} d_O \\ d_A \\ d_B \\ d_{\text{AB}} \end{bmatrix}$ for the first bank, as well as a similar one $\tilde{d}$ for the second bank, and we express the distribution of blood used via policy matrices $B, \tilde{B}$, where B_{ij} means how much blood type j contributes to the demand for type i. So in other words, our policy matrices are forced to take the "sparse" form

$$\begin{bmatrix} * & 0 & 0 & 0 \\ * & * & 0 & 0 \\ * & 0 & * & 0 \\ * & * & * & * \end{bmatrix}.$$

(So this is essentially adding equality constraints to the optimization problem – in CVXPY, we can do this by specifying a list of ordered pairs of (i, j) of allowed nonzero entries of the matrix.) The rows of this matrix, obtained by computing $B\mathbf{1}$, must then be **equal to** the corresponding values of the demand d. Similarly, the columns of this matrix, obtained by computing $B^T\mathbf{1}$, must be **at most as large as** the total supply of blood at the bank.

What we're allowed to do (and optimize over) is transport units of blood $t \in \mathbb{R}^4$ to add to the supply of one bank and subtract from the other. Calling the starting supply vectors $s, \tilde{s}$ at the two banks, we can set up the constraints at the two banks with $s + t$ and $\tilde{s} - t$ instead of $s, \tilde{s}$; our end goal will then just be to minimize the resulting objective in terms of $t \in \mathbb{R}^4$, $B \in \mathbb{R}^{4\times4}$, and $\tilde{B} \in \mathbb{R}^{4\times4}$. (Alternatively, in terms of the graph language above, we can now form a graph with eight nodes instead of four, and we also draw edges between the corresponding nodes between the same blood type. We then optimize over all possible values of the resulting 8×8 matrix, with some entries constrained to be nonnegative and others constrained to be zero, such that consumption does not exceed supply but meets demand.) Our objective is then the sum of consumption cost at bank 1, consumption cost at blank 2, and the shipment cost, the former two of which we can write by dotting the column sums with the cost vectors and the latter of which is given:

$$f = p^T B^T \mathbf{1} + \tilde{p}^T \tilde{B}^T \mathbf{1} + \kappa ||t||_1.$$

This is indeed convex in our problem variables (it's linear in $B, \tilde{B}$ and $||t||_1$ is convex), and the constraints $B\mathbf{1} = d, \tilde{B}\mathbf{1} = \tilde{d}, B^T\mathbf{1} \preceq s - t, \tilde{B}^T 1 \preceq \tilde{s} + t$, and the zero / nonnegative conditions on the entries of $B, \tilde{B}$ are all affine, so there are no issues with setting this up with our given variables.

The only additional detail with the CVXPY implementation to point out here is how we specify the sparsity rules for blood donation. If we have a Python list of pairs (i, j) of locations where B_{ij} must be zero (remembering to zero-index), we can add the sparsity constraints by just explicitly adding the constraints `B[i,j] == 0` by looping over that list.

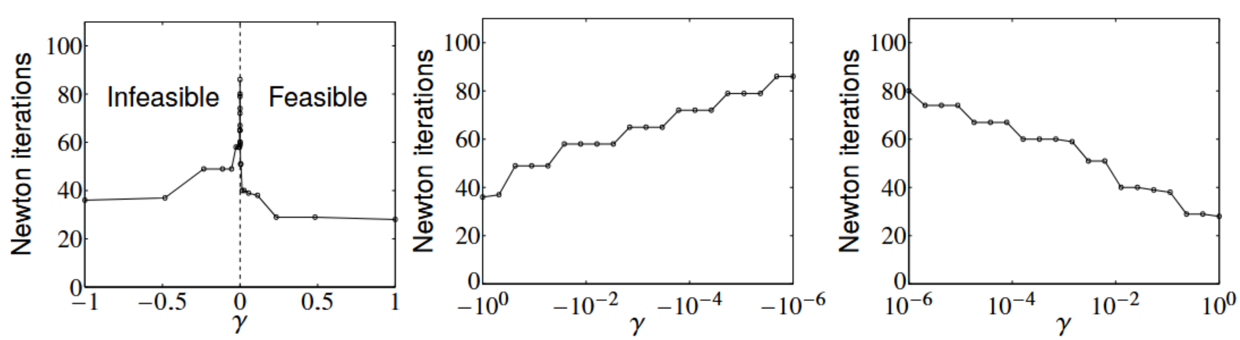
24 March 5, 2026

We'll finish up the "main part" of the course today. We discussed Phase I methods last time, giving us a method to get a feasible starting point to use for our barrier method. One idea was to use the epigraph representation (minimize $\max_i f_i$) and the other was to consider $\mathbf{1}^T s$ instead so that we can produce solutions which satisfy most constraints even if infeasible.

Example 245

Suppose we have a family of linear inequalities where $Ax = b$ is the boundary of feasibility, and we consider $Ax \preceq b + \gamma \Delta b$. So when $\gamma > 0$, we grow the feasible polyhedron, and when $\gamma < 0$, the polyhedron disappears but then the dual problem of certificates proving feasibility gives us a polyhedron in dual space. We can then ask how long phase I methods take as a function of γ (before either our margin s is negative or the dual objective is positive); it turns out to be harder for γ closer to zero and the number of iterations required is something like $\log\left(\frac{1}{|\gamma|}\right)$.

The plot is shown below – usually it takes something like 30 total Newton steps, but for ridiculously small γ things do start to grow.



Example 246

We'll turn now to thinking about complexity of the barrier method. Remember that every time we recenter in the outer iteration, the dual gap is reduced by the factor μ that we increment t by. Thus we know exactly how many outer iteration steps it will take to conclude: it'll just be $\lceil \frac{\log(m/(\varepsilon t^{(0)}))}{\log \mu} \rceil$. So we just need to bound the number of Newton steps per centering iteration, and we'll do so with the same **self-concordance analysis** that we saw earlier on. (Classical analysis gives a very bad upper bound, because as we increase t the third derivative just keeps increasing.)

We'll use the same assumptions as we did before (sublevel sets are bounded, and $tf_0 + \phi$ is always self-concordant with closed sublevel sets). This holds for a very large class of problems, though we may need to add some additional constraints or reformulate the problem. But it's important to remember that this is theory-only; our methods work perfectly well even when the functions aren't actually self-concordant.

In each centering step (going from $x^*(t)$ to $x^+ = x^*(\mu t)$), we are trying to minimize $\mu t f_0(x) + \phi(x)$ starting from $x^*(t)$. We know from self-concordance theory that it basically takes some linear function of (current value minus optimal value); specifically

$$\#\text{ iterations} \le \frac{\mu t f_0(x) + \phi(x) - \mu t f_0(x^+) - \phi(x^+)}{\gamma} + c$$

for γ, c some constants. So all we need to do is bound this numerator in a way that is independent of t. First of all, remember the dual variables here are $\lambda_i = \lambda_i^*(t) = -\frac{1}{tf_i(x))}$, so we can write

$$\phi(x) = \sum_{i=1}^m -\log(-f_i(x)) = \sum_{i=1}^m \log(t\lambda_i)$$

and thus plugging back in yields

$$\phi(x) - \phi(x^+) = \sum_{i=1}^m \log(t\lambda_i) + \log(-f_i(x^+)) = \sum_{i=1}^m \log(-\mu t \lambda_i f_i(x^+)) - m \log \mu.$$

And now using concavity, linearizing $\log u \le u - 1$ tells us that we can bound this by $-\mu t \sum_{i=1}^m \lambda_i f_i(x^+) - m - m \log mu$. Thus

$$\mu t f_0(x) + \phi(x) - \mu t f_0(x^+) - \phi(x^+) = \mu t f_0(x) - \mu t \left(f_0(x^+) + \sum_{i=1}^m \lambda_i f_i(x^+) + \nu^T(Ax^+ - b) \right) - m - m \log \mu;$$

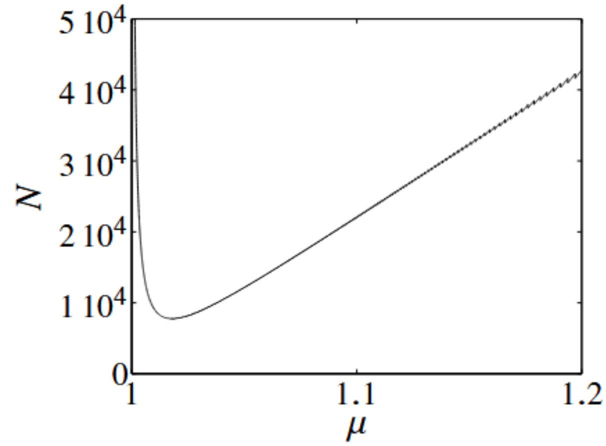
that is, we've written our numerator in terms of the Lagrangian at a particular point x^+, λ, ν, and in particular $L(x^+, \lambda, \nu) \ge g(\lambda, \nu)$. But that means our bound looks like

$$\mu t f_0(x) - \mu t g(\lambda \nu) - m - m \log \mu = m(\mu - 1 - \log \mu),$$

because we know that the gap is exactly $f_0(x) - g(\lambda, \nu) = \frac{m}{t}$. So we get a t-independent bound, and we know exactly what this right-hand side is because m is the number of inequalities and we chose μ! Therefore, the number of Newton iterations is at most

$$\left(\lceil \frac{\log(m/(\varepsilon t^{(0)}))}{\log \mu} \rceil \right) \left(\frac{m(\mu - 1 - \log \mu)}{\gamma} + c \right)$$

If μ is really small, this is quite big because we take many outer iterations, and if μ is quite big then we might need a lot of Newton steps per outer iteration. But we get a tradeoff and see that we actually get a somewhat sensible bound (even if it is still very far from the truth) somewhere in the middle: the graph for the bound on number of iterations N versus μ, with $m = 100$ and an initial duality gap of 10^5.



In practice, the number of iterations is really in the tens and not really sensitive for large μ. But for theoretical purposes, optimizing N above by plugging in $\mu = 1 + \frac{1}{\sqrt{m}}$ shows that **the number of Newton iterations is something like** $O(\sqrt{m}\log(m/(\varepsilon t^{(0)})))$. So even multiplying with the cost of a single Newton iteration, which is polynomial in the problem dimensions, gives us a polynomial upper bound on the number of flops required regardless of what specific convex optimization problem (LP, QP, SOCP, etc.)! It's pretty interesting to think about how at the end of the day, we end up just solving any problem by doing something like twenty total least-squares problems.

Example 247

We'll now generalize to **generalized inequalities**, where instead of $f_i \le 0$ we have cone inequalities $f_i(x) \preceq_{K_i} 0$ for K_i proper cones. (This is particularly useful for SOCPs or SDPs.) The key is to come up with a good concept of a logarithm that works for vectors instead of just numbers.

This covers the case where our f is for example symmetric matrix-valued and we're using the positive semidefinite order, so $f \preceq_K 0$ means f is negative semidefinite matrix.

Definition 248

A function $\psi : \mathbb{R}^q \to \mathbb{R}$ is a **generalized logarithm** for a proper cone K if

- $\text{dom}\,\psi = \text{int}\,K$ and $\nabla^2\psi(y) \prec 0$ for any $y \succ_K 0$,
- it looks like a logarithm on lines through the origin, meaning that for some θ (called the **degree** of ψ), we have $\psi(sy) = \psi(y) + \theta \log s$ for any $y \succeq_K 0$ and $s > 0$.

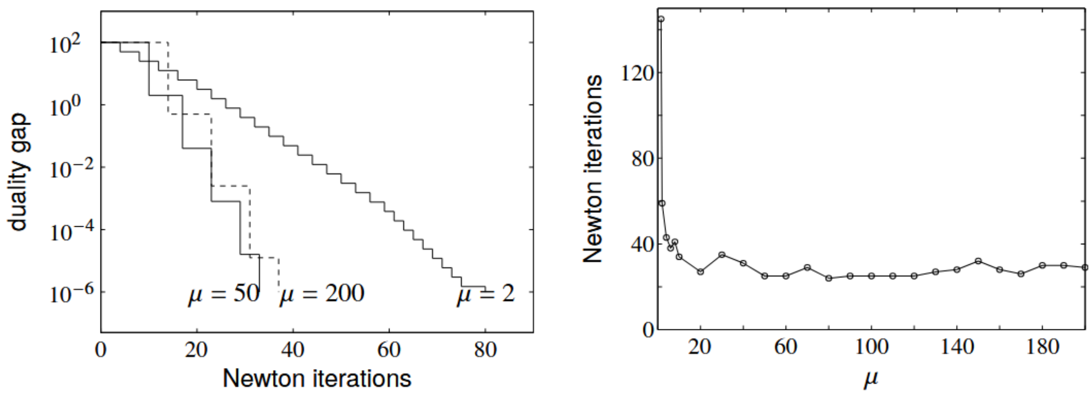
For example for the positive semidefinite cone, $\psi(Y) = \log\det Y$ with degree n. There's various useful properties here – for example, for any y in the cone K, we have $\nabla\psi(y) \succeq_{K^*} 0$ for K^* the dual cone, and in fact $y^T\nabla\psi(y) = \theta$.

Once we have the concept of a generalized logarithm, we can define a central path: the **logarithmic barrier** for these cone inequalities $f_i(x) \preceq_{K_i} 0$ is

$$\phi(x) = -\sum_{i=1}^{m} \psi_i(-f_i(x))$$

for ψ_i the generalized logarithms for K_i. The central path again consists of the points where we minimize $tf_0(x) + \phi(x)$, and actually everything here works out the same way as before except instead of having a gap of $\frac{m}{t}$, our duality gap is now $\frac{1}{t}\sum_{i=1}^{m}\theta_i$. So we can deal with things like matrix inequalities in semidefinite programming and so on, and the barrier method just works exactly the same as before except using the new duality gap.

The figure below shows an example of a second-order cone program for 50 variables and 50 SOC constraints in $\mathbb{R}^6$; notice that the plots look very similar to what we had before. Again notice that the algorithm takes something like 30 Newton iterations over a wide range of μ.



And once again, if we try specifying problems over a pretty wide range of dimensions, the number of Newton iterations only grows very slowly in practice.

Remark 249. *If we feed CVXPY a bunch of different types of constraints, it always compiles things into a good form for solving, and generally the answer is a cone program of the simplest possible case. But notice that we have never actually had to do this transformation ourself, and that's the really nice thing about the state of the field these days!*

Fact 250

The **actual** methods used in practice (for real) are not exactly this barrier method; we do need a couple of tricks to get something fully competitive. First of all, we don't actually distinguish between outer and inner iterations: t (or usually the parametrization is $\kappa = \frac{1}{t}$) is chosen at every step, and we always update the dual and primal variables. But the same KKT conditions and log barriers are indeed being used; it's just that there's a way to adjust κ at each iteration in a way that often exhibits superlinear asymptotic convergence.

Essentially, we factorize the KKT system matrix and get the "affine direction" and "centering direction," and we take some combination using the "Mehrotra predictor-corrector method." Then the step size is literally exactly 0.99 times the distance to the boundary in that direction. No one really knows why this works at all (at either the convergence or complexity level), but it's been powering so many systems for decades!

Remark 251. *Notice that CVXPY doesn't have to actually take any derivatives or compute any Hessians, because we actually just compile the problem into some explicit form. So this is rather different from the kinds of "oracle-based" optimization we see in machine learning or AI.*

So to conclude this "main part" of the course, let's walk through some of the takeaways. Lots of problems in various fields are mathematical optimization problems, possibly with multiple objectives and considerations involving data uncertainty. The ones that are generally tractable are the convex ones, and luckily lots of problems can be formulated as convex ones and so we have a lot of interesting theory and useful algorithms to work with (in particular, local optima are global, duality gives us lower bounds and sensitivity analysis, and we have solution methods that have worst-case polynomial complexity and take really 20 to 80 steps in practice).

So if we want to use convex optimization in some applied context, it's probably useful to do **rapid prototyping and approximate modeling**: fit a piecewise linear thing and then throw it into CVXPY and see if we like the results. If they're good, then we can keep working by simplifying our conditions or looking at the dual problems. And even in nonconvex problems, we can still use convex optimization in some sequential way (which we'll see next time).

We haven't covered things like subgradient methods for very large problems, more detailed convex analysis, proximal methods, distributed convex optimization, and so on. But those are covered in the next quarter's course if we're interested!

25 March 10, 2026

Today's topic is the **convex-concave procedure**, which is a cool extension of the material we've been discussing. In this class, we pretend everything is convex, and we have problems where things are "pre-digested" in a way that lets us render them convex (possibly after some transformations). On a few occasions, we've thought about things like integer problems where we can use relaxation as a heuristic. Once we've mastered that material, we should note that in the real world things aren't actually convex (for example, because when we trade on the stock market we can only trade an integer number of shares), so what really matters is being able to turn "almost-convex" things into things that work. Today's material is a non-obvious way to do that.

Basically, the convex-concave procedure (CCP) is a street-fighting trick which lets us solve a typically small sequence of convex problems to heuristically solve a specific kind of nonconvex problem.

Definition 252

A **difference of convex** function $h : \mathbb{R}^n \to \mathbb{R}$ is a function of the form $h(x) = f(x) - g(x)$ for f, g convex.

This is basically all functions, in the sense that any function with continuous second derivative has this form (take the positive part of the second derivative and integrate it twice to get f; then take the negative part and integrate it twice to get g). But the point is that we can get some fairly general nonconvex functions:

Example 253

Here are two illustrative cases:

- x^3 is a difference of convex functions via $(x^3)_+ - (x^3)_-$.
- For any nonconvex quadratic function $\frac{1}{2}x^T Px + q^T x + r$ for P symmetric, decompose it into its PSD and NSD parts via breaking up the eigenvalue decomposition $P = Q\Lambda Q^T$ into the elementwise Λ_+ and Λ_- diagonal matrices. This then lets us write $h = f - g$ with $f = \frac{1}{2}x^T P_{\text{psd}}x + q^T x + r$ and $g(x) = \frac{1}{2}x^T P_{\text{nsd}}x$.

The point now is that for any convex g, we know that the linear Taylor approximation satisfies

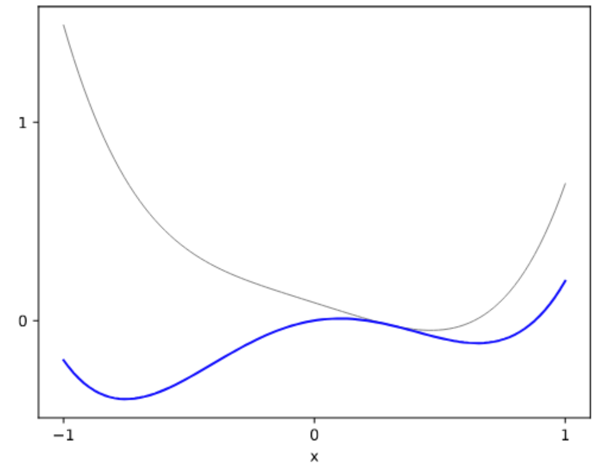
$$\hat{g}(x; z) = g(z) + \nabla g(z)^T (x - z) \leq g(x)$$

for all x, z. (If g is nondifferentiable, we can replace ∇g with any subgradient.) So we can define the **approximator at** z

$$\hat{h}(x; z) = f(x) - \hat{g}(x; z),$$

where we keep the convex part and linearize the concave part. So now this is a convex function since we subtract off something affine, and in particular since $\hat{g}(x; z) \leq g(x)$ we have $\hat{h}(x; z) \geq h(x)$; that is $\hat{h}$ is a **majorizer** of h (above the original nonconvex function), which is tight at z.

For an example, below we have the graph of $x^4 + 0.2x - x^2$ in blue and its convex majorizer in gray at $x = 0.3$, where $f(x) = x^4 + 0.2x$ and $g(x) = -x^2$:



So the idea now is that if we want to minimize $h = f - g$ (which may have local minima), the **convex-concave procedure** just repeatedly minimizes $\hat{h}$ via

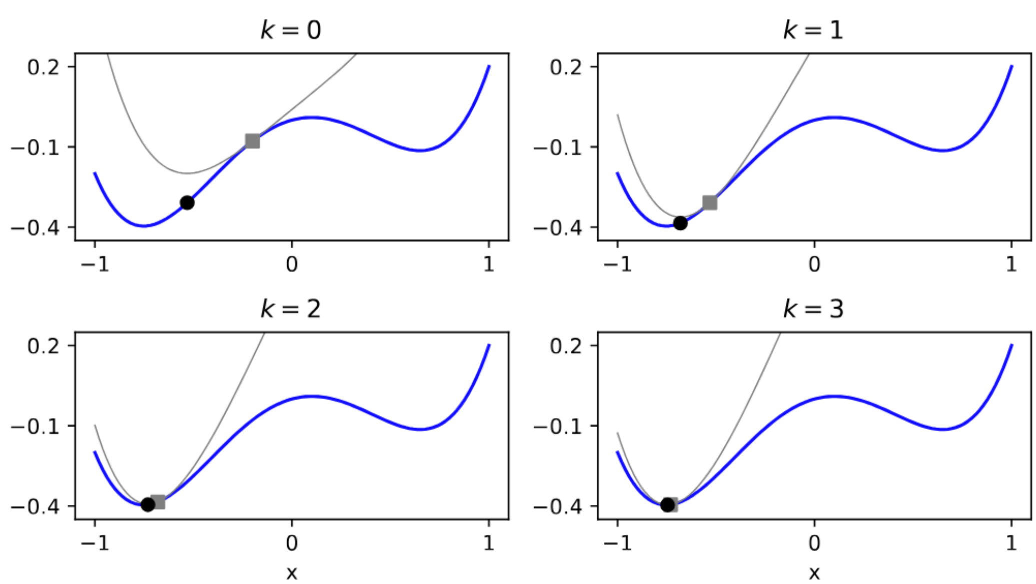
$$x^{k+1} = \text{argmin}_x \hat{x}(x; x^k) = \text{argmin}_x (f(x) - \hat{g}(x; x^k)).$$

Notice that there are no algorithm parameters at all; we just optimize with convex optimization, and then form another function to minimize, and repeat. This is called an "MM method' (majorize, minimize). Notice that this is a descent method, since we're minimizing the majorizer and the true value must have gone down by at least that much:

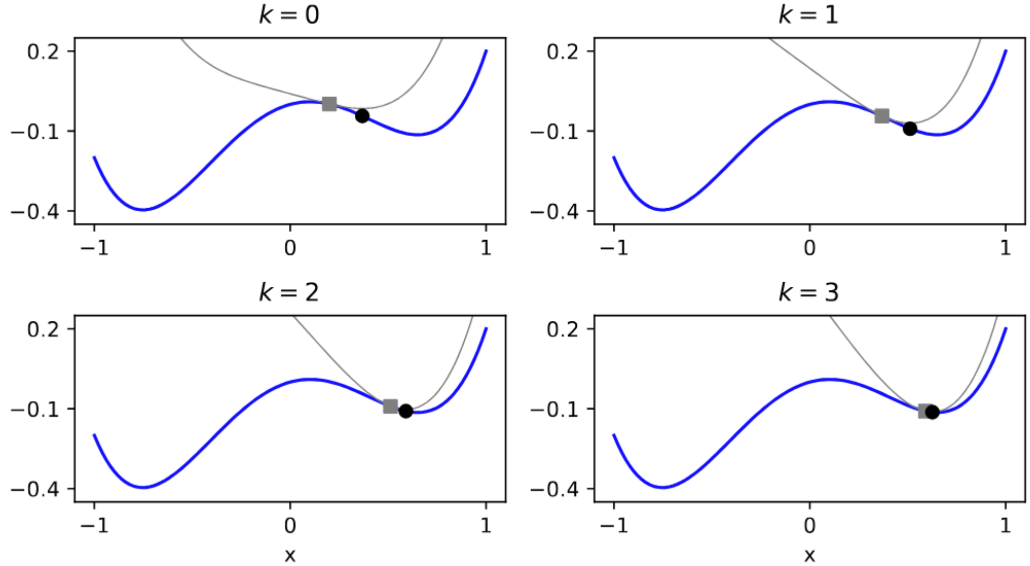
$$h(x^{k+1}) \leq \hat{h}(x^{k+1}; x^k) \leq \hat{h}(x^k; x^k) = h(x^k).$$

This is a local optimization method which converges, but it doesn't necessarily converge to the true minimum $h^* = \inf_x h(x)$; it depends on our initial condition. So the trick is usually to **run CCP for multiple initial points and take the best ultimate point as a suspected global minimum.**

For the function example above, here are two different sample runs of the convex-concave procedure from different starting points -0.2 versus 0.2: in both cases, we converge to a local minimum, but only one of them is the true global minimum. In the first case, we converge to the left local minimum:



And in the second case, we converge to the right local minimum. (Observe that this process looks kind of Newton-like with its approximations.)



Example 254

For a more general example, consider the **SAT problem** where we want to find boolean $x_i \in \{0, 1\}$ satisfying $Ax \preceq b$. Encoding $x_i \in \{0, 1\}$ by $x_i^2 - x_i = 0$, the SAT problem is actually equivalent to minimizing the difference-of-convex function $h = f - g$ with

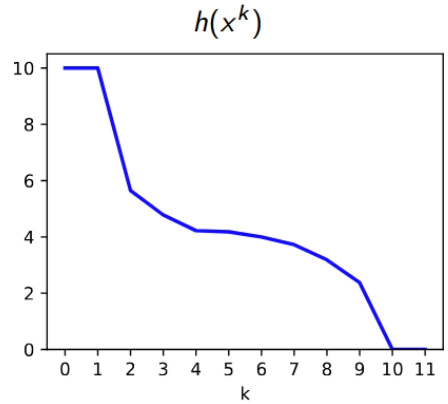
$$f(x) = \begin{cases} 0 & Ax \preceq b \text{ and } 0 \preceq x \preceq \mathbf{1}, \\ \infty & \text{otherwise,} \end{cases} \qquad g(x) = \sum_{i=1}^{n}(x_i^2 - x_i).$$

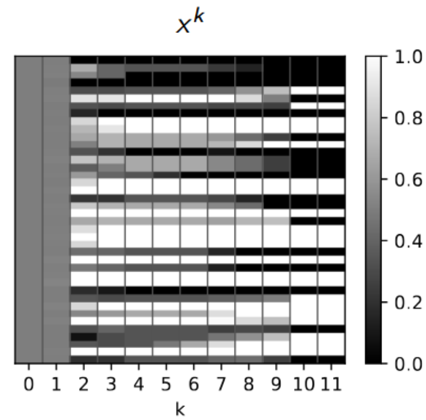
Indeed, for all x we have $h(x) \geq 0$, and we have equality if and only if all x_i actually lie in $\{0, 1\}$ and our point is feasible. So we can run this procedure and try to see if we converge to zero (if it converges to a positive value, we failed and should try again).

It turns out that in the convex-concave procedure, we find x^{k+1} by solving

$$\begin{aligned} \text{minimize} \quad & (\mathbf{1} - 2x^k)^T x \\ \text{subject to} \quad & Ax \preceq b, \\ & 0 \preceq x \preceq \mathbf{1}. \end{aligned}$$

So we can take any predicate and encode it a matrix A, and then perhaps run the CCP starting from the all-0.5s state. In different iterations, we'll see that some fraction of the time, we'll find a feasible point (meaning we've found a feasible x where all points are either 0 or 1) – here is an example with $n = 40$ and with 120 clauses:





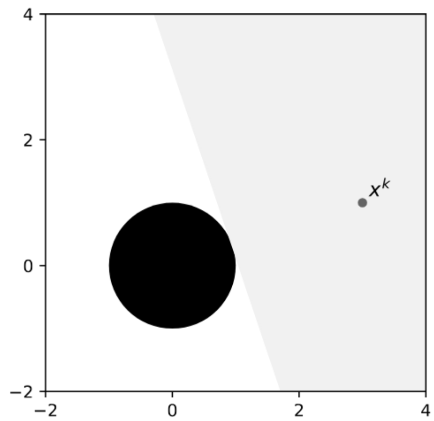
Fact 255

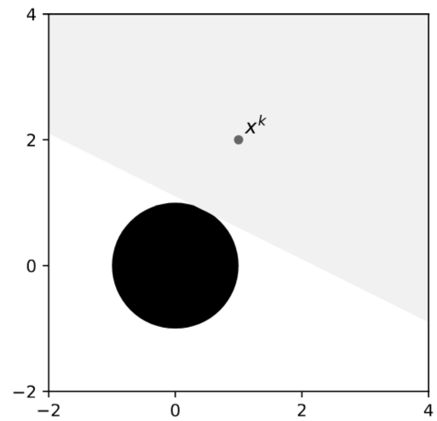
We can also think about constrained difference-of-convex problems like

$$\begin{aligned} \text{minimize} \quad & f_0(x) - g_0(x) \\ \text{subject to} \quad & f_i(x) - g_i(x) \leq 0 \text{ for } 1 \leq i \leq m, \end{aligned}$$

where all f_i, g_i are convex. It turns out we get a feasible descent method by linearizing all convex parts g_0, g_i; the convexified constraint ends up being a convex restriction of the original constraint.

One important example case is **obstacle avoidance** in trajectory optimization, for example if we want to avoid a sphere in space. The idea is that instead of avoiding the black circle below, we want to be inside the gray region, which is a halfspace tangent to that circle (which depends on our point x^k).





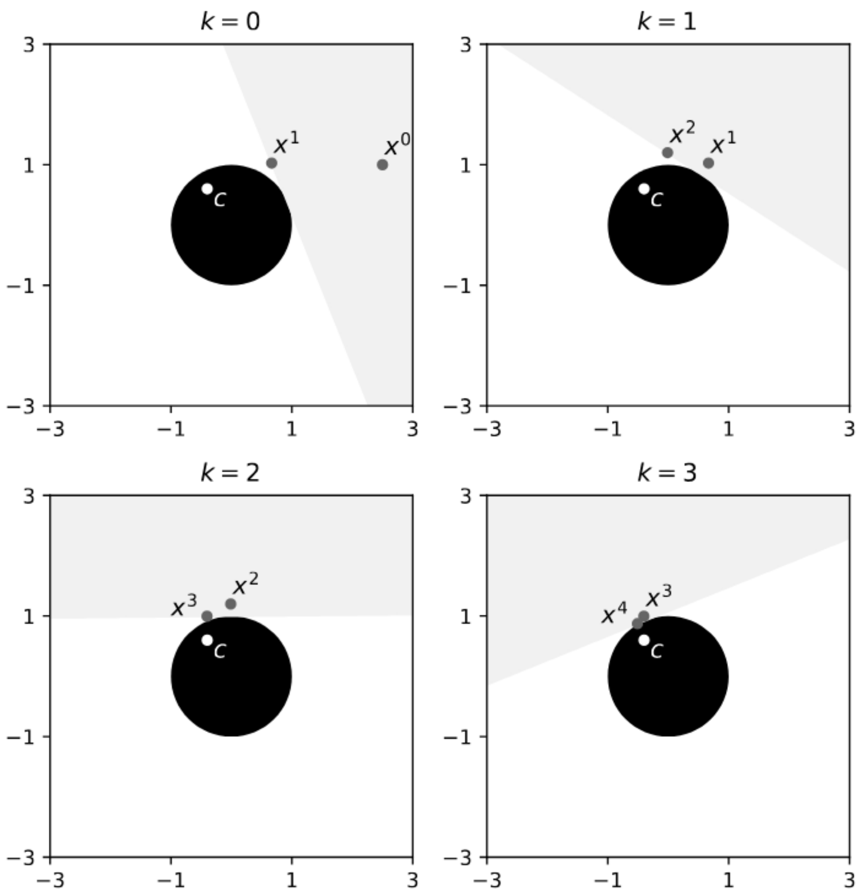
So we still have a descent method because x^{k+1} will be feasible if x^k is, and by the same reasoning as before we must have $h(x^{k+1}) \leq h(x^k)$.

Example 256

Suppose we want to solve the simple problem

$$\begin{aligned} \text{minimize} \quad & ||x - c||_2 \\ \text{subject to} \quad & ||x||_2 \geq 1. \end{aligned}$$

The actual solution is clear – if c is outside the disk we take c, and otherwise we take the closest point on the perimeter to it (which is c scaled up). If c is inside the unit disk, then what the convex-concave procedure does is "wind around the circle" while staying within the permissible halfspaces, as shown below.



Fact 257

Next, thinking about how we satisfy equality constraints, we can just add linear equality constraints as indicator functions to f. But we can even deal with general difference-of-convex equality constraints $p_i(x) = q_i(x)$; this is done by expressing the constraint as a combination of inequalities $p_i - q_i \leq 0$ and $q_i - p_i \leq 0$ and then doing the linearization separately (on q and p, respectively).

Note however that convex subproblems can be infeasible even if the original problem is feasible, though, so we can introduce slack variables s_i and use the **penalty CCP algorithm** instead: increase some constant τ_k between iterations and try successively solving

$$\begin{aligned} \text{minimize} \quad & f_0(x) - \hat{g}_0(x; x^k) + \tau_k \sum_{i=1}^m s_i \\ \text{subject to} \quad & f_i(x) - \hat{g}_i(x; x^k) \le s_i \text{ for } 1 \le i \le m, \\ & s_i \ge 0 \text{ for } 1 \le i \le m. \end{aligned}$$

This problem is always feasible, and gradually we will get xs which are possibly "closer and closer to being feasible." But we may never find a feasible point for the original problem even if there is one.

This leads us to the idea of **disciplined convex-concave programming (DCCP)**, which is like DCP but where we're allowed to now minimize concave or maximize convex functions, and also our constraints can now take the form $\ell_i(x) \sim r_i(x)$ where ℓ_i, r_i can be either DCP convex or DCP concave, and $\sim$ can be any of $\le, \ge, =$. So the point is that as long as we know the curvature of our expressions, we don't force the problem to be convex in the usual sense. Then if we want to minimize the difference-of-convex objective $f_0 - g_0$, we introduce an epigraph variable t and add in the constraint $f_0(x) - g_0(x) \le t$, and we minimize t. So we can say things like `cp.norm2(x) >= 1`, which will not be DCP, but they will be DCCP.

Remark 258. *Both gradient descent and Newton can be explained as majorize-minimize methods, and this is a useful way to think about optimization. Conceptually, when we figure out the step size, we're adjusting a quadratic where we modify the first- or second-approximation to make it a majorizer, and then we minimize that.*

Fact 259

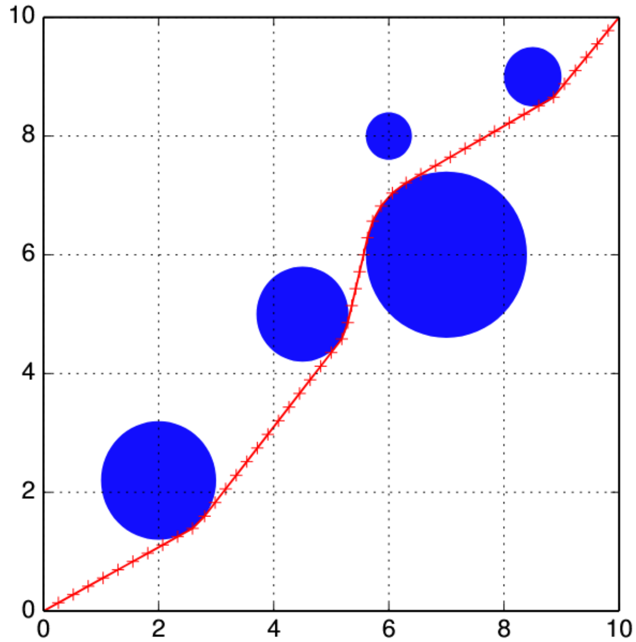
In ordinary CVXPY, we don't need to set the value of our problem variable before we begin, and that's one of the most important parts of convex optimization. But in DCCP problems, the initialization matters a lot because we want to try a bunch of random ones and see if we get the right answer.

These kinds of problems are most interesting when most of our constraints are convex except for some specific conditions – that's where the convex-concave procedure works really well.

Example 260

In a **path-planning with obstacles** problem, we want to find the shortest path connecting two points $a, b \in \mathbb{R}^d$ which avoids some set of disks with specified centers and radii. We can do this by discretizing the path as a sequence of n points, and we minimize L subject to all $||x_i - x_{i-1}||_2 \le \frac{L}{n}$ and all $||x_i - c_j||_2 \ge r_j$. A good initialization for this would be the straight line connecting a and b.

In the example below, we do indeed converge to the global maximum, but there's no guarantee that we would in general:



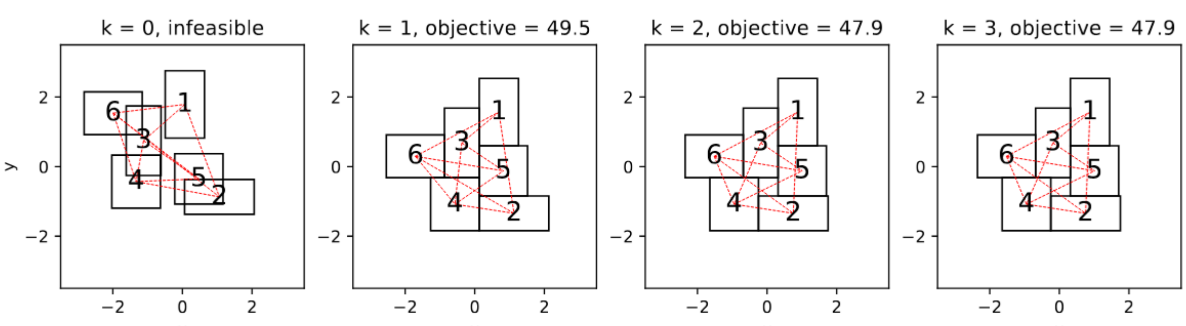
Example 261

We've talked about placement problems before, but in a **floor planning problem** we have a collection of n axis-aligned rectangles, and we want some pairs of them to be close to each other via some least-squares penalty. This would be a convex problem as stated, but now we also add the non-overlapping constraint that the rectangles are disjoint. This is no longer convex, and the way we deal with this is that there are four possible certificates of two rectangles not overlapping: rectangle i is either left of, right of, below, or above rectangle j, which we can express via

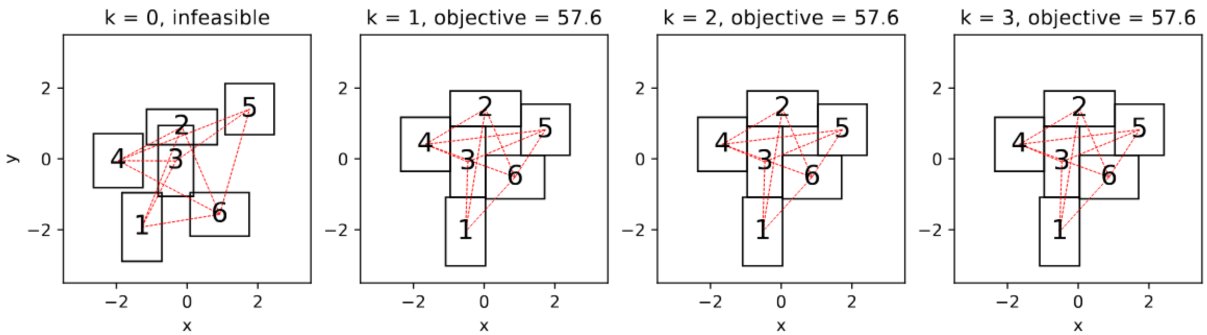
$$\min\{x_i + w_i - x_j, x_j + w_j - x_i, y_i + h_i - y_j, y_j + h_j - y_i\} \leq 0$$

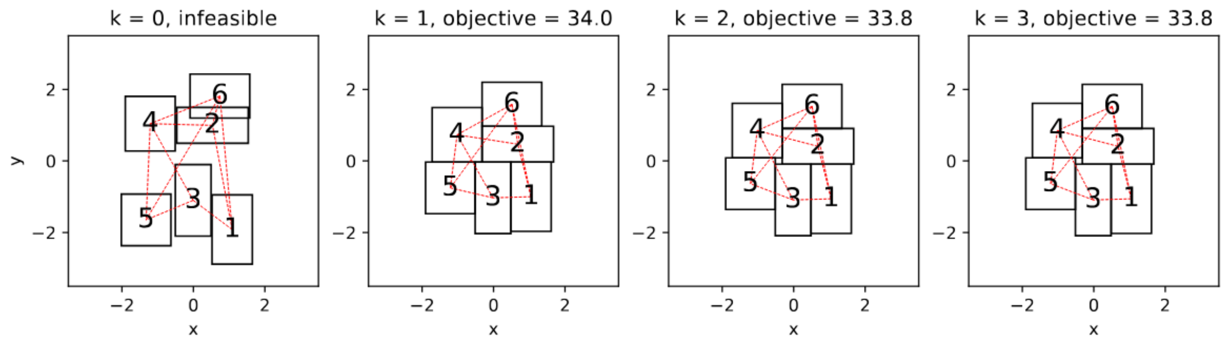
for w, h the widths, heights of the rectangles. In particular, the left-hand side is concave in (x, y) so we can express these requirements as difference-of-convex inequality constraints.

Below is an example of this problem with six rectangles and some random initialization. We start off with an infeasible point, but then at the next iteration we specify the feasibility inequalities and so things are feasible from there on. (In general it may not be cleared to be feasible in one step, but it often is in problems like this.)

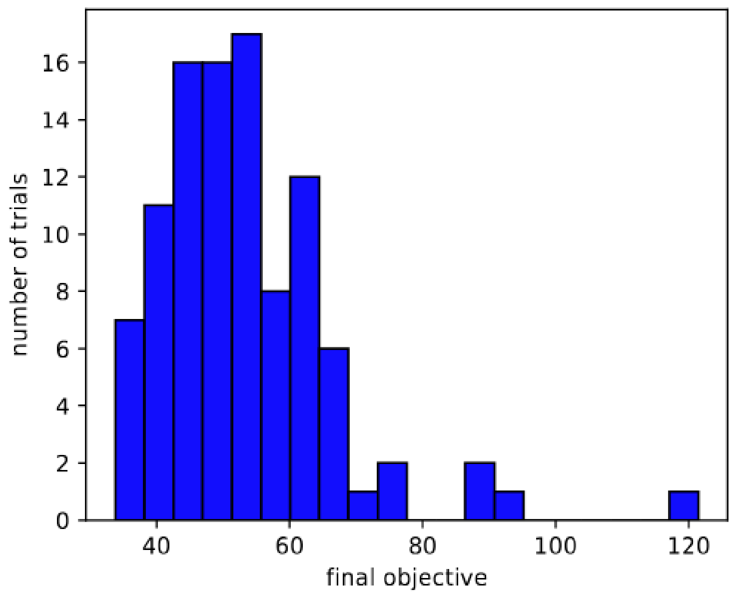


On the other hand, we may lock onto a configuration which is much worse or much better, so this convex-concave procedure does indeed depend on the initial conditions:





In practice, the histogram of final objectives will be rather spread out, as shown below.



Example 262

In collision avoidance, suppose we have N vehicles that want to go from some initial positions to some final positions, and we want to keep some minimum distance apart from each other at all times.

We can then minimize the total length of travel (or give some penalty to fuel use, etc.) while avoiding collisions (and potentially setting some maximum third derivative, etc.). And this all tends to work quite well – for example, if we have vehicles with starting points uniformly around a circle and every point wants to go to its diametrically opposite point, we would discover the roundabout.

Of course, there are no guarantees on convergence or optimality, but we can read the documentation for details on how it actually works (we just have some number of max iterations before halting). There's no definitive proof that our problem is infeasible and so the convex-concave procedure won't claim to know that.

Example 263

Next, we'll discuss $\ell_{1/2}$-regularized regression. Recall that minimizing $||Ax-b||_2^2 + \lambda||x||_1$ will do regression with sparse x; if we instead minimized with $\sum_i |x_i|^{1/2}$ in place of $\sum_i |x_i|$, this would yield even sparser x (because there is an infinite-slope penalty at $x = 0$). This is sort of saying that $\sqrt{|x|}$ is more reasonable than $|x|$ as an approximation to the function which is 0 at 0 and 1 otherwise.

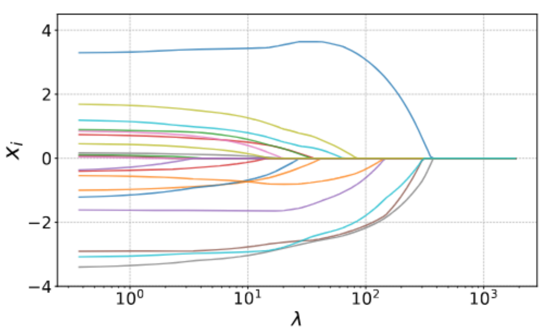
Note that $|x_i|^{1/2}$ is very much not a convex function, so this is not a convex problem. But we can write this problem in epigraph form as

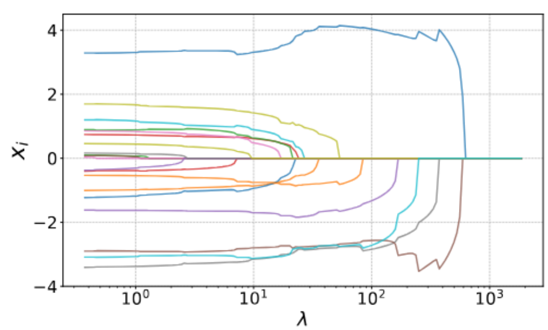
$$\begin{aligned} \text{minimize} \quad & ||Ax-b||_2^2 + \lambda \mathbf{1}^T t \\ \text{subject to} \quad & t \succsim 0, \\ |x_i| \le t_i^2 \text{ for } 1 \le i \le n. \end{aligned}$$

If we linearize the t_i^2 part, it turns out that convex-concave is solving a **weighted ℓ^1 regularization problem**

$$\text{minimize} \quad ||Ax-b||_2^2 + \lambda \sum_{i=1}^{n} \frac{1}{2t_i^k}|x_i|.$$

(so if t_i is small, we have a big penalty for having it be nonzero). The regularization paths for lasso (ℓ_1) versus $\ell_{1/2}$ look as shown; indeed entries extinguish somewhat faster on the right and so we get more sparsity for the "same level of fit."





26 March 10, 2026 (Problem Session)

We'll finish off the course with some more example problems:

Problem 264

Suppose we have a discrete-time stochastic process $(X_1, X_2, \cdots)$ which is **autoregressive** in the sense that each X_{t+1} is Poisson with parameter $\nu\omega^{X_t}$. We are given some data points $x_1, \cdots, x_T$ and want to estimate the parameters ν, ω using maximum likelihood estimation.

For a bit of background, the "standard $AR(1)$ process" (the 1 here referring to first-order, meaning that we just depend on the previous time) takes the form

$$X_t = \alpha + \beta X_{t-1} + \varepsilon_t,$$

where X_{t-1} is the value of the process at the previous step and ε_t is some noise (for example iid Gaussian). If $\beta = 0$ then the signal is just a sequence of iid random variables, but for large β things are a lot closer to being deterministically dependent on the previous values.

Our autoregressive sample is rather different from this – this is not the model we're using, since our X_ts are nonnegative integer-valued (we have a counting process). So instead we use the previous X_t to affect the probability law indirectly. In our case the interpretation is that **the average number of events in a time interval is dependent on the number of such events in a previous interval**. For example, in a run on banks, the number of withdrawals in one day tends to influence the number of withdrawals in the next day (this is an example of a **self-excitatory process**), and similar for the number of people who are infected by a given disease.

The random variable being considered here (Poisson) is a very fundamentally important discrete random variable which comes up in various applications and contexts. The law here is that for a Poisson of parameter λ, for any nonnegative integer k we have

$$\mathbb{P}(X = k) = \frac{e^{-\lambda}\lambda^k}{k!}.$$

(If we look at the probability mass of this random variable, it is integer valued, typically has a peak close to λ, and is roughly bell-curve-shaped for large λ.) And the point is that in our case, we model the stochastic process as a **conditional probability**

$$P(X_t = x_t | X_{t-1} = x_{t-1}) = \frac{e^{-\lambda_t}\lambda_t^{x_t}}{x_t!}, \text{ where } \lambda_t = \nu\omega^{x_{t-1}}.$$

Since λ is also the mean of a Poisson(λ) random variable, what this says is that we dictate the average arrival rate based on the arrival rate in the last time interval.

Remark 265. *The reason Poisson random variables are so ubiquitous is essentially that they model* **rare events in short time-scales.** *Indeed, if we break up an interval* $[0, 1]$ *into* n *time intervals, and in each one we flip a* p*-coin which is heads with probability* $\frac{\lambda}{n}$*, then the number of heads will be roughly Poisson*(λ) *as* $n \to \infty$*:*

$$\begin{aligned} \mathbb{P}(\#heads = k) &= \binom{n}{k} p^k (1-p)^{n-k} \\ &\approx \frac{n^k}{k!} p^k (1-p)^n \\ &\approx \frac{(np)^k}{k!} \exp(-np) \\ &= \frac{\lambda^k e^{-\lambda}}{k!}. \end{aligned}$$

(Here in the second line we use that $n(n-1)\cdots(n-k+1)$ *is approximately* n^k *to leading order, and also that* $(1-p)^k \approx 1$ *since* $p \to 0$ *and* k *is constant. Then we use that* $\lim_{n\to\infty}(1 + \frac{c}{n})^n = e^c$*.) So* $B(n, p)$ *approaches a normal random variable if the mean* np *grows, but it approaches a Poisson random variable if the mean is constant.*

In time-series analysis, **generalized linear models** and **link functions** let us extend ordinary linear regressions to allow for more restrictive formats like our counting process. These models have three components:

1. A **random component**, where we describe the probability law in terms of past behavior (in our case, the mean depends on the behavior from the immediately previous time-instance),
2. A **linear predictor**, which is our best estimate if the randomness is averaged out (in the standard process this

would be $\alpha + \beta X_{t-1}$),

3. A **link function**, which is a monotonic, differentiable function linking the expected value to the linear predictor. Specifically, the function $f(\lambda_t) = \alpha + \beta X_{t-1}$ should be such that regardless of what is on the right-hand side, we produce a nonnegative λ_t when we invert the function.

To get our setting, we take $f(\lambda_t) = \log \lambda_t$ (because the Poisson must have positive mean), so that $\log(\lambda_t) = \alpha + \beta X_{t-1} \implies \lambda_t = e^{\alpha} e^{\beta X_{t-1}}$, which can indeed be reparametrized as $\nu \omega^{X_{t-1}}$. Notice that $\omega = 1$ means we have a constant mean ν and iid random variables; this is indeed the same as having $\beta = 0$. And ν is also the mean we get for any ω if $X_{t-1} = 0$; thus ν is called the **baseline rate**. In general, we call ω the **autoregressive multiplier** (so if $\omega > 1$ we have a excitatory process and if $\omega < 1$ we have an inhibitory one).

Of course, there is still randomness in this process because we only specify the mean of the Poisson random variable, not its value. And so if we want to do maximum likelihood estimation (that is, what ν, ω most likely led to our measurements), we should write it out in terms of the probability mass functions. We have

$$\begin{aligned}\mathbb{P}(X_t = x_t | X_{t-1} = x_{t-1}) &= \frac{\lambda_t^{x_t} e^{-\lambda_t}}{x_t!} \\ &= \frac{(\nu\omega^{x_{t-1}})^{x_t} e^{-(\nu\omega^{x_{t-1}})}}{x_t!},\end{aligned}$$

and because each X_t only depends on the previous one, the overall likelihood is just a product of such terms:

$$\mathcal{L}(x_1, \cdots, x_T) = \mathbb{P}(x_T|x_{T-1})\mathbb{P}(x_{T-1}|x_{T-2}) \cdots \mathbb{P}(x_2|x_1)\mathbb{P}(x_1).$$

(Importantly, **normally we would have to condition on all past events** – for example the first term on the right would usually be $\mathbb{P}(x_T | x_{T-1}, \cdots, x_1)$, but we have the Markov property here. So the given form of our process makes things much easier!) Taking the log so we don't have to worry about products, we can thus write

$$\ell = \log \mathcal{L} = \sum_{t=1}^{T} \left(-\lambda_t + x_t \log \lambda_t - \log x_t!\right).$$

This last term might seem kind of ugly to us, but remember we're maximizing this function over the parameters ν, ω, **not** over our dataset x_t, so it's actually fine to completely drop the $\log x_t!$ terms. Thus the function we wish to maximize is (now plugging in our expressions for λ_t)

$$\tilde{\ell}(\nu, \omega) = \sum_{t=1}^{T} \left(-\nu\omega^{x_{t-1}} + x_t \log(\nu\omega^{x_{t-1}})\right),$$

where we've just defined $x_0 = 0$ for convenience. Further expanding this out a bit using logarithm rules, we have

$$\tilde{\ell}(\nu, \omega) = \sum_{t=1}^{T} \left(-\nu\omega^{x_{t-1}} + x_t \left(\log \nu + x_{t-1} \log \omega\right)\right),$$

so maximizing this is equivalent to minimizing

$$-\tilde{\ell}(\nu, \omega) = \sum_{t=1}^{T} \left(\nu\omega^{x_{t-1}} - x_t \left(\log \nu + x_{t-1} \log \omega\right)\right).$$

Unfortunately, this is not yet in the right form for convex optimization; even though $-\log \nu$ and $-\log \omega$ are indeed convex, the $\nu\omega^{x_{t-1}}$ term is not. And this is where the motivation for the problem pays off: we'll **change variables back** to $\nu = e^{\alpha}$ and $\omega = e^{\beta}$. (Indeed, if we know α, β, we can recover ν, ω, and vice versa.) We now want to perform the optimization problem

$$\text{minimize} \quad \sum_{t=1}^{T} \left(e^{\alpha + \beta x_{t-1}} - x_t \left(\alpha + x_{t-1}\beta \right) \right),$$

and this is okay because we are now optimizing a convex plus affine function.

Turning to implementation now, our problem data has many ts with $X_t = 0$, some substantial fraction with $X_t = 1$, just a few with $X_t = 2$, and a very few rare ones with $X_t = 3$. Furthermore, the large X values usually come with large X values before them, so this is likely an excitatory process but with a low baseline rate. (Indeed, the true parameters were actually $\nu = 0.3$ and $\omega = 1.5$.) And there aren't really any tricky things in the implementation – there are no constraints because α, β can be any real values, and we just need to remember to convert back to our original variables when reporting our parameters.

Problem 266

Finally, we'll conclude with a few remarks about the moving obstacle avoidance problem. We have a vehicle of some mass m which moves in a 2D plane, so we have some positions $p_1, \cdots, p_T$ and some corresponding velocities $v_1, \cdots, v_T$ and force vectors $f_1, \cdots, f_T$. Time increments are h apart, so we have some physics rules for how p relates to v and how v relates to f.

We specify that the vehicle must start at $(0, 0)$ and end at $(1, 0)$, that its initial and final velocity must be exactly what is needed to travel in a straight line, and that it must avoid some collection of moving particles moving in straight lines by staying at least $d^{\min}$ away from them at all times. Our goal is then to minimize the total force $\sum_t ||f_t||_2$ subject to all of these constraints.

The physics rules essentially are discretized versions of kinematics laws, and we have that

$$p_{t+1} = p_t + hv_t, \quad v_{t+1} = v_t + \frac{h}{m} f_t.$$

Viewing this as an optimization problem in the variables p_t, v_t, f_t, we see that everything is okay for convex optimization (the initial and final constraints, and also $||f_t||_2 \le 1$) **except** the avoidance criterion $||p_t - p_t^k||_2 \ge d^{\min}$, since this is essentially the superlevel set of a convex function. Another viewpoint on this is that this last criterion can be written as a difference of two convex functions $d^{\min} - ||p_t - p_t^k||_2 \le 0$ (where f_k is the constant function $d^{\min}$ and g_k is $||p_t - p_t^k||_2$). In other words, this is not in our framework of convex programming, but it is valid in the convex-concave programming viewpoint!

So the way we would solve this is to repeatedly linearize the subtracted function g_k **at our current x value** and then solve the resulting convex optimization problem; notice that because we are linearizing a constraint, this is giving us a **more restrictive problem** to solve than before at each stage. And because our constraints are essentially just disks to avoid (which, in the convex relaxation, become tangent halfspaces that we must stay within), it's plausible that we may converge to the optimal solution with the right initial conditions.