Finm 34800: Modern Applied Optimization

Lectures by Lek-Heng Lim

University of Chicago, Spring 2023

Lecture 1 (Tuesday, September 26) **1**
Overview . . . 1
Convention . . . 2
Univariate optimality conditions . . . 2
Lipschitz functions . . . 7
Newton's method . . . 9
Rates of convergence . . . 10

Lecture 2 (Tuesday, October 3) **12**
Univariate global optimality conditions . . . 12
Convergence analysis of Newton's method . . . 12
Other root finding methods . . . 14
Applications . . . 19
Linear algebra refresher . . . 20

Lecture 3 (Tuesday, October 10) **23**
Convergence and continuity . . . 23
Multivariate derivatives . . . 24
Product, quotient, and chain rules . . . 26
Calculating gradients and Jacobians . . . 28
Calculating Hessians . . . 30

Lecture 4 (Tuesday, October 17) **34**
Linear and quadratic approximations . . . 34
Multivariable optimization . . . 35
Necessary conditions for optimality . . . 36
Sufficient conditions for optimality . . . 38
Applications of unconstrained optimization . . . 40

Lecture 5 (Tuesday, October 24) **45**
More applications of unconstrained optimization . . . 45
Algorithms for nonlinear/convex optimization: line search . . . 49
Search direction from linear approximation . . . 50
Search direction from quadratic approximation . . . 51
Descent direction . . . 51

Wolfe conditions . . . 52
Finding step size . . . 57
Newton method for optimization . . . 58
Starting and stopping . . . 59

Lecture 6 (Tuesday, October 31) 60
Newton method for systems of nonlinear equations . . . 60
Gauss–Newton method for nonlinear least squares . . . 61
Stochastic gradient descent . . . 62
Quasi-Newton methods . . . 63
Davidon–Fletcher–Powell . . . 63
Broyden–Fletcher–Goldfarb–Shanno . . . 65
Barzilai–Borwein step size . . . 65
Constrained optimization . . . 66
Equality constrained optimization . . . 67

Lecture 7 (Tuesday, November 7) 72
Methods for equality constrained optimization . . . 72
Applications of equality constrained optimization . . . 75

Lecture 8 (Tuesday, November 14) 85
Inequality constrained optimization . . . 85
Optimality conditions for inequality constrained optimization . . . 87
Convex optimization . . . 90
Applications of inequality constrained optimization . . . 94

Lecture 1 (Tuesday, September 26)

▶ Overview

Prerequisites:

officially real analysis and linear algebra;

really some familiarity with univariate and multivariate calculus; know how to multiply matrices and vectors.

Objectives:

1. applied and computational real analysis: how the abstract stuff you learned can be applied to real-world problems and how you can compute their solutions;
2. bridges undergraduate real analysis sequence and graduate course on optimization.

Motivation:

$$\boxed{\text{Every problem in the world is an optimization problem.}}$$

In other words, any real-world problem can be formulated as

$$\begin{array}{rrl}
\text{minimize} & f(x_1,\ldots,x_n) & \text{objective function} \\
 & \left.\begin{array}{r} g_1(x_1,\ldots,x_n)=0 \\ \vdots \\ g_m(x_1,\ldots,x_n)=0 \end{array}\right\} & \text{equality constraints} \\
\text{subject to} & \left.\begin{array}{r} h_1(x_1,\ldots,x_n)\le 0 \\ \vdots \\ h_p(x_1,\ldots,x_m)\le 0 \end{array}\right\} & \text{inequality constraints}
\end{array}$$

A huge part of the effort is to take a real-world problem and translate it into this form, i.e., choose f, g_i, h_i so that the solution to the optimization problem is also the solution to your real-world problem. This is what the 'applied' in the course title means.

Course road map:

$$\begin{array}{ccccc}
\boxed{\text{univariate}} & \longrightarrow & \text{multivariate unconstrained} & & \\
 & & \downarrow & & \\
 & & \text{multivariate equality constrained} & & \\
 & & \downarrow & & \\
\text{convex} & \longleftarrow & \text{multivariate inequality constrained} & \longrightarrow & \text{nonsmooth} \\
 & & \downarrow & & \\
 & & \begin{array}{c}\text{multiobjective, combinatorial,}\\ \text{dynamic, stochastic, integer, etc}\end{array} & &
\end{array}$$

In the first lecture, we will discuss optimization of real-valued functions of one real variable defined on a subset $X \subset \mathbb{R}$, i.e. $f\colon X \to \mathbb{R}$; typically $X = [a, b]$ or (a, b). Aside from being the simplest case, it is also important for multivariate optimization because any multivariate minimization or maximization problem can be solved coordinatewise

$$\min_{(x,y,z)\in\mathbb{R}^3} f(x, y, z) = \min_{x\in\mathbb{R}}\left[\min_{y\in\mathbb{R}}\left(\min_{z\in\mathbb{R}} f(x, y, z)\right)\right]$$

and likewise for more variables or for max in place of min. However typically there are more clever ways than just searching the coordinate directions — we could pick a special direction

$$v = \alpha x + \beta y + \gamma z$$

to search. Such algorithms are called *line search* algorithms, where we essentially convert a multivariate optimization problem into a sequence of univariate ones. For example we could just pick

$$\alpha = -\frac{\partial f}{\partial x}, \qquad \beta = -\frac{\partial f}{\partial y}, \qquad \gamma = -\frac{\partial f}{\partial z}$$

and this gives us *steepest descent*, the backbone of all modern machine learning algorithms

▶ Convention

We will adopt the notation that $\operatorname{dom}(f)$ refers to the largest domain for a function f. For example, $\operatorname{dom}(\log) = (0, \infty)$, $\operatorname{dom}(\sin^{-1}) = [-1, 1]$, $\operatorname{dom}(\sqrt{\cdot}) = [0, \infty)$, etc.

In optimization we sometimes write $f : \mathbb{R} \to \mathbb{R}$ even when $\operatorname{dom}(f) \neq \mathbb{R}$, the reason is that we adopt the convention

$$f(x) = +\infty \qquad \text{if } x \notin \operatorname{dom}(f).$$

Since we will formulate all optimization problems as minimization problems, we will never need to worry about points outside the domain.

▶ Univariate optimality conditions

For univariate optimization problems, we only deal with unconstrained problem

$$\min_{x\in\mathbb{R}} f(x)$$

or constrained problem of the form

$$\min_{x\in I} f(x)$$

where I is an interval, i.e., $I = (a, b)$, $(a, b]$, $[a, b)$, or $[a, b]$ where $a \le b$ and $a, b \in \mathbb{R} \cup \{\pm\infty\}$. More generally, a constrained problem over a union of disjoint intervals $I_1, \dots, I_m$ can be reduced to m optimization problems

$$\min_{x\in I_1\cup\cdots\cup I_m} f(x) = \min_{j=1,\dots,m}\left[\min_{x\in I_j} f(x)\right].$$

Almost all subsets of $\mathbb{R}$ that arise in real-life applications are of this form.

The most important conditions in optimization are so-called *optimality conditions* that tells you whether a point is a candidate for a minimum or maximum. Depending on whether they involve just on the values of f, the values of its derivative f', or the values of its second derivatives f'', they are called zeroth, first, or second order optimality conditions respectively. There are higher order optimality conditions but we rarely use them in practice. Optimality conditions are further divided into necessary or sufficient conditions, or for local or global optimality.

The next two sets of definitions apply to any real-valued function, not necessarily differentiable or continuous. They may either be taken as definitions or *zeroth order optimality conditions* that are both necessary and sufficient.

Definitions 1.1 (Zeroth order condition for local optimality)**.** Let $f\colon X \to \mathbb{R}$ be any function and $x_0 \in X$. Then

- $f(x_0)$ is a *local minimum* if $f(x) \geq f(x_0)$ for all x in some neighborhood of x_0;
- x_0 is a *local minimizer* if $f(x_0)$ is a local minimum.
- $f(x_0)$ is a *local maximum* if $f(x) \leq f(x_0)$ for all x in some neighborhood of x_0;
- x_0 is a *local maximizer* if $f(x_0)$ is a local maximum.

Recall that a *neighborhood* or more precisely a δ-neighborhood of x_0 in X is a set of the form

$$\{x \in X : |x - x_0| < \delta\}$$

for some $\delta > 0$. In other words, it is the set of points in X that falls within distance from x_0 or alternatively $(x_0 - \delta, x_0 + \delta) \cap X$. The set $(x_0 - \delta, x_0) \cap X$ is called a *left neighborhood* and the set $(x_0, x_0 + \delta) \cap X$ is called a *right neighborhood*.

Definitions 1.2 (Zeroth order condition for global optimality)**.** Let $f\colon X \to \mathbb{R}$ be any function and $x_0 \in X$. Then

- $f(x_0)$ is a *global minimum* if $f(x_0) = \inf_{x \in X} f(x)$;
- x_0 is a *global minimizer*.
- $f(x_0)$ is a *global maximum* if $f(x_0) = \sup_{x \in X} f(x)$;
- x_0 is a *global maximizer*.

We also use the term local/global *optimum* to refer to a value that is either a local/global minimum or maximum, and local/global *optimizer* to refer to a point that is either a local/global minimizer or maximizer.

The next result is the best known condition for local optimality, stated in a more general form that does not assume f is differentiable.

Theorem 1.3 (First order necessary condition for local optimality)**.** *If x_0 is a local optimizer of f, then either $f'(x_0) = 0$ or $f'(x_0)$ does not exist.*

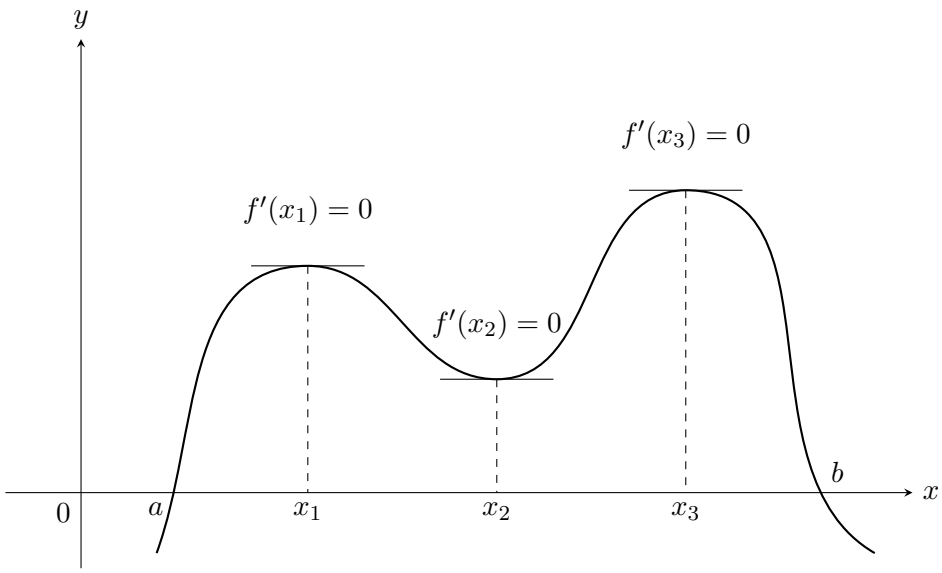


Figure 1: $X = [a, b]$; x_1, x_2, x_3 are stationary points; x_1, x_3 are local maximizers; x_2 is local minimizer; x_3 is global maximizer; a, b are global minimizers.

Proof. Suppose x_0 is a local minimizer. There exists $\delta > 0$ such that for all $x \in B_\delta(x_0) := \{x \in \mathbb{R} : |x - x_0| < \delta\}$, we have $f(x) \geq f(x_0)$. Hence

$$\frac{f(x) - f(x_0)}{x - x_0} \begin{cases} \geq 0 & \text{if } x_0 < x < x_0 + \delta \\ \leq 0 & \text{if } x_0 - \delta < x < x_0. \end{cases}$$

If $f'(x_0)$ exists, then

$$f'(x_0) = \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0} \geq 0, \quad \text{(right limit)}$$
$$f'(x_0) = \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0} \leq 0. \quad \text{(left limit)}$$

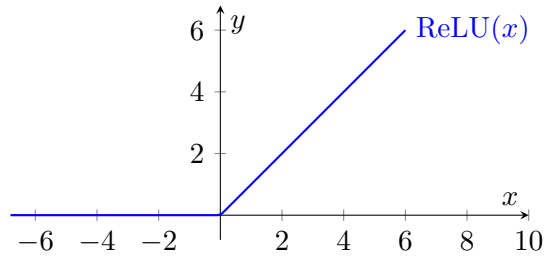
Hence $f'(x_0) = 0$. □

Definition 1.4. If $f'(x_0) = 0$, then x_0 is called a *stationary point* or *critical point* and $f(x_0)$ is called a *stationary value* or *critical value*.

By Theorem 1.3, if $f \in C^1(X)$, then every local optimizer must be a stationary point.

Example 1.5 (ReLU function)**.** It is possible for $f'(x_0)$ to not exist but x_0 be a local minimizer. For example, the function $f : \mathbb{R} \to \mathbb{R}$ given by $f(x) = \max(x, 0)$ has a local minimum at $x = 0$ but f is not differentiable at 0.

This by the way is called the ReLU function, or *rectified linear unit*, and has become enormously important in deep neural networks. □



Theorem 1.6 (Mean value theorem)**.** *Let $f\colon [a,b] \to \mathbb{R}$ be continuous on $[a,b]$ and differentiable on (a,b). Then there exists $c \in (a,b)$ such that*

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

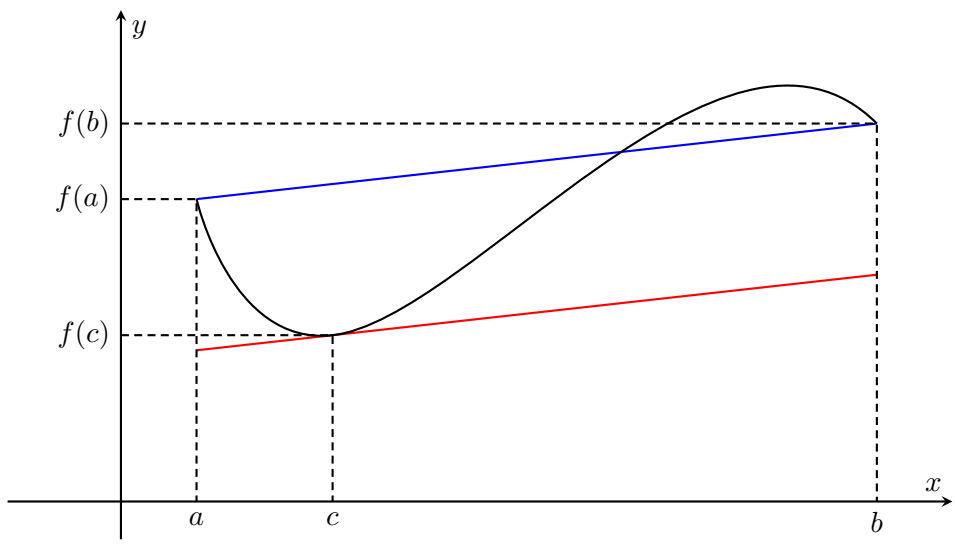


Figure 2: Mean value theorem: slope of tangent $f'(c) = [f(b) - f(a)]/(b - a)$ slope of secant.

Recall that a function is said to be *monotone increasing* (or nondecreasing) if

$$x_1 < x_2 \quad \Rightarrow \quad f(x_1) \leq f(x_2),$$

and *monotone decreasing* (or nonincreasing) if

$$x_1 < x_2 \quad \Rightarrow \quad f(x_1) \geq f(x_2).$$

We say *strictly increasing* or *strictly decreasing* if the inequalities on the right hand sides are strict.

Corollary 1.7 (First order sufficient condition for monotonicity)**.** *If f is differentiable on (a,b), then*

(i) *$f'(x) \geq 0$ for all $x \in (a,b) \Rightarrow f$ is monotone increasing;*

(ii) $f'(x) = 0$ *for all* $x \in (a, b) \Rightarrow f$ *is constant;*

(iii) $f'(x) \leq 0$ *for all* $x \in (a, b) \Rightarrow f$ *is monotone decreasing.*

Proof. Suppose $x_1, x_2 \in (a, b)$ with $x_1 < x_2$. Then f is continuous on $[x_1, x_2]$ and differentiable on (x_1, x_2), so by the mean value theorem, there exists c with $x_1 < c < x_2$ such that

$$f'(c) = \frac{f(x_2) - f(x_1)}{x_2 - x_1}.$$

Since $x_2 > x_1$ and $f'(c) \geq 0$, we have $f(x_2) - f(x_1) \geq 0$, i.e., $f(x_2) \geq f(x_1)$. The other statements are proved similarly. □

Exercise 1.8. Deduce similar statements for $f'(x) > 0$ and $f'(x) < 0$.

The next two results help us ascertain whether a local optimizer is a minimizer or maximizer.

Theorem 1.9 (First order sufficient condition for local optimality)**.** *If f is continuous at x_0 and differentiable in a neighborhood of x_0, then*

- *if $f'(x) < 0$ in a left neighborhood of x_0 and $f'(x) > 0$ in a right neighborhood of x_0, then x_0 is a minimizer;*
- *if $f'(x) > 0$ in a left neighborhood of x_0 and $f'(x) < 0$ in a right neighborhood of x_0, then x_0 is a maximizer.*

For an illustration, see Figure 1: the point x_2 is a local minimizer and the points x_1, x_3 are local maximizers. But note that you do not need differentiability at x_0 to apply the first order sufficient condition. For example, we can deduce that $f(x) = |x|$ has a minimizer at $x = 0$ by applying Theorem 1.9.

Theorem 1.10 (Second order sufficient condition for local optimality)**.** *Let $f \in C^2(X)$ and x_0 be a local optimizer, so $f'(x_0) = 0$. Then*

- *if $f''(x_0) > 0$, then x_0 is a minimizer;*
- *if $f''(x_0) < 0$, then x_0 is a maximizer.*

A common rookie mistake is to overestimate the usefulness of Theorem 1.10 and underestimate that of Theorem 1.9, as you will see in Homework 1.

Exercise 1.11. Prove Theorems 1.9 and 1.10.

Why did we start with 'local optimizer' instead of 'stationary point' in Theorems 1.9 and 1.10? This is to avoid having to deal with *inflection points*, which you probably think you know but are actually trickier to define properly than you think. We will not define it formally since we won't be using it.

For most of our purposes, we need a stronger refinement of mean value theorem — Taylor's theorem for $d = 1$ or $d = 2$.

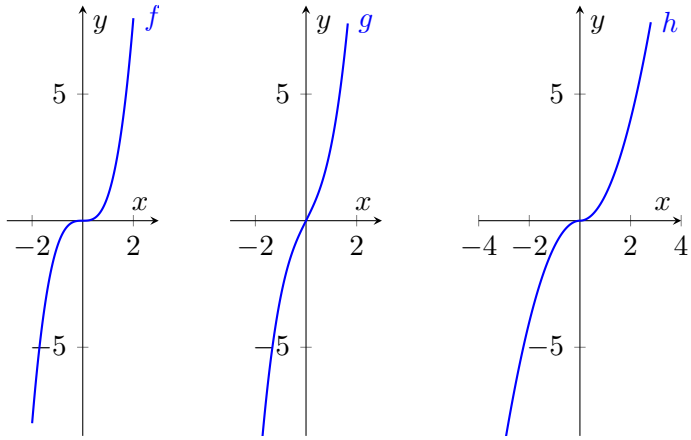


Figure 3: f, g, h have inflection point at $x = 0$. *Left*: $f(x) = x^3$ has $f(0) = f'(0) = f''(0) = 0$. *Center*: $g(x) = x^3 + 2x$ has $g(0) = g''(0) = 0$, $g'(0) \neq 0$. *Right*: $h(x) = -x^2$ if $x < 0$ and x^2 if $x > 0$ has $h(0) = h'(0) = 0$, $h''(0)$ does not exist.

Theorem 1.12 (Taylor)**.** *If f has continuous derivatives of order up to and including $d+1$ in some neighborhood of x_0, then*

$$f(x) = \sum_{k=0}^{d} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + R_d(x),$$

where the remainder *term is*

$$\begin{aligned} R_d(x) &= \frac{1}{d!}\int_{x_0}^{x} f^{(d+1)}(t)(x-t)^d \,\mathrm{d}t && \text{(integral form)}, \\ &= \frac{1}{d!} f^{(d+1)}(\xi)\frac{(x-x_0)^{d+1}}{d+1} && \text{(mean value form)}, \end{aligned}$$

for some ξ between x_0 and x.

Proof. Fundamental Theorem of Calculus. □

The integral form is usually much more useful in optimization, not least because it does not involve an unknown ξ that we don't have much control over.

But really what is most important for us is a Taylor's theorem for "$d = 1.5$" — a function that is more than C^1 but less than C^2, or more precisely, a function $f \in C^1(X)$ such that $f' \in \mathrm{Lip}_\gamma(X)$. We will discuss this next.

▶ Lipschitz functions

Definition 1.13 (Lipschitz)**.** A function $f : X \to \mathbb{R}$ is said to be *Lipschitz* if there is some $\gamma < \infty$ such that

$$|f(x) - f(y)| \leq \gamma|x - y|$$

for all $x, y \in X$. We write $f \in \mathrm{Lip}_\gamma(X)$.

Lemma 1.14. *Let $f : X \subset \mathbb{R} \to \mathbb{R}$ be a differentiable function. Then f' is bounded if and only if f is Lipschitz on X.*

Proof. Suppose $|f'(x)| \leq \gamma$ for all x, and let $x_1, x_2 \in X$ be distinct. By the Mean Value Theorem, there exists ξ between x_1 and x_2 such that

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} = f'(\xi).$$

Taking absolute values and re-arranging,

$$\frac{|f(x_2) - f(x_1)|}{|x_2 - x_1|} = |f'(\xi)| \leq \gamma \qquad \Rightarrow \qquad |f(x_2) - f(x_1)| \leq \gamma |x_2 - x_1|$$

Hence $f \in \mathrm{Lip}_\gamma(X)$. Conversely, if $f \in \mathrm{Lip}_\gamma(X)$, then

$$|f(x) - f(x_0)| \leq \gamma |x - x_0|$$

for every x and x_0 in X. Re-arranging,

$$\left| \frac{f(x) - f(x_0)}{x - x_0} \right| \leq \gamma$$

The inequality holds as $x \to x_0$, so

$$|f'(x)| = \left| \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} \right| = \lim_{x \to x_0} \left| \frac{f(x) - f(x_0)}{x - x_0} \right| < \gamma.$$

□

In optimization, the Lipschitz condition usually arises in the derivative, not the function. That is, $f' \in \mathrm{Lip}_\gamma(X)$. This is a stronger condition than $f \in C^1(X)$ but weaker than $f \in C^2(X)$ and turns out to be the right level of regularity for optimization. Assuming that X is compact, then

$$f \in C^2(X) \quad \Rightarrow \quad f' \in \mathrm{Lip}_\gamma(X) \quad \Rightarrow \quad f \in C^1(X),$$

so the condition $f' \in \mathrm{Lip}_\gamma(X)$ is a bit like saying that "$f \in C^{1.5}(X)$."

Lemma 1.15. *Let $f \colon X \to \mathbb{R}$ be C^1 and $f' \in \mathrm{Lip}_\gamma(X)$. Then for any $x, y \in X$,*

$$|f(y) - f(x) - f'(x)(y - x)| \leq \frac{\gamma}{2}(y - x)^2.$$

Proof. By the Fundamental Theorem of Calculus,

$$f(y) - f(x) = \int_x^y f'(z) \, \mathrm{d}z$$

Since $f'(x)$ is constant in z,

$$f(y) - f(x) - f'(x)(y - x) = \int_x^y f'(z) - f'(x) \, \mathrm{d}z$$

Applying the change of variables $z = x + t(y - x)$,

$$\int_x^y f'(z) - f'(x)\,\mathrm{d}z = \int_0^1 \big(f'(x + t(y - x)) - f'(x)\big)\,(y - x)\,\mathrm{d}t.$$

Taking absolute values and applying the Lipschitz condition on f',

$$\begin{aligned}
|f(y) - f(x) - f'(x)(y - x)| &= \left|\int_0^1 (f'(x + t(y - x)) - f'(x))(y - x)\,\mathrm{d}t\right| \\
&\leq \int_0^1 \big|f'(x + t(y - x)) - f'(x)\big|\,|y - x|\,\mathrm{d}t \\
&\leq \int_0^1 \gamma|t(y - x)||y - x|\,\mathrm{d}t \\
&= \gamma|y - x|^2 \cdot \int_0^1 t\,\mathrm{d}t \\
&= \frac{\gamma}{2}(y - x)^2.
\end{aligned}$$

□

Note that this lemma is stronger than Taylor's Theorem since we do not require f to be twice differentiable. Why is this important? The lemma gives us a linear approximation of f:

$$f(y) = f(x) + f'(x)(y - x) + \text{error},$$

where

$$|\text{error}| \leq \frac{C}{2}|y - x|^2.$$

Optimization algorithms all based on either linear or quadratic approximations. Quadratic:

$$f(y) = f(x) + f'(x)(y - x) + \frac{f''(x)}{2}(y - x)^2 + \text{ error}.$$

▶ Newton's method

We now turn to Newton's method for univariate optimization. In the univariate case, it is also called the Newton–Raphson method, where the goal is to find roots of a nonlinear equation $g(x) = 0$. In the context of optimization, the function is usually some derivative, i.e., $g = f'$. In the multivariate case, Newton's method is used to find roots of system of nonlinear equations:

$$\begin{cases}
g_1(x_1, \ldots, x_n) = 0 \\
g_2(x_1, \ldots, x_n) = 0 \\
\qquad\qquad\vdots \\
g_n(x_1, \ldots, x_n) = 0.
\end{cases}$$

In the context of optimization, the functions are partial derivatives, i.e., $g_i = \partial f / \partial x_i$.

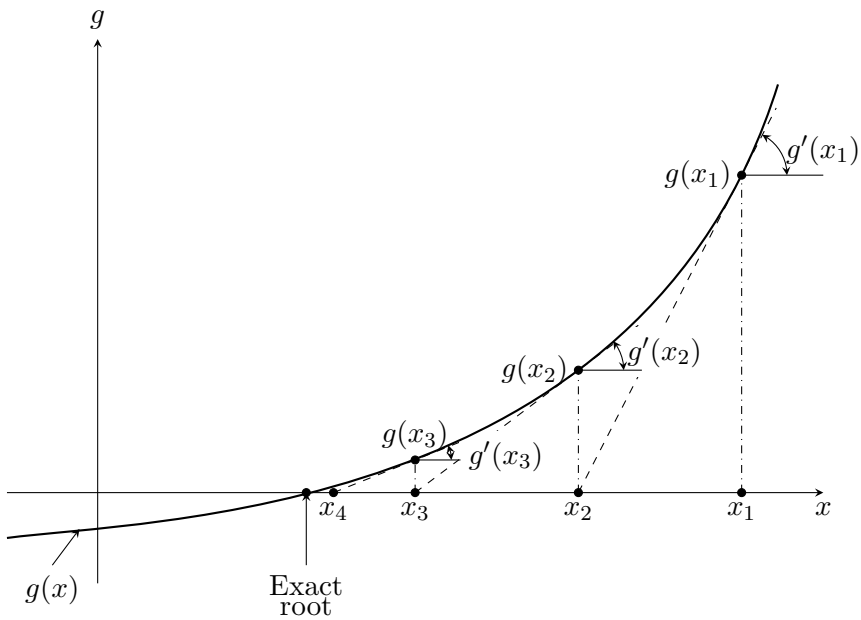


Figure 4: Newton method applied to find root of g.

The idea is to use tangent lines to approximate the zeroes of g: if ℓ is the tangent line to g at x, then $\ell(y) = g(x) + g'(x)(y - x)$, and $\ell(y) = 0$ when

$$y = x - \frac{g(x)}{g'(x)}.$$

Iterating this procedure using y as the next best guess, we obtain

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}, \qquad k = 0, 1, 2, \ldots.$$

Ideally, $x_k \to x_*$, where $g(x_*) = 0$. Newton's method can fail, for example, if g' is unbounded or is zero. Thus we need to identify a class of functions or impose conditions for which we can find zeros.

Example 1.16 (Square root)**.** Let $r \in \mathbb{R}^+$; we compute $\sqrt{r}$, i.e., x_* such that $g(x) = x^2 - r = 0$. Newton's method is:

$$x_{k+1} = x_k - \frac{x_k^2 - r}{2x_k} = \frac{1}{2}\left(x_k + \frac{r}{x_k}\right).$$

The number of correct digits doubles every iteration, a phenomenon known as quadratic convergence. For example, with $r = 17$ and $x_0 = 4$,

▶ Rates of convergence

Definition 1.17. Let $(x_k)_{k=1}^{\infty}$ be a sequence converging to x_*. We say $(x_k)_{k=1}^{\infty}$:

(i) *converges* if

$$\lim_{k\to\infty} |x_k - x_*| = 0;$$

k	x_k
0	4.000000000000000
1	4.125000000000000
2	4.123106060606061
3	4.123105625617683
4	4.123105625617661

(ii) *converges linearly* if there is some $N \in \mathbb{N}$ and $0 < c < 1$ such that

$$|x_{k+1} - x_*| \leq c|x_k - x_*|$$

whenever $k > N$;

(iii) *converges superlinearly* if there is some $N \in \mathbb{N}$ and $(c_k)_{k=1}^{\infty}$ converging to 0 such that

$$|x_{k+1} - x_*| \leq c_k|x_k - x_*|$$

whenever $k > N$;

(iv) *converges sublinearly* if there is some $N \in \mathbb{N}$ and $(c_k)_{k=1}^{\infty}$ converging to 1 such that

$$|x_{k+1} - x_*| \leq c_k|x_k - x_*|$$

whenever $k > N$;

(v) *converges quadratically* if there is some $N \in \mathbb{N}$ and $c > 0$ such that

$$|x_{k+1} - x_*| \leq c|x_k - x_*|^2$$

whenever $k > N$.

N.B. the terms "q-linearly convergent" and "q-superlinearly" are occasionally used for what we called linear and superlinear, respectively. Higher order convergences exist (e.g., cubic convergence, with "3" instead of "2" in the quadratic convergence definition), but in practice, quadratic convergence is good enough.

Here are canonical examples of sequences converging at different rates.

Example 1.18 (Convergence rates)**.** Let $\gamma \in (0, 1)$. Then

- $(1/k)_{k=1}^{\infty}$ exhibits sublinear convergence
- $(\gamma^k)_{k=1}^{\infty}$ exhibits linear convergence
- $(\gamma^{k^2})_{k=1}^{\infty}$ exhibits superlinear convergence
- $(\gamma^{2^k})_{k=1}^{\infty}$ exhibits quadratic convergence

You would be asked to check these against their definitions in Homework 1.

Lecture 2 (Tuesday, October 3)

▶ Univariate global optimality conditions

In the last lecture we have provided several useful conditions to check for optimality — first order, second order, neccessary, sufficient — but they are all for *local* optimality. In principle, for univarate optimization, we can use local optimality to get global optimality. For an open interval (a, b) this is straightforward, just find all the stationary points or local optimizers and pick the one where f has the lowest value. For example suppose f is differentiable, then suppose $f'(x) = 0$ at $x \in \{x_1, \ldots, x_m\}$, then we just compare the values $f(x_1), \ldots, f(x_m)$ and pick the x_k where $f(x_k) = \min\{f(x_1), \ldots, f(x_m)\}$.

For other types of interval $[a, b)$ or $(a, b]$ or $[a, b]$, we will need to check the boundary points too. For example for the case $[a, b]$, in addition the the points $x_1, \ldots, x_m \in (a, b)$ where $f'(x) = 0$, we need to check the values $f(a)$ and $f(b)$ to see if they might be even smaller.

This procedure in principle gives us the global minimizers for a univariate optimization problem as long as m is not infinite but it can be prohibitively expensive if m is exponentially large.

▶ Convergence analysis of Newton's method

In univariate optimization, the most important optimality condition is the first order necessary condition $f'(x) = 0$. Since f' can be any function g, the problem reduces to a root finding problem, i.e., find an $x_* \in (a, b)$ so that

$$g(x_*) = 0.$$

We saw in the last lecture that Newton's method is a great way to do this. Its biggest advantage is that it is very, very fast, essentially the iterates x_k converges to the root x_* as fast as a doubly expoential sequence $1/2^{2^k}$ converges to 0. We now give a proper version of this statement and a proof. Such proofs depend very much on the assumptions. Here we give a set of practically useful assumptions that extend readily to a multivariate scenario.

Theorem 2.1 (Quadratic convergence of Newton's method)**.** *Let $g\colon (a, b) \to \mathbb{R}$ and $g' \in \mathrm{Lip}_\gamma(a, b)$. Suppose $|g'(x)| \geq \rho > 0$ for all $x \in (a, b)$. If $g(x) = 0$ has solution $x_* \in (a, b)$, then there exists some $\eta > 0$ such that if $|x_0 - x_*| < \eta$, then the sequence $(x_k)_{k=1}^\infty$ defined by*

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$$

converges to x_* quadratically. Furthermore,

$$|x_{k+1} - x_*| \leq \frac{\gamma}{2\rho}|x_k - x_*|^2. \tag{2.2}$$

Proof. Let $0 < \tau < 1$ be fixed, and choose $\delta > 0$ be such that $(x_* - \delta, x_* + \delta) \subset (a, b)$. Put $\eta = \min\{\delta, 2\rho\tau/\gamma\}$. We claim that

$$|x_{k+1} - x_*| \leq \tau|x_k - x_*| < \eta$$

for all $k \in \mathbb{N}$. By assumption, we have

$$|x_0 - x_*| < \eta.$$

Next

Final Exam Question

$$\begin{aligned}
|x_1 - x_*| &= \left| x_0 - x_* - \frac{g(x_0)}{g'(x_0)} \right| \\
&= \frac{1}{|g'(x_0)|} \cdot \left| x_0 g'(x_0) - x_* g'(x_0) - g(x_0) \right| \\
&= \frac{1}{|g'(x_0)|} \cdot \left| g(x_*) - g(x_0) - g'(x_0)(x_* - x_0) \right| \\
&\le \frac{1}{|g'(x_0)|} \cdot \frac{\gamma}{2} |x_* - x_0|^2 \qquad \text{(by “}C^{1.5}\text{ Taylor's theorem” from earlier)} \\
&\le \frac{\gamma}{2\rho} |x_* - x_0|^2 \qquad \text{(this gives (2.2) for } k = 0\text{)} \\
&\le \frac{\gamma}{2\rho} \cdot \eta |x_* - x_0| \\
&\le \frac{\gamma}{2\rho} \cdot \frac{2\rho\tau}{\gamma} \cdot |x_* - x_0| \\
&= \tau |x_* - x_0| \\
&\le \tau\eta < \eta.
\end{aligned}$$

Since $|x_1 - x_*| < \eta$, the same argument can be repeated for $|x_2 - x_*|$, etc., to finish. Note that the τ factor is needed to guarantee strict inequality $|x_k - x_*| < \eta$ at every step. $\square$

The condition $|x_0 - x_*| < \eta$ means that Newton's method is only quadratically convergent if we start sufficiently close to the solution x_*. In fact the method may not even be convergent if we don't have some kind of “sufficiently close” assumption.

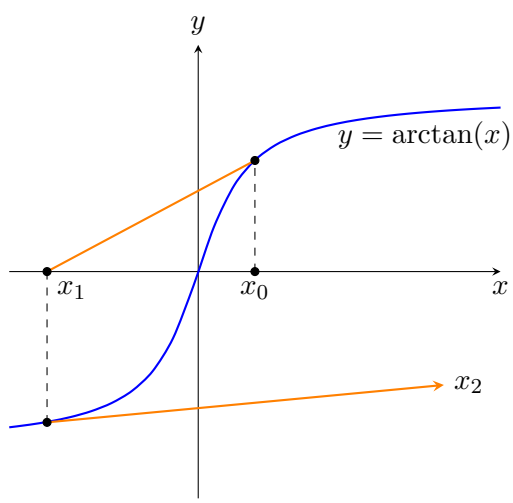


Figure 5: Newton method applied to $\arctan(x) = 0$.

For example, if we take $g = \arctan$, then the Newton's iteration is

$$x_{k+1} = x_k - (x_k^2 + 1) \arctan x_k.$$

If we choose $x_0 = 3$, then $x_1 = -9.4904, x_2 = 124.0007, \ldots,$ and $|x_k| \to \infty$.

▶ Other root finding methods

Despite the simplicity of Newton's method, it is not always possible to perform the procedure: for example, the derivative may be unbounded, close to 0, or not feasible to compute. Here, we discuss some weaker variants of Newton's method and their advantages/disadvantages.

Secant method: Avoids derivatives by using only function evaluations. Using the approximation

$$g'(x_k) \approx \frac{g(x_k) - g(x_{k-1})}{x_k - x_{k-1}},$$

modify Newton's method to

$$x_{k+1} = x_k - g(x_k)\left[\frac{x_k - x_{k-1}}{g(x_k) - g(x_{k-1})}\right].$$

It can be shown (and the proof is very similar to Theorem 2.1) that the secant method exhibits superlinear convergence: if $g \in C^2$ and x_0, x_1 are sufficiently close to x_*, then

$$|x_{k+1} - x_*| \approx c|x_k - x_*|^{\phi},$$

where $\phi = (1+\sqrt{5})/2 \approx 1.618$ is the golden ratio, i.e., the largest real zero of $\phi^2 - \phi - 1 = 0$. The "sufficiently close" assumption is crucial here as it is in Newton's method. Figure 6 shows a divergent instance of secant method.

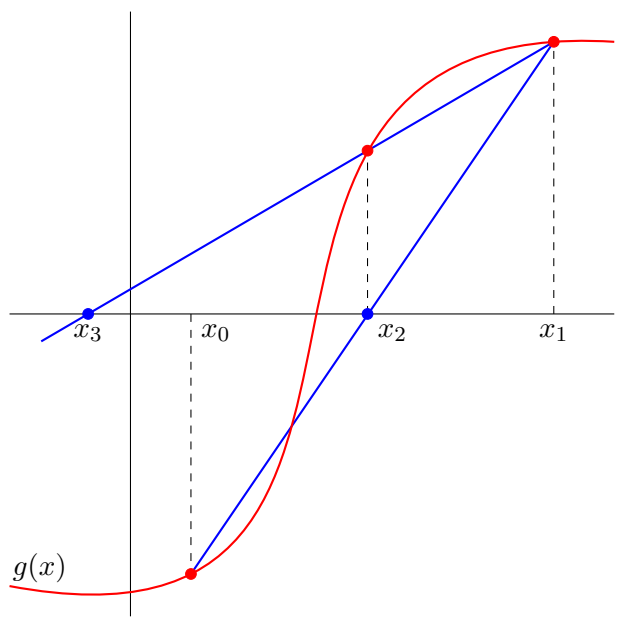


Figure 6: Secant method can diverge. The secant line in the interval $[x_2, x_1]$ takes the next iterate x_3 outside the original interval $[x_0, x_1]$. But if we had used the secant line in the interval $[x_0, x_2]$ to generate x_3, we would have been fine, this leads to *regula falsi*.

There are other variants of secant method. Note that secant method uses linear interpolation — since two points x_k and x_{k-1} determine a line. Now if we use three

points x_k, x_{k-1}, x_{k-2}, then we can pass a quadratic curve through them and this is called quadratic interpolation. *Müller's method* is essentially a quadratic variant of secant method and it improves the convergence order to $\alpha \approx 1.84$ where α is the largest real zero of $\alpha^3 - \alpha^2 - \alpha - 1 = 0$. The multivariate version of secant method is known as *Broyden's method*. We shall see this when we discuss multivariate quasi-Newton methods.

Steffensen's method: Avoids derivatives and preserves quadratic convergence using "divided difference":

$$x_{k+1} = x_k - \frac{g(x_k)^2}{g(x_k + g(x_k)) - g(x_k)}.$$

The multivariate Steffensen's method usage of divided difference is very expensive and should never be used. Even in the univariate case, the multiple function evaluations can be expensive.

In summary, the derivative-based methods discussed thusfar take the form

$$x_{k+1} = x_k - \frac{g(x_k)}{h(x_k)},$$

where h captures the local rate of change near x_*:

$$h(x_k) = g'(x_k) \tag{Newton}$$

$$h(x_k) = \frac{g(x_k) - g(x_{k-1})}{x_k - x_{k-1}} \tag{Secant}$$

$$h(x_k) = \frac{g(x_k + g(x_k)) - g(x_k)}{g(x_k)} \tag{Steffensen}$$

Algorithm 1 Method of bisection

Require: $g(a_0)g(b_0) < 0$, $\varepsilon > 0$

```
procedure BISECTION(a_0, b_0, ε)
    k ← 0
    x_k ← (a_k + b_k)/2
    while |b_k − a_k| > ε do
        if g(a_k)g(x_k) < 0 then
            x_k ← a_k
        else
            x_k ← b_k
        end if
        k ← k + 1
    end while
    return x_* := x_k
end procedure
```

Method of bisection: Only needs sign, not magnitude or value, of $g(x)$. It relies on:

Theorem 2.3 (Intermediate Value Theorem)**.** *If $g \in C([a,b])$ and $g(a)g(b) < 0$, then there is some $x_* \in (a,b)$ such that $g(x_*) = 0$.*

The idea is as follows: set $c = (a+b)/2$ (i.e., bisect $[a,b]$) and evaluate the function at c.

- if $g(a)g(c) < 0$, then $x_* \in [a,c]$; set $b = c$
- if $g(c)g(b) < 0$, then $x_* \in [c,b]$; set $a = c$

Repeating this procedure, we obtain a sequence of nested intervals

$$[a,b] = [a_0,b_0] \supset [a_1,b_1] \supset [a_2,b_2] \supset \cdots \tag{2.4}$$

Note that

$$a_0 \le a_1 \le \cdots \le b_0$$

so the sequence $(a_k)_{k=0}^{\infty}$ is monotone increasing and bounded above. Similarly, since

$$b_0 \ge b_1 \ge b_2 \ge \cdots \ge a_0,$$

the sequence $(b_k)_{k=0}^{\infty}$ is monotone decreasing and bounded below. Hence $\lim_{k\to\infty} a_k$ and $\lim_{k\to\infty} b_k$ exist. The length of each interval is $b_k - a_k$, and

$$b_k - a_k = \frac{1}{2}(b_{k-1} - a_{k-1}) = \cdots = \frac{1}{2^k}(b_0 - a_0),$$

so

$$\lim_{k\to\infty}(b_k - a_k) = (b_0 - a_0) \cdot \lim_{k\to\infty} \frac{1}{2^k} = 0.$$

Hence

$$\lim_{k\to\infty} a_k = \lim_{k\to\infty} b_k = x_*.$$

Let x_k be the midpoint at the kth step. Then

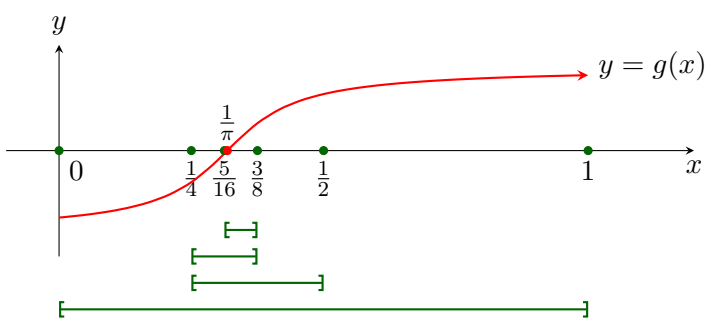


Figure 7: Pictorial view of the method of bisection.

$$|x_{k+1} - x_*| \le \frac{1}{2^k}|b-a|.$$

If we want an answer up to ε-precision, i.e. $|x_{k+1} - x_*| < \varepsilon$, we require

$$k = \left\lceil \log_2\left(\frac{b-a}{\varepsilon}\right)\right\rceil.$$

Hence the method of bisection is linearly convergent. More importantly, it is *globally convergent*, i.e., it converges regardless of the choice of x_0. Nevertheless it has no known multivariate version as there is no n-dimensional intermediate value theorem. The closest multivariate version is probably Nelder–Meade, a derivative-free or zeroth order optimization algorithm, but it is incredibly expensive in the worst case.

Regula falsi: The fact that method of bisection requires only the sign of the function value is of course an advantage — if we can find the sign much faster than the value. But if that's not the case, then it would be a waste to not use the value of the function. This can be remedied by using

$$x_k = \frac{a_k g(b_k) - b_k g(a_k)}{g(b_k) - g(a_k)} \tag{2.5}$$

in place of the mid-point $x_k = (a_k + b_k)/2$ in method of bisection as in the figure below.

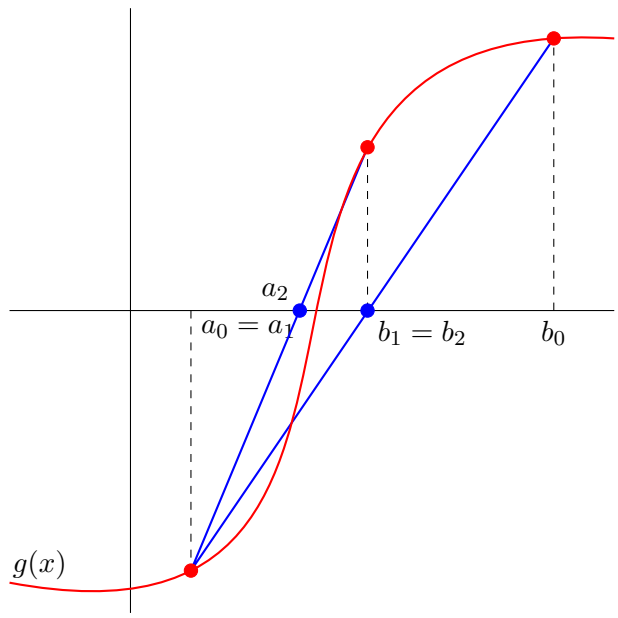


Figure 8: Pictorial view of the method of *regula falsi*.

The resulting method is called regula falsi (meaning 'false position'), a method that dates back to the Babylons and Egyptians and is found in the *Nine Chapters of Arithmetic*, an well-known ancient Chinese book on mathematics. The idea is that instead of taking the mid-point in an interval $[a_k, b_k]$, you take x_k to be the x-intercept of the line through $(a_k, g(a_k))$ and $(b_k, g(b_k))$, i.e.,

$$y - g(b_k) = \left[\frac{g(b_k) - g(a_k)}{b_k - a_k}\right](x - b_k).$$

Plugging in $x = x_k$ and $y = 0$ gives you (2.5). Why use this instead of secant method? Note that the divergent scenario in Figure 6 would not happen here — regula falsi guarantees that our iterates lie in a sequence of of nested intervals (2.4) as in bisection method.

Dekker's method: Regula falsi is not the only way to combine bisection and secant methods. Another possibility is suggested by Dekker's. While Regula Falsi uses two points, Dekker's method uses three: either a_k, a_{k-1}, b_k or a_k, b_{k-1}, b_k. While Regula Falsi uses the value of g to construct x_k, it relies only on the sign of g in its `if-else` decision; Dekker's method will use the value of g in its decision. If $|g(b_k)| \leq |g(a_k)|$, i.e., b_k is a better solution than a_k as $g(b_k)$ is closer to 0 than $g(a_k)$, then Dekker's method applies secant method with b_k and b_{k-1}. On the other hand, if $|g(a_k)| \leq |g(b_k)|$, then it applies secant method with a_k and a_{k-1}. See Algorithm 2.

Algorithm 2 Dekker's method

Require: $g(a_0)g(b_0) < 0$, $\varepsilon > 0$

procedure DEKKER(a_0, b_0, ε)
  $k \leftarrow 0$
  while $|b_k - a_k| > \varepsilon$ **do**
    if $|g(b_k)| \leq |g(a_k)|$ **then**
      if $g(b_k) \neq g(b_{k-1})$ **then**
        $x_k \leftarrow \dfrac{b_{k-1}g(b_k) - b_k f(b_{k-1})}{g(b_k) - g(b_{k-1})}$
      else
        $x_k \leftarrow \dfrac{a_k + b_k}{2}$
      end if
    else
      if $g(a_k) \neq g(a_{k-1})$ **then**
        $x_k \leftarrow \dfrac{a_{k-1}g(a_k) - a_k f(a_{k-1})}{g(a_k) - g(a_{k-1})}$
      else
        $x_k \leftarrow \dfrac{a_k + b_k}{2}$
      end if
    end if
    $k \leftarrow k + 1$
  end while
  return $x_* := x_k$
end procedure

Brent's method: This is the fastest known root finding algorithm even though it's been around for 50 years. It is a further improvement of Dekker's method in the following way: Note that even though Dekker's method uses three points in every step, the x_k involves only two. Brent proposed to use all three using a quadratic interplation just like in Müller's method. For example, if the three points are a_k, b_{k-1}, b_k and the values of $g(a_k)$, $g(b_{k-1})$, $g(b_k)$ are all distinct, then Brent's method would compute x_k as

$$x_k = \frac{a_k g(b_{k-1})g(b_k)}{(g(a_k) - g(b_{k-1}))(g(a_k) - g(b_k))} + \frac{b_{k-1}g(a_k)g(b_k)}{(g(b_{k-1}) - g(a_k))(g(b_{k-1}) - g(b_k))} + \frac{b_k g(a_k)g(b_{k-1})}{(g(b_k) - g(a_k))(g(b_k) - g(b_{k-1}))}$$

▶ Applications

Most applications of optimization are multivariate because most real-life problems depend on multiple factors. However that doesn't mean that there are no direct applications of the things that we have learned so far.

Example 2.6 (Group testing)**.** The principle underlying the method of bisection applies much more generally. Say we need to test $1024 = 2^{10}$ subjects and one of them is infected. If

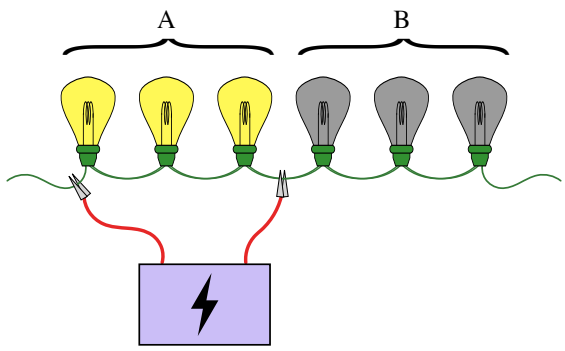


Figure 9: Method of bisection to identify a faulty light bulb.

we test each of them individually, we need 1024 tests; but if we divide them into two groups of 512, mix the tissue samples in each group and test each group, then repeat on the group with positive mixture, we just need 10 tests. This is the principle that underlies so-called *group testing*. □

Example 2.7 (Optimal speed of container ships)**.** Sometimes the way to arrive at a univariate optimization is to realize that despite the apparently numerous number of factors involved, only one matters to solve the problem — everything else can be kept constant. Here is a real-life example from maritime commerce. We can show that the daily gross profit f of a container ship is given by

$$f(c, d, p, r, s, w) = \frac{rw}{d/(24s)} - c - ps^3$$

where the variables are

$$\begin{aligned} c &= \text{running costs per day,} \\ d &= \text{distance in nautical miles,} \\ p &= \text{price of fuel in dollars per ton,} \\ r &= \text{freight rate per ton of cargo,} \\ s &= \text{speed in knots,} \\ w &= \text{weight of cargo in tons.} \end{aligned}$$

Note that speed is multiplied by 24 because a knot is a nautical mile per *hour* and cost is measured in *day*. The s^3 term comes from the famous *Cube Law* that says when we have anything that turns — fan in your laptop, crankshaft in a car, propeller in a boat, turbine in a

plane — the power drawn is proportional to the cube of the rotational speed. For a container ship, the Cube Law translates roughly to mean that fuel consumption is proportional to the cube of the speed, which accounts for the ps^3 term. The functional form of f is essentially profit minus running cost minus fuel cost. Important thing to observe is that even though this is a multivariate problem, the only variable that can be controlled and readily changed by the ship's operator is the speed. So from the operator's perspective, this is a univariate maximization problem with objective

$$f(s) = -ps^3 + bs - c,$$

where p, $b := 24rw/d$, c are regarded as constants and $s > 0$. Since $f'(s) = -3ps^2 + b = 0$ has the one positive solution $s_* = \sqrt{b/3p}$ at which $f''(s_*) = -6ps_* < 0$, we conclude that the optimal speed is given by

$$s_* = \sqrt{\frac{8rw}{pd}}. \qquad \square$$

We now move on to the next topic:

$$\begin{tikzcd}
\text{univariate} \arrow[r] & \boxed{\text{multivariate unconstrained}} \arrow[d] & \\
& \text{multivariate equality constrained} \arrow[d] & \\
\text{convex} & \text{multivariate inequality constrained} \arrow[l] \arrow[r] \arrow[d] & \text{nonsmooth} \\
& \begin{array}{c}\text{multiobjective, combinatorial,}\\ \text{dynamic, stochastic, integer, etc}\end{array} &
\end{tikzcd}$$

▶ Linear algebra refresher

Before studying multivariate optimization, we need some linear algebra and multivariable calculus. Fortunately we just need a tiny bit of linear algebra. The only thing I will assume is that you know how to multiply two matrices.

First we use the following notations for vectors and matrices:

$$\mathbb{R}^n := \left\{ \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} : x_1, \dots, x_n \in \mathbb{R} \right\},$$

$$\mathbb{R}^{m\times n} = \left\{ \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} : x_{ij} \in \mathbb{R},\ 1 \le i \le m, 1 \le j \le n \right\}.$$

Note that $\mathbb{R}^n = \mathbb{R}^{n\times 1}$ by our convention, i.e., every vector in $\mathbb{R}^n$ is a column vector.

The next thing we need to know are *norms*. The most common norms that we will use in this class, are:

(i) the Euclidean norm on $\mathbb{R}^n$ given by

$$\|\mathbf{x}\| = \left[\sum_{i=1}^{n} x_i^2\right]^{1/2}$$

(ii) the Frobenius norm on $\mathbb{R}^{m\times n}$ given by

$$\|X\| = \left[\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^2\right]^{1/2}.$$

Note that if m or n is 1, then the matrix is a row or column, respectively, and in which case the two norms agree — so there is no ambiguity in using the same notation $\|\cdot\|$ for both. It is easy to check that these norms satisfy the following properties:

- $\|X\| \geq 0$ for all $X \in \mathbb{R}^{m\times n}$;
- $\|X\| = 0$ iff $X = 0$;
- $\|\lambda X\| = |\lambda|\|X\|$ for all $\lambda \in \mathbb{R}$ and $X \in \mathbb{R}^{m\times n}$;
- $\|X + Y\| \leq \|X\| + \|Y\|$ for all $X, Y \in \mathbb{R}^{m\times n}$.

The last one is the famous *triangle inequality*. Although these are stated for the Frobenius norm on $\mathbb{R}^{m\times n}$, they hold true for the Euclidean norm too since that would just be the special case $n = 1$. The Frobenius norm satisfy an additional *submultiplicativity inequality* with respect to matrix product:

$$\|XY\| \leq \|X\|\|Y\|$$

for any $X \in \mathbb{R}^{m\times n}$ and $Y \in \mathbb{R}^{n\times p}$.

There are also occasions when we will need to use the p-norm on $\mathbb{R}^n$ given by

$$\|\mathbf{x}\|_p = \left[\sum_{i=1}^{n} |x_i|^p\right]^{1/p}$$

for $p \in [1, \infty)$. Obviously the $p = 2$ case is just the Euclidean norm above, i.e., $\|\mathbf{x}\|_2 = \|\mathbf{x}\|$. There is also the special case of $p = \infty$, defined by

$$\|\mathbf{x}\|_\infty = \max(|x_1|, \dots, |x_n|),$$

so called because

$$\lim_{p\to\infty} \|\mathbf{x}\|_p = \|\mathbf{x}\|_\infty.$$

The special case $p = 1$ is particularly important and deserves special mention:

$$\|\mathbf{x}\|_\infty = |x_1| + \cdots + |x_n|.$$

Indeed the $p = 1, 2, \infty$ cases will all play important roles when we discuss regression and classification in Lecture 5. There is also a simple variant called a *weighted p-norm* given by

$$\|\mathbf{x}\|_{\boldsymbol{\alpha},p} = \begin{cases} \left[\sum_{i=1}^{n} \alpha_i |x_i|^p\right]^{1/p} & \text{if } 1 \le p < \infty, \\ \max(\alpha_1|x_1|, \dots, \alpha_n|x_n|) & \text{if } p = \infty, \end{cases} \tag{2.8}$$

where $\alpha_1, \dots, \alpha_n > 0$ are positive *weights* and $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)$ a weight vector.

When $p < 1$, the formula in (2.8) do not define norms, e.g., triangular inequality fails, but they are nonetheless important. We will encounter them in the form of the CES production function when we discuss mathematical economics in Lecture 7.

Another thing we will need are *inner products*. We will only be interested in the two inner products:

(i) the Euclidean inner product between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$:

$$\mathbf{x}^{\mathsf{T}}\mathbf{y} = \sum_{i=1}^{n} x_i y_i;$$

(ii) the Frobenius inner product between two matrices $X, Y \in \mathbb{R}^{m\times n}$:

$$\operatorname{tr}(X^{\mathsf{T}}Y) = \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} y_{ij}.$$

As you can imagine, the Euclidean and Frobenius norms are intimately related to these inner products:

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^{\mathsf{T}}\mathbf{x}}, \qquad \|X\| = \sqrt{\operatorname{tr}(X^{\mathsf{T}}X)}$$

and they satisfy the *Cauchy–Schwarz inequality*:

$$|\mathbf{x}^{\mathsf{T}}\mathbf{y}| \le \|\mathbf{x}\|\|\mathbf{y}\|, \qquad |\operatorname{tr}(X^{\mathsf{T}}Y)| \le \|X\|\|Y\|.$$

We will only need three inequalities in this course: (i) triangle inequality, (ii) Cauchy–Schwartz inequality, (iii) submultiplicativity inequality.

Exercise 2.9. Prove these three inequalities for $\mathbb{R}^{m\times n}$:

$$\|X + Y\| \le \|X\| + \|Y\|, \qquad |\operatorname{tr}(X^{\mathsf{T}}Y)| \le \|X\|\|Y\|, \qquad \|XY\| \le \|X\|\|Y\|.$$

Note that you get the corresponding inequalities on $\mathbb{R}^m$ for free by setting $n = 1$.

Other linear algebraic notions that we need (e.g., positive definite matrices) we will introduce as we go along.

Lecture 3 (Tuesday, October 10)

▶ Convergence and continuity

We review convergence of sequences and continuity of functions.

Let $(\mathbf{x}_k)_{k=1}^{\infty}$ be a sequence of vectors in $\mathbb{R}^n$, i.e.,

$$\mathbf{x}_k = \begin{bmatrix} x_1^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} \in \mathbb{R}^n$$

for all $k \in \mathbb{N}$.

Definition 3.1. We say $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_*$ if

$$\lim_{k\to\infty} \|\mathbf{x}_k - \mathbf{x}_*\|_2 = 0.$$

That is, the norms of the errors is a convergent sequence in $\mathbb{R}$.

Theorem 3.2 (Entry-wise convergence of vectors)**.** *Let $(\mathbf{x}_k)_{k=1}^{\infty}$ be a sequence of vectors in $\mathbb{R}^n$. Then*

$$\mathbf{x}_k = \begin{bmatrix} x_1^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} \to \begin{bmatrix} x_1^{*} \\ \vdots \\ x_n^{*} \end{bmatrix} = \mathbf{x}_*$$

iff $x_i^{(k)} \to x_i^*$ for all $i = 1, \ldots, n$. That is,

$$\lim_{k\to\infty} \|\mathbf{x}_k - \mathbf{x}_*\| = 0 \qquad \Longleftrightarrow \qquad \lim_{k\to\infty} x_i^{(k)} = x_i^*$$

for $i = 1, \ldots, n$.

The formal, rigorous definition of continuity is given in the definition below but basically what you need to remember is that a function is continuous if

$$\lim_{k\to\infty} F(\mathbf{x}_k) = F\Big(\lim_{k\to\infty} \mathbf{x}_k\Big).$$

Definition 3.3 (Continuity)**.** A function $F : \mathbb{R}^n \to \mathbb{R}^m$ is *continuous* at $\mathbf{x} \in \mathbb{R}^n$ if

$$\lim_{\mathbf{y}\to\mathbf{x}} F(\mathbf{y}) = F(\mathbf{x}),$$

i.e., for all $\varepsilon > 0$ there is some $\delta > 0$ such that

$$\|F(\mathbf{y}) - F(\mathbf{x})\| < \varepsilon$$

whenever $\|\mathbf{y} - \mathbf{x}\| < \delta$. Alternatively, F is continuous at $\mathbf{x} \in \mathbb{R}^n$ if and only if

$$\lim_{k\to\infty} F(\mathbf{x}_k) = F(\mathbf{x})$$

whenever $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}$. We say F is continuous on an open subset $\Omega \subseteq \mathbb{R}^n$ if and only if F is continuous at $\mathbf{x}$ for all $\mathbf{x} \in \Omega$. We say F is Lipschitz on Ω if there is a Lipschitz constant $\gamma > 0$ such that

$$\|F(\mathbf{y}) - F(\mathbf{x})\| \le \gamma \|\mathbf{y} - \mathbf{x}\|$$

for all $\mathbf{x}, \mathbf{y} \in \Omega$.

Here are some useful results regarding continuity.

Proposition 3.4. *We have the following.*

(i) *Norm is a continuous function $\mathbb{R}^n \to \mathbb{R}$, i.e.,*

$$\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_* \qquad \Longrightarrow \qquad \lim_{k\to\infty} \|\mathbf{x}_k\| = \|\mathbf{x}_*\|.$$

(ii) *Matrix product is a contiuous function $\mathbb{R}^n \to \mathbb{R}^m$, i.e.,*

$$\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_* \qquad \Longrightarrow \qquad \lim_{k\to\infty} A\mathbf{x}_k = A\mathbf{x}_*$$

for any $A \in \mathbb{R}^{m\times n}$.

(iii) *Inner product is a continuous function on $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, i.e.,*

$$\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_* \quad \text{and} \quad \lim_{k\to\infty} \mathbf{y}_k = \mathbf{y}_* \qquad \Longrightarrow \qquad \lim_{k\to\infty} \mathbf{x}_k^\mathsf{T}\mathbf{y}_k = \mathbf{x}_*^\mathsf{T}\mathbf{y}_*.$$

(iv) *Linear combination is a continous function on $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$, i.e.,*

$$\lim_{k\to\infty} \alpha\mathbf{x}_k + \beta\mathbf{y}_k = \alpha \lim_{k\to\infty} \mathbf{x}_k + \beta \lim_{k\to\infty} \mathbf{y}_k$$

for any $\alpha, \beta \in \mathbb{R}$.

▶ Multivariate derivatives

We write upper case F for a vector-valued function $F : \mathbb{R}^n \to \mathbb{R}^m$ and lower case f for a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ in the following.

The simplest type of derivative for multivariate functions is the *partial derivative*: Consider $F\colon \mathbb{R}^n \to \mathbb{R}^m$, i.e.,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \mapsto \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix}.$$

Then the ith partial derivative of F at $\mathbf{x}$, if it exists, is given by

$$\frac{\partial F}{\partial x_i}(\mathbf{x}) = \lim_{h\to 0} \frac{F(\mathbf{x} + h\mathbf{e}_i) - F(\mathbf{x})}{h}.$$

Partial derivatives are special cases of *directional derivative*: Let $F\colon \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{0} \ne \mathbf{v} \in \mathbb{R}^n$. The directional derivative along $\mathbf{v}$ of F at $\mathbf{x} \in \mathbb{R}^n$, if it exists, is

$$\partial_{\mathbf{v}} F(\mathbf{x}) = \lim_{h\to 0} \frac{F(\mathbf{x} + h\mathbf{v}) - F(\mathbf{x})}{h}.$$

In particular,

$$\partial_{\mathbf{e}_i} F(\mathbf{x}) = \frac{\partial F}{\partial x_i}(\mathbf{x}).$$

It is convenient to collect all partial derivatives together and store them in a vector or a matrix.

Gradient of a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ is a vector-valued function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ given by

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^n$$

Jacobian of a vector-valued function $F : \mathbb{R}^n \to \mathbb{R}^m$ is a matrix-valued function $\mathrm{J}F : \mathbb{R}^n \to \mathbb{R}^{m\times n}$ given by

$$\mathrm{J}F(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_n}(\mathbf{x}) \end{bmatrix} = \left[\frac{\partial f_i}{\partial x_j}(\mathbf{x})\right]_{i,j=1}^{m,n} \in \mathbb{R}^{m\times n}.$$

Hessian of a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ is a matrix-valued function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ given by

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(\mathbf{x}) \end{bmatrix} = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x})\right]_{i,j=1}^{n} \in \mathbb{R}^{n\times n}.$$

Note that this is a symmetric matrix, since

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$$

for C^2 functions.

If $m = 1$, i.e., $f : \mathbb{R}^n \to \mathbb{R}$ is scalar-valued, the Jacobian is related to the gradient in that

$$\nabla f(\mathbf{x}) = \big(\mathrm{J}f(\mathbf{x})\big)^\mathsf{T}$$

and it is related to the Hessian in that

$$\nabla^2 f(\mathbf{x}) = \mathrm{J}\nabla f(\mathbf{x}).$$

The directional derivative is related to the gradient in that

$$\partial_{\mathbf{v}} f(\mathbf{x}) = \nabla f(\mathbf{x})^\mathsf{T}\mathbf{v}.$$

In particular, note that

$$\partial_{\mathbf{e}_i} f(\mathbf{x}) = \nabla f(\mathbf{x})^\mathsf{T} \mathbf{e}_i = \frac{\partial f}{\partial x_i}(\mathbf{x}).$$

These objects are incredibly important in optimization. Getting ahead of ourselves, a necessary condition for a local minimum at $\mathbf{x}$ is $\nabla f(\mathbf{x}) = \mathbf{0}$, and a sufficient condition for a local minimum is $\nabla f(\mathbf{x}) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}) \succ 0$, the shorthand '$\succ 0$' means 'is positive definite.'

▶ Product, quotient, and chain rules

Just as in univariate calculus, complicated functions are built out of simpler functions and it helps to have some rules that we can may use to calculate derivatives.

Proposition 3.5 (Linear combination rule)**.** *Let $F, G \colon \mathbb{R}^n \to \mathbb{R}^m$ be differentiable at $\mathbf{x} \in \mathbb{R}^n$. Then $\alpha F + \beta G \colon \mathbb{R}^n \to \mathbb{R}^m$ is also differentiable at $\mathbf{x}$, and*

$$\mathrm{J}(\alpha F + \beta G)(\mathbf{x}) = \alpha \mathrm{J}F(\mathbf{x}) + \beta \mathrm{J}G(\mathbf{x})$$

for any $\alpha, \beta \in \mathbb{R}$.

The most important result is the chain rule.

Proposition 3.6 (Chain rule)**.** *Let $G \colon \mathbb{R}^n \to \mathbb{R}^m$ be differentiable at $\mathbf{x} \in \mathbb{R}^n$ and $F \colon \mathbb{R}^m \to \mathbb{R}^p$ be differentiable at $G(\mathbf{x}) \in \mathbb{R}^m$. Then $H \colon \mathbb{R}^n \to \mathbb{R}^p$ be defined by $H(\mathbf{x}) = F(G(\mathbf{x}))$ is differentiable at $\mathbf{x}$, and*

$$\mathrm{J}H(\mathbf{x}) = \mathrm{J}F(G(\mathbf{x})) \cdot \mathrm{J}G(\mathbf{x}),$$

where $\cdot$ is matrix multiplication.

Note that the function H is called a *composition* of F and G and usually denoted by $H = F \circ G$ and sometimes depected as

$$\mathbb{R}^n \xrightarrow{G} \mathbb{R}^m \xrightarrow{F} \mathbb{R}^p.$$

Here are some important special cases.

Corollary 3.7 (Special chain rules)**.**

(i) *Composition with scalar-valued functions: Let $g \colon \mathbb{R}^n \to \mathbb{R}$ be differentiable at $\mathbf{x} \in \mathbb{R}^n$ and $f \colon \mathbb{R} \to \mathbb{R}$ be differentiable at $g(\mathbf{x}) \in \mathbb{R}$. Then $h \colon \mathbb{R}^n \to \mathbb{R}$ defined by $h(\mathbf{x}) = f(g(\mathbf{x}))$ is differentiable at $\mathbf{x}$, and*

$$\nabla h(\mathbf{x}) = f'(g(\mathbf{x})) \nabla g(\mathbf{x}).$$

(ii) *Composition with affine functions: $G \colon \mathbb{R}^n \to \mathbb{R}^m$ and $H \colon \mathbb{R}^p \to \mathbb{R}^m$ defined by $H(\mathbf{x}) = G(A\mathbf{x} + \mathbf{b})$ where $A \in \mathbb{R}^{n \times p}$ and $\mathbf{b} \in \mathbb{R}^n$*

$$\mathrm{J}H(\mathbf{x}) = \mathrm{J}G(A\mathbf{x} + \mathbf{b})A.$$

The special case where $m = 1$ gives:

$$\nabla h(\mathbf{x}) = A^\mathsf{T} \nabla g(A\mathbf{x} + \mathbf{b}).$$

You may occasionally find the following for gradients useful.

Corollary 3.8 (Product and quotient rules)**.** *Let $f, g\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable. Then*

$$\nabla(fg) = f\nabla g + g\nabla f,$$
$$\nabla\left(\frac{f}{g}\right) = \frac{g\nabla f - f\nabla g}{g^2}.$$

If we have a repeated composition of functions like $F = F_k \circ F_{k-1} \circ \cdots \circ F_2 \circ F_1$,

$$\mathbb{R}^{n_0} \xrightarrow{F_1} \mathbb{R}^{n_1} \xrightarrow{F_2} \mathbb{R}^{n_2} \xrightarrow{F_3} \mathbb{R}^{n_3} \xrightarrow{F_4} \cdots \xrightarrow{F_{k-1}} \mathbb{R}^{n_{k-1}} \xrightarrow{F_k} \mathbb{R}^{n_k},$$

then applying Proposition 3.6 multiple times gives us

$$\underbrace{\mathrm{J}F(\mathbf{x})}_{n_k \times n_0} = \underbrace{\mathrm{J}F_k\big(F_{k-1} \circ \cdots \circ F_1(\mathbf{x})\big)}_{n_k \times n_{k-1}} \cdot \underbrace{\mathrm{J}F_{k-1}\big(F_{k-2} \circ \cdots \circ F_1(\mathbf{x})\big)}_{n_{k-1} \times n_{k-2}} \cdot \cdots \cdot \underbrace{\mathrm{J}F_2\big(F_1(\mathbf{x})\big)}_{n_2 \times n_1} \cdot \underbrace{\mathrm{J}F_1(\mathbf{x})}_{n_1 \times n_0}$$

where again $\circ$ denotes function composition and $\cdot$ denotes matrix product. There are two obvious ways to compute this product of k matrices:

- *backward propagation*: compute from left to right;
- *forward propagation*: compute from right to left.

Clearly we should do backward propagation when $n_k \ll n_0$ and forward propagation when $n_0 \ll n_k$ since we will be working with smaller matrices. An extreme example is when $n_k = 1$, which is particularly relevant because in optimization we work with real-valued functions $f : \mathbb{R}^n \to \mathbb{R}$ so $n_0 = n$ and $n_k = 1$. Consider the case when we have

$$f = f_k \circ F_{k-1} \circ \cdots \circ F_2 \circ F_1$$

where the last function $f_k : \mathbb{R}^{n_{k-1}} \to \mathbb{R}$ is a real-valued function. This particularly relevant in the situation of *deep neural networks* where $F_1, \dots, F_{k-1}$ alternates between affine functions and activation functions and the last f_k is a *loss function*. In this case it is standard to express in terms of gradient so

$$\underbrace{\nabla f(\mathbf{x})}_{n} = \underbrace{\mathrm{J}F_1(\mathbf{x})^{\mathsf{T}}}_{n \times n_1} \underbrace{\mathrm{J}F_2\big(F_1(\mathbf{x})\big)^{\mathsf{T}}}_{n_1 \times n_2} \cdot \cdots \cdot \underbrace{\mathrm{J}F_{k-1}\big(F_{k-2} \circ \cdots \circ F_1(\mathbf{x})\big)^{\mathsf{T}}}_{n_{k-2} \times n_{k-1}} \cdot \underbrace{\nabla f_k\big(F_{k-1} \circ \cdots \circ F_1(\mathbf{x})\big)}_{n_{k-1}}.$$

Hence if we backpropagate, we start from the right most term, which is a vector of dimension n_{k-1}, then in every step we multiply the vector by a matrix, so we end up with $k-1$ matrix-vector products. On the other hand, if we start from the left most term, which is a matrix of dimension $n \times n_1$, then in every step we have to do a matrix-matrix product until the last step when we do a matrix-vector product. Since matrix-matrix products are much more expensive than matrix-vector products, backpropagation is clearly a better idea when we have to evaluate a gradient of this form.

The importance of chain rule and backpropagation in optimization goes far beyond training neural networks. In general, we would like to be able evaluate gradients and Hessians without actually using symbolic computing to determine their expressions explicitly. The way to do

this is find clever ways to express a function as linear combinations, products, quotients, and compositions of simpler functions and apply linear combination, product, quotient, and chain rules. This is called *automatic differentiation.* Take a univariate toy example $f(x) = x\sin(x^2)$ and we want the values of $y = f(3)$ and $y' = f'(3)$ without first finding $f'(x)$ symbolically, we can do a forward propagation:

f	f'
$x = 3$	$x' = 1$
$y_1 = x^2 = 9$	$y_1' = 2xx' = 6$
$y_2 = \sin(y_1) = 0.4121$	$y_2' = \cos(y_1)y_1' = -5.46668$
$y = xy_2 = 1.2363$	$y' = x'y_2 + xy_2' = -15.9883$

More generally we can do this with any multivariate function and for higher derivatives as well. In this case we can show that backward propagation is much more efficient than forward propagation. But the details are too involved to go into.

▶ Calculating gradients and Jacobians

Let's start with some simple examples that can be computed directly, or with the linear combination rule. Later we will look at more complicated examples requiring chain rule.

Example 3.9. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix, $\mathbf{b} \in \mathbb{R}^n$, $c \in \mathbb{R}$, and $f : \mathbb{R}^n \to \mathbb{R}$ be the quadratic function

$$f(\mathbf{x}) = \mathbf{x}^\mathsf{T} A\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x} + c = \sum_{i=1}^n \sum_{j=1}^n a_{ij}x_ix_j + \sum_{i=1}^n b_ix + c.$$

Then it follows from

$$\frac{\partial}{\partial x_i}\mathbf{x}^\mathsf{T} A\mathbf{x} = 2\sum_{j=1}^n a_{ij}x_j, \qquad \frac{\partial}{\partial x_i}\mathbf{b}^\mathsf{T}\mathbf{x} = b_i,$$

and the linear combination rule that the Jacobian and gradient of f at $\mathbf{x}$ are

$$\begin{aligned} \mathrm{J}f(\mathbf{x}) &= 2\mathbf{x}^\mathsf{T} A^\mathsf{T} + \mathbf{b}^\mathsf{T}, \\ \nabla f(\mathbf{x}) &= 2A\mathbf{x} + \mathbf{b}. \end{aligned}$$

Note that this is consistent with the univariate case $(ax^2 + bx + c)' = 2ax + b$.

Question 3.10. *What is the gradient of* $\mathbf{x}^\mathsf{T} A\mathbf{x}$ *when* $A \in \mathbb{R}^{n\times n}$ *is not symmetric?*

Every matrix can be written as the sum of a symmetric matrix and a skew symmetric matrix:

$$A = \underbrace{\frac{A + A^\mathsf{T}}{2}}_{\text{symmetric}} + \underbrace{\frac{A - A^\mathsf{T}}{2}}_{\text{skew-symmetric}} = B + C,$$

i.e., $B^\mathsf{T} = B$ and $C^\mathsf{T} = -C$. Note that $\mathbf{x}^\mathsf{T} C\mathbf{x}$ is a 1×1 matrix, so in particular, $(\mathbf{x}^\mathsf{T} C\mathbf{x})^\mathsf{T} = \mathbf{x}^\mathsf{T} C\mathbf{x}$. It follows that

$$\mathbf{x}^\mathsf{T} C\mathbf{x} = (\mathbf{x}^\mathsf{T} C\mathbf{x})^\mathsf{T} = \mathbf{x}^\mathsf{T} C^\mathsf{T}\mathbf{x} = -\mathbf{x}^\mathsf{T} C\mathbf{x},$$

implying $\mathbf{x}^\mathsf{T} C\mathbf{x} = 0$ for all $\mathbf{x} \in \mathbb{R}^n$. Then

$$\mathbf{x}^\mathsf{T} A\mathbf{x} = \mathbf{x}^\mathsf{T} B\mathbf{x} + \mathbf{x}^\mathsf{T} C\mathbf{x} = \mathbf{x}^\mathsf{T} B\mathbf{x} + 0 = \mathbf{x}^\mathsf{T} \left(\frac{A + A^\mathsf{T}}{2}\right) \mathbf{x}.$$

In summary, we always have

$$\mathbf{x}^\mathsf{T} A\mathbf{x} = \mathbf{x}^\mathsf{T} \left(\frac{A + A^\mathsf{T}}{2}\right) \mathbf{x} \tag{3.11}$$

for all $\mathbf{x} \in \mathbb{R}^n$.

So if A is not symmetric in Example 3.9, we just need to rewrite $\mathbf{x}^\mathsf{T} A\mathbf{x}$ in symmetric form $\mathbf{x}^\mathsf{T}(A + A^\mathsf{T})\mathbf{x}/2$ using (3.11) and we immediately get

$$\nabla \mathbf{x}^\mathsf{T} A\mathbf{x} = (A + A^\mathsf{T})\mathbf{x}.$$

Note that if A is symmetric, we get back $\nabla f(\mathbf{x}) = 2A\mathbf{x}$.

Exercise 3.12 (Norm)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x} \mapsto \|\mathbf{x}\|^2$. Note that $f(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{x}$ and so this is just Example 3.9 with $A = I$, $\mathbf{b} = \mathbf{0}$, $c = 0$. In particular

$$\nabla \|\mathbf{x}\|^2 = 2\mathbf{x}.$$

What about the norm function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x} \mapsto \|\mathbf{x}\|$? We just use chain rule with the square root function $g : \mathbb{R} \to \mathbb{R}$, $x \mapsto \sqrt{x}$ and $\|\cdot\| = g \circ f$. You will be asked to do this in Homework 2.

Example 3.13 (Matrix-vector product)**.** $F : \mathbb{R}^n \to \mathbb{R}^m$ defined by $F(\mathbf{x}) = A\mathbf{x}$ for some fixed $A \in \mathbb{R}^{m\times n}$. Note that this is a vector-valued map and gradient does not make sense for F but its Jacobian is

$$\mathrm{J}F(\mathbf{x}) = A.$$

We present some examples to illustrate chain rule.

Example 3.14 (Geometric programming)**.** Let $f\colon \mathbb{R}^m \to \mathbb{R}$ be the log-sum-exp function defined by

$$f(\mathbf{y}) = \log(e^{y_1} + \cdots + e^{y_m}).$$

Let $\mathbf{a}_1, \dots, \mathbf{a}_m \in \mathbb{R}^n$ and $b_1, \dots, b_m \in \mathbb{R}$. We set

$$A = \begin{bmatrix} \mathbf{a}_1^\mathsf{T} \\ \vdots \\ \mathbf{a}_m^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{m\times n}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \in \mathbb{R}^m.$$

Then the function $g\colon \mathbb{R}^n \to \mathbb{R}$,

$$g(\mathbf{x}) = \log\Big(\sum\nolimits_{i=1}^m \exp(\mathbf{a}_i^\mathsf{T}\mathbf{x} + b_i)\Big)$$

may be written as a composition of functions: $g(\mathbf{x}) = f(A\mathbf{x}+\mathbf{b})$. The gradient of f is simple enough that the dumb formula can be applied directly (very rare in this course):

$$\nabla f(\mathbf{y}) = \frac{1}{\sum_{i=1}^m e^{y_i}} \begin{bmatrix} e^{y_1} \\ \vdots \\ e^{y_m} \end{bmatrix}.$$

By chain rule,

$$\nabla g(\mathbf{x}) = \frac{1}{\sum_{i=1}^m e^{\mathbf{a}_i^\mathsf{T}\mathbf{x}+b_i}} A^\mathsf{T} \begin{bmatrix} e^{\mathbf{a}_1^\mathsf{T}\mathbf{x}+b_1} \\ \vdots \\ e^{\mathbf{a}_m^\mathsf{T}\mathbf{x}+b_m} \end{bmatrix}.$$

Optimizing such functions, that is,

$$\begin{array}{rll} \text{minimize} & g_0(\mathbf{x}) & \\ \text{subject to} & g_i(\mathbf{x}) = 0, & i = 1, \dots, p, \\ & g_i(\mathbf{x}) \le 0, & i = p+1, \dots, r, \end{array}$$

where $g_0, g_1, \dots, g_r$ are all functions of such form, is called *geometric programming* and is very important in circuit and chip designs. The log-sum-exp function f arises as a smooth approximation of the nondifferentiable max function as

$$\max(x_1, \dots, x_n) \le \operatorname{lse}(x_1, \dots, x_n) \le \max(x_1, \dots, x_n) + \log n.$$

You might wish to prove this as an exercise.

▶ Calculating Hessians

We now move on to the Hessian. For a real-valued function $f\colon \mathbb{R}^n \to \mathbb{R}$ tat is twice continuously differentiable, its gradient is a vector-valued function $\nabla f\colon \mathbb{R}^n \to \mathbb{R}^n$. The *Hessian* of f at $\mathbf{x}$ is then defined to be the Jacobian of ∇f at $\mathbf{x}$, i.e.,

$$\mathrm{J}(\nabla f)(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2}(\mathbf{x}) & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{x}) & \cdots & \dfrac{\partial^2 f}{\partial x_n^2}(\mathbf{x}) \end{bmatrix} = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x})\right]_{i,j=1}^n \in \mathbb{R}^{n\times n}.$$

and denoted

$$\nabla^2 f(\mathbf{x}) = \mathrm{J}(\nabla f)(\mathbf{x}).$$

In other words, the Hessian is a matrix-valued function $\nabla^2 f\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$.

Let's start with some simple examples.

Example 3.15. Let $f(x_1, x_2) = x_1 x_2$. Then

$$\nabla^2 f(x_1, x_2) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

This can be seen by writing

$$f(x_1, x_2) = \frac{1}{2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\mathsf{T} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Example 3.16. Let $f(x_1, x_2) = \dfrac{1}{x_1 x_2}$. Then

$$\nabla^2 f(x_1, x_2) = \frac{1}{x_1 x_2} \begin{bmatrix} 2/x_1^2 & 1/(x_1 x_2) \\ 1/(x_1 x_2) & 2/x_2^2. \end{bmatrix}$$

Example 3.17. Let $f(x_1, x_2) = \dfrac{x_1}{x_2}$. Then

$$\nabla^2 f(x_1, x_2) = \begin{bmatrix} 0 & -1/x_2^2 \\ -1/x_2^2 & 2x_1/x_2^3 \end{bmatrix}.$$

Example 3.18. Revisiting the quadratic function in Example 3.9,

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\intercal A\mathbf{x} + \mathbf{b}^\intercal \mathbf{x} + c,$$

we see that

$$\nabla^2 f(\mathbf{x}) = \begin{cases} \frac{1}{2}(A + A^\intercal) & \text{if } A \text{ is not necessarily symmetric,} \\ A & \text{if } A \text{ is symmetric.} \end{cases}$$

More complicated examples will require the chain rule. The chain rule for Hessian is used most often in the following forms.

Proposition 3.19 (Chain rules for Hessian)**.**

(i) *Composition with scalar-valued functions: Let* $f\colon \mathbb{R}^n \to \mathbb{R}$ *and* $g\colon \mathbb{R} \to \mathbb{R}$*. Then*

$$\underbrace{\nabla^2 (g \circ f)(\mathbf{x})}_{\text{matrix}} = \underbrace{g'(f(\mathbf{x}))}_{\text{scalar}} \underbrace{\nabla^2 f(\mathbf{x})}_{\text{matrix}} + \underbrace{g''(f(\mathbf{x}))}_{\text{scalar}} \underbrace{\nabla f(\mathbf{x}) \nabla f(\mathbf{x})^\intercal}_{\text{matrix}}$$

(ii) *Composition with affine maps: Let* $f\colon \mathbb{R}^m \to \mathbb{R}$ *and* $A \in \mathbb{R}^{m\times n}$ *and* $\mathbf{b} \in \mathbb{R}^m$*. Define* $g\colon \mathbb{R}^n \to \mathbb{R}$ *by* $g(\mathbf{x}) = f(A\mathbf{x} + \mathbf{b})$*. Then*

$$\underbrace{\nabla^2 g(\mathbf{x})}_{n\times n} = \underbrace{A^\intercal}_{n\times m} \underbrace{\nabla^2 f(A\mathbf{x} + \mathbf{b})}_{m\times m} \underbrace{A}_{m\times n}$$

Notation 3.20 (Vector products)**.** Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. The *inner product* of $\mathbf{a}$ and $\mathbf{b}$ is

$$\mathbf{a}^\intercal \mathbf{b} = a_1 b_1 + \cdots + a_n b_n \in \mathbb{R}.$$

The *outer product* is

$$\mathbf{a}\mathbf{b}^\intercal = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_n \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

A special case is $\mathbf{a}\mathbf{a}^\intercal = [a_i a_j]_{i,j=1}^n \in \mathbb{R}^{n\times n}$.

Example 3.21 (Linear Fractional Transformation)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be defined by

$$f(\mathbf{x}) = \frac{\|A\mathbf{x}+\mathbf{b}\|^2}{\mathbf{c}^\top\mathbf{x}+d}.$$

Define $g\colon \mathbb{R}^n \times (0,\infty) \to \mathbb{R}$ by

$$g(\mathbf{y},t) = \frac{\mathbf{y}^\top\mathbf{y}}{t} = \frac{\|\mathbf{y}\|^2}{t}.$$

Then

$$\nabla^2 g(\mathbf{y},t) = 2\begin{bmatrix} \frac{1}{t}I_n & -\frac{\mathbf{y}}{t^2} \\ \frac{-\mathbf{y}^\top}{t^2} & \frac{\mathbf{y}^\top\mathbf{y}}{t^3} \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}.$$

This representation is given as an *block matrix* or *partitioned matrix*:

$$\left[\begin{array}{c|c} \mathbf{M} & \mathbf{c} \\ \hline \mathbf{r} & s \end{array}\right], \qquad \begin{array}{rl} \mathbf{M} & \text{matrix} \\ \mathbf{c} & \text{column vector} \\ \mathbf{r} & \text{row vector} \\ s & \text{scalar} \end{array}$$

Note that matrix-vector multiply works as if the block 2×2 matrix is really a 2×2 matrix

$$\begin{bmatrix} M & \mathbf{c} \\ \mathbf{r}^\top & s \end{bmatrix}\begin{bmatrix} \mathbf{x} \\ t \end{bmatrix} = \begin{bmatrix} M\mathbf{x}+t\mathbf{c} \\ \mathbf{r}^\top\mathbf{x}+s \end{bmatrix}.$$

Now let $\mathbf{y} = A\mathbf{x}+\mathbf{b}$ and $t = \mathbf{c}^\top\mathbf{x}+d$ and so

$$\begin{bmatrix} \mathbf{y} \\ t \end{bmatrix} = \begin{bmatrix} A\mathbf{x}+\mathbf{b} \\ \mathbf{c}^\top\mathbf{x}+d \end{bmatrix} = \begin{bmatrix} A \\ \mathbf{c}^\top \end{bmatrix}\mathbf{x} + \begin{bmatrix} \mathbf{b} \\ d \end{bmatrix}.$$

Applying Proposition 3.19, we get

$$\nabla^2 f(\mathbf{x}) = \frac{2}{\mathbf{c}^\top\mathbf{x}+d}\begin{bmatrix} A \\ \mathbf{c}^\top \end{bmatrix}^\top \begin{bmatrix} I_n & -\frac{A\mathbf{x}+\mathbf{b}}{\mathbf{c}^\top\mathbf{x}+d} \\ \frac{-(A\mathbf{x}+\mathbf{b})^\top}{\mathbf{c}^\top\mathbf{x}+d} & \frac{\|A\mathbf{x}+\mathbf{b}\|^2}{(\mathbf{c}^\top\mathbf{x}+d)^2} \end{bmatrix}\begin{bmatrix} A \\ \mathbf{c}^\top \end{bmatrix}$$

Notation 3.22 (Diag operator)**.** Let $\mathbf{a} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$. We use the notation diag in two different ways:

$$\operatorname{diag}(\mathbf{a}) = \begin{bmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{n\times n}, \qquad \operatorname{diag}(A) = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^n$$

depending on whether the argument is a vector or a matrix. So diag takes a vector to a matrix and a matrix to a vector.

Example 3.23 (Log-sum-exp or LSE function)**.** Define $f\colon \mathbb{R}^n \to \mathbb{R}$ by

$$f(\mathbf{x}) = \log\left(\sum_{i=1}^{m} e^{\mathbf{a}_i^\mathsf{T}\mathbf{x}+b_i}\right),$$

where $\mathbf{a}_1, \dots, \mathbf{a}_m \in \mathbb{R}^n$ and $b_1, \dots, b_m \in \mathbb{R}$. Write

$$A = \begin{bmatrix} \mathbf{a}_1^\mathsf{T} \\ \vdots \\ \mathbf{a}_m^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{m\times n}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

so that

$$\begin{bmatrix} \mathbf{a}_1^\mathsf{T}\mathbf{x} + b_1 \\ \vdots \\ \mathbf{a}_m^\mathsf{T}\mathbf{x} + b_m \end{bmatrix} = A\mathbf{x} + \mathbf{b}.$$

Define $g\colon \mathbb{R}^n \to \mathbb{R}$ by

$$g(\mathbf{y}) = \operatorname{lse}(\mathbf{y}) := \log\left(\sum_{i=1}^{m} e^{y_i}\right).$$

Then $f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{b})$. The gradient and Hessian of g are given by

$$\nabla g(\mathbf{y}) = \frac{1}{\sum_{i=1}^m e^{y_i}} \nabla\left(\sum_{i=1}^{m} e^{y_i}\right) = \frac{1}{\sum_{i=1}^m e^{y_i}} \begin{bmatrix} e^{y_1} \\ \vdots \\ e^{y_m} \end{bmatrix}.$$

$$\nabla^2 g(\mathbf{y}) = \mathrm{J}(\nabla g)(\mathbf{y}) = \begin{bmatrix} \dfrac{\partial g}{\partial y_1} & & \\ & \ddots & \\ & & \dfrac{\partial g}{\partial y_n} \end{bmatrix} - \left[\frac{\partial g}{\partial y_i}\frac{\partial g}{\partial y_j}\right]_{i,j=1}^{m}.$$

More compactly,

$$\nabla^2 g(\mathbf{y}) = \underbrace{\operatorname{diag}(\nabla g(\mathbf{y}))}_{n\times n} - \underbrace{[\nabla g(\mathbf{y})]}_{n\times 1}\underbrace{[\nabla g(\mathbf{y})]^\mathsf{T}}_{1\times n}$$

By Proposition 3.19 (note that the roles of f and g have been reversed in the way we stated chain rule for Hessian),

$$\nabla^2 f(\mathbf{x}) = A^\mathsf{T}\nabla^2 g(A\mathbf{x}+\mathbf{b})A =$$

$$A^\mathsf{T}\left[\left(\frac{1}{\sum_{i=1}^m e^{\mathbf{a}_i^\mathsf{T}\mathbf{x}+b_i}}\right)\begin{bmatrix} e^{\mathbf{a}_1^\mathsf{T}\mathbf{x}+b_1} & & \\ & \ddots & \\ & & e^{\mathbf{a}_m^\mathsf{T}\mathbf{x}+b_m} \end{bmatrix} - \frac{1}{\left(\sum_{i=1}^m e^{\mathbf{a}_i^\mathsf{T}\mathbf{x}+b_i}\right)^2}\begin{bmatrix} e^{\mathbf{a}_1^\mathsf{T}\mathbf{x}+b_1} \\ \vdots \\ e^{\mathbf{a}_m^\mathsf{T}\mathbf{x}+b_m} \end{bmatrix}\begin{bmatrix} e^{\mathbf{a}_1^\mathsf{T}\mathbf{x}+b_1} \\ \vdots \\ e^{\mathbf{a}_m^\mathsf{T}\mathbf{x}+b_m} \end{bmatrix}^\mathsf{T}\right]A.$$

Lecture 4 (Tuesday, October 17)

▶ Linear and quadratic approximations

Gradients and Hessians are of paramount importance in optimization in multiple ways but primarily they are used to (i) construct algorithms, (ii) check optimality, (iii) check convexity. We will discuss these in the following and the next few lectures.

The most common optimization algorithms are based on linear and quadratic approximations of the objective function. For a C^1 function $f : \mathbb{R}^n \to \mathbb{R}$, its gradient can be defined by

$$\lim_{\mathbf{h}\to\mathbf{0}} \frac{|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^{\mathsf{T}}\mathbf{h}|}{\|\mathbf{h}\|} = 0 \tag{4.1}$$

which may be reinterpreted as a *first order* or *linear* approximation

$$f(\mathbf{x}+\mathbf{h}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^{\mathsf{T}}\mathbf{h}.$$

For a C^2 function $f : \mathbb{R}^n \to \mathbb{R}$, its gradient and Hessian can be simultaneously defined via

$$\lim_{\mathbf{h}\to 0} \frac{|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^{\mathsf{T}}\mathbf{h} - \frac{1}{2}\mathbf{h}^{\mathsf{T}}\nabla^2 f(X)\mathbf{h}|}{\|\mathbf{h}\|^2} = 0 \tag{4.2}$$

which may be reinterpreted as a *second order* or *quadratic* approximation

$$f(\mathbf{x}+\mathbf{h}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^{\mathsf{T}}\mathbf{h} + \tfrac{1}{2}\mathbf{h}^{\mathsf{T}}\nabla^2 f(\mathbf{x})\mathbf{h}.$$

For a C^1 *vector-valued* function $F : \mathbb{R}^n \to \mathbb{R}^m$, its Jacobian can be defined by

$$\lim_{\mathbf{h}\to\mathbf{0}} \frac{\|F(\mathbf{x}+\mathbf{h}) - F(\mathbf{x}) - \mathrm{J}f(\mathbf{x})\mathbf{h}\|}{\|\mathbf{h}\|} = 0 \tag{4.3}$$

which may be reinterpreted as a *first order* or *linear* approximation

$$F(\mathbf{x}+\mathbf{h}) \approx F(\mathbf{x}) + \mathrm{J}f(\mathbf{x})\mathbf{h}.$$

Note the product between $\mathrm{J}f(\mathbf{x})$ and $\mathbf{h}$ is a matrix-vector product.

The linear and quadratic approximations (4.1) and (4.2) will be the basis behind our optimization algorithms in the next lecture, whereas the linear approximation (4.3) will be the basis behind our algorithms for solving nonlinear equations (these in turn provide another type of algorithms for optimization) in the next lecture.

Taylor's theorem may be viewed as a more refined version of the linear and quadratic approximations, providing a precise estimate for the errors. One advantage Taylor's theorem provides is that instead of involving a vector $\mathbf{h}$ that goes to zero, it involves an arbitrary nonzero vector $\mathbf{p}$ that we may choose — this will be very useful in our algorithms.

Theorem 4.4 (Taylor's theorem multivariate functions)**.**

(a) *First-order approximation: If $f\colon \mathbb{R}^n \to \mathbb{R}$ is C^1 at $\mathbf{x} \in \mathbb{R}^n$, then for any nonzero $\mathbf{p} \in \mathbb{R}^n$,*

$$f(\mathbf{x}+\mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x}+\xi\mathbf{p})^{\mathsf{T}}\mathbf{p}$$

for some $\xi \in (0,1)$.

(b) *Second order approximation: If $f\colon \mathbb{R}^n \to \mathbb{R}$ is C^2 at $\mathbf{x} \in \mathbb{R}^n$, then for any nonzero $\mathbf{p} \in \mathbb{R}^n$,*

(i) *in terms of ∇f,*

$$\nabla f(\mathbf{x}+\mathbf{p}) = \nabla f(\mathbf{x}) + \int_0^1 \nabla^2 f(\mathbf{x}+t\mathbf{p})\mathbf{p}\,\mathrm{d}t;$$

(ii) *in terms of f,*

$$f(\mathbf{x}+\mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\nabla^2 f(\mathbf{x}+\xi\mathbf{p})\mathbf{p}$$

for some $\xi \in (0,1)$.

Remarks 4.5. (i) First order approximation is also called mean value theorem.

(ii) The remainder for ∇f cannot be written in a differential form, i.e., in general there is no $\xi \in (0,1)$ such that

$$\nabla f(\mathbf{x}+\mathbf{p}) = \nabla f(\mathbf{x}) + \nabla^2 f(\mathbf{x}+\xi\mathbf{p})\mathbf{p}.$$

(iii) The integral in 2(a) refers to the integration of a *vector-valued* univariate function $F\colon \mathbb{R} \to \mathbb{R}^n$ and is given by

$$F(t) = \begin{bmatrix} f_1(t) \\ \vdots \\ f_n(t) \end{bmatrix}$$

where f_i's are integrable on $[a,b]$, we define

$$\int_a^b F(t)\,\mathrm{d}t = \begin{bmatrix} \int_a^b f_1(t)\,\mathrm{d}t \\ \vdots \\ \int_a^b f_m(t)\,\mathrm{d}t \end{bmatrix}.$$

▶ Multivariable optimization

We begin with the basic definitions.

Definition 4.6 (Stationary/critical point)**.** Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be differentiable. We say $\mathbf{x}_* \in \mathbb{R}^n$ is a *stationary/critical point* of F if

$$\operatorname{rank}\big(\mathrm{J}F(\mathbf{x}_*)\big) < m.$$

In particular if $m = 1$, then we have a scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$ and in which case the condition above becomes the well-known one

$$\nabla f(\mathbf{x}_*) = 0,$$

i.e., a stationary point is simply one at which the gradient is zero. The corresponding function value $f(\mathbf{x}_*)$ is called a *stationary/critical value*.

Definition 4.7 (Global minimizer)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$. We say $\mathbf{x}_* \in \mathbb{R}^n$ is a *global minimizer* of f if

$$f(\mathbf{x}_*) \le f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$

Definition 4.8 (Local minimizer)**.** Let f be as above. We say $\mathbf{x}_* \in \mathbb{R}^n$ is a *local minimizer* if

$$f(\mathbf{x}_*) \le f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in N(\mathbf{x}_*) \subset \mathbb{R}^n,$$

where $N(\mathbf{x}_*)$ is a neighborhood of $\mathbf{x}_*$. Without loss of any generality, we can assume $N(\mathbf{x}_*) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{x}_*\| < \delta$ for some $\delta > 0$. So, alternatively, $\mathbf{x}_* \in \mathbb{R}^n$ is local minimizer iff there exists $\delta > 0$ such that

$$f(\mathbf{x}_*) \le f(\mathbf{x}) \quad \text{for all } \mathbf{x} \text{ such that } \|\mathbf{x} - \mathbf{x}_*\| < \delta.$$

Definition 4.9 (Strict local minimizer)**.** Let f be as above. We say $\mathbf{x}_* \in \mathbb{R}^n$ is a *strict local minimizer* if there is δ such that $f(\mathbf{x}_*) < f(\mathbf{x})$ for all $\mathbf{x}$ such that $0 < \|\mathbf{x}_* - \mathbf{x}\| < \delta$.

Global optimization is *very* difficult and generally impossible except in very special cases (an important special case is convex optimization). We shall instead look at local minimization problems. In local minimization problems, the goal is to find the local minimizer $\mathbf{x}_*$, not just the local minimum $f(\mathbf{x}_*)$. But even local optimization is difficult if we don't assume something more about the function f. The most common assumption is that $f \in C^1(\mathbb{R}^n)$ or $C^2(\mathbb{R}^n)$. So we want to characterize notions like local minimizer/minimum in terms of first and second derivatives.

The optimality conditions that we gave last time are in essence zeroth order conditions, i.e., defined in terms of the values of the objective function f. These are the most general but also the hardest to use. As soon as we assume that $f \in C^1(\mathbb{R}^n)$ or $C^2(\mathbb{R}^n)$, we may derive more easily checkable conditons that define local minimizer/minimum in terms of derivatives. We have seen several such results in the univariate case and now we will do the multivariate versions.

► Necessary conditions for optimality

In the following proofs of Theorems 4.10 and 4.13, you are reminded of the fact that if a univariate continuous function is positive (resp. negative) at a point $t_0 \in \mathbb{R}$, then it must be positive (resp. negative) in a neighborhood of t_0. This is almost trivial: say $g(t_0) > 0$, then pick $\varepsilon = g(t_0)/2$, by continuity at t_0 there exists $\delta > 0$ so that whenever $|t - t_0| < \delta$, we have $|g(t) - g(t_0)| < g(t_0)/2$, i.e., $-g(t_0)/2 < g(t) - g(t_0) < g(t_0)/2$. So $g(t) > g(t_0)/2 > 0$ for all $t \in (t_0 - \delta, t_0 + \delta)$.

In the proof of Theorem 4.10, we will apply this to the continuous univariate function of t

$$g(t) = \nabla f(\mathbf{x}_* + t\mathbf{p})^\top \mathbf{p},$$

and in the proof of Theorem 4.13, we will apply it to

$$g(t) = \mathbf{p}^\top \nabla^2 f(\mathbf{x}_* + t\mathbf{p})\mathbf{p}.$$

We now give the multivariate version of the first derivative test: Any local minimizer must be a stationary point.

Theorem 4.10 (First Order Necessary Condition)**.** *Let $f \in C^1(\mathbb{R}^n)$. If $\mathbf{x}_* \in \mathbb{R}^n$ is a local minimizer, then $\nabla f(\mathbf{x}_*) = \mathbf{0}$.*

Proof. Suppose $\nabla f(\mathbf{x}_*) \neq \mathbf{0}$. Let $\mathbf{p} = -\nabla f(\mathbf{x}_*)$. Then $\nabla f(\mathbf{x}_*)^\top \mathbf{p} = -\|\nabla f(\mathbf{x}_*)\|^2$. Since ∇f is continuous at $\mathbf{x}_*$, there exists $\delta > 0$ such that

$$\nabla f(\mathbf{x}_* + t\mathbf{p})^\top \mathbf{p} < 0$$

for all $t \in [0, \delta]$. For any $t \in (0, \delta]$, Taylor's Theorem applied with the nonzero vector $t\mathbf{p} \in \mathbb{R}^n$ implies

$$f(\mathbf{x}_* + t\mathbf{p}) = f(\mathbf{x}_*) + t\nabla f(\mathbf{x}_* + \xi\mathbf{p})^\top \mathbf{p}$$

for some $\xi \in (0, t)$. Then

$$f(\mathbf{x}_* + t\mathbf{p}) < f(\mathbf{x}_*)$$

for all $t \in (0, \delta]$. But then $\mathbf{x}_*$ is not a local minimizer, a contradiction. □

The above proof contains an important idea that we will later use in optimization algorithms:

Definition 4.11 (Descent direction)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable at $\mathbf{x}$. A vector $\mathbf{p}$ is a *descent direction* of f at $\mathbf{x}$ if there exists $\delta > 0$ such that $f(\mathbf{x} + t\mathbf{p}) < f(\mathbf{x})$ for all $t \in (0, \delta)$.

The descent direction is a direction leading away from $\mathbf{x}$ along which f decreases. For example, the choice of $\mathbf{p}$ in the proof of the first order necessary condition is a descent direction at $\mathbf{x}$. Moral: can always find a descent direction if $\mathbf{x}$ is not a local minimizer: $\mathbf{p} = -\nabla f(\mathbf{x})$.

We have been raving about convex optimization all this time. The reason is that for convex optimization, the first order necessary condition is also sufficient; not just that but it even guarantees a global optimality.

Before introducing the second order condition, we remind the reader of the following definition.

Definition 4.12. A matrix $A \in \mathbb{R}^{n\times n}$ is *positive semidefinite* if $\mathbf{x}^\top A\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$, and is *positive definite* if $\mathbf{x}^\top A\mathbf{x} > 0$ for all $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^n$. We also say that A is negative semidefinite (resp. negative definite) if $-A$ is positive semidefinite (resp. positive definite). We will use the following notations:

- positive definite: $A \succ 0$,

- positive semdefinite: $A \succeq 0$,
- negative definite: $A \prec 0$,
- negative semidefinite $A \preceq 0$.

In all our discussions involving definiteness, we will only be dealing with symmetric matrices A so we will henceforth assume this, i.e., whenever we speak of positive/negative (semi)definiteness, the matrix involved is automatically symmetric. With this in mind, we have the following characterizations:

- $A \succ 0$ if and only if all eigenvalues of A are positive;
- $A \succeq 0$ if and only if all eigenvalues of A are nonnegative;
- $A \succ 0$ if and only if A is an invertible covariance matrix;
- $A \succeq 0$ if and only if A is a covariance matrix.

Note that a covariance matrix is one of the form $B^\mathsf{T} B \in \mathbb{R}^{n\times n}$ for some matrix $B \in \mathbb{R}^{n\times p}$.

Theorem 4.13 (Second Order Necessary Conditions)**.** *Let $f \in C^2(\mathbb{R}^n)$. If $\mathbf{x}_* \in \mathbb{R}^n$ is a local minimizer, then*

$$\nabla f(\mathbf{x}_*) = \mathbf{0} \quad \text{and} \quad \nabla^2 f(\mathbf{x}_*) \succeq 0.$$

Proof. The first equality is true by the first order necessary condition. Now suppose for contradiction that $\nabla^2 f(\mathbf{x}_*)$ not positive semidefinite. Then there is some nonzero $\mathbf{p} \in \mathbb{R}^n$ such that $\mathbf{p}^\mathsf{T}\nabla^2 f(\mathbf{x}_*)\mathbf{p} < 0$. Since $\nabla^2 f$ is continuous at $\mathbf{x}_*$, there exists $\delta > 0$ such that

$$\mathbf{p}^\mathsf{T}\nabla^2 f(\mathbf{x}_* + t\mathbf{p})\mathbf{p} < 0$$

for all $t \in [0, \delta]$. By Taylor's theorem, for all $t \in (0, \delta)$,

$$f(\mathbf{x}_* + t\mathbf{p}) = f(\mathbf{x}_*) + \underbrace{t\nabla f(\mathbf{x}_*)^\mathsf{T}\mathbf{p}}_{=0} + \underbrace{\frac{t^2}{2}\mathbf{p}^\mathsf{T}\nabla^2 f(\mathbf{x}_* + \xi t\mathbf{p})\mathbf{p}}_{<0} < f(\mathbf{x}_*)$$

for $\xi \in (0, 1)$ and t nonzero. Hence $\mathbf{x}_*$ is not a local minimizer. □

▶ Sufficient conditions for optimality

In our study of univariate optimization, we have a first order sufficient condition: if $f'(x) < 0$ in a left neighborhood of x_*, and $f'(x) > 0$ in a right neighborhood of x_*, then x_* is a minimizer. Note that the first order sufficient condition does not require f to be differentiable at $x = x_*$, i.e., $f'(x_*)$ may not even exist. There is a multivariate generalization.

Theorem 4.14 (First Order Sufficient Conditions)**.** *Let f be continuous at $\mathbf{x}_* \in \mathbb{R}^n$ and differentiable in a neighborhood of $\mathbf{x}_*$. If there is a $\delta > 0$ with*

$$\nabla f(\mathbf{x}_* + \mathbf{p})^\mathsf{T}\mathbf{p} > 0$$

whenever $\|\mathbf{p}\| < \delta$, then $\mathbf{x}_*$ is a local minimizer. Likewise, if there is a $\delta > 0$ with

$$\nabla f(\mathbf{x}_* + \mathbf{p})^\mathsf{T}\mathbf{p} < 0$$

whenever $\|\mathbf{p}\| < \delta$*, then* $\mathbf{x}_*$ *is a local maximizer.*

Proof. We will just prove the local minimizer statement. We may assume that f is differentiable in the δ-neighborhood of $\mathbf{x}_*$. Let $\mathbf{0} \neq \mathbf{p} \in \mathbb{R}^n$ with $\|\mathbf{p}\| < \delta$. Define the univariate function $g(t) = f(\mathbf{x}_* + t\mathbf{p})$. Then g is continuous on $[0,1]$ and differentiable on $(0,1)$ with

$$g'(t) = \nabla f(\mathbf{x}_* + t\mathbf{p})^\intercal \mathbf{p}.$$

By the Mean Value Theorem, there exists $\xi \in (0,1)$ such that $g(1) - g(0) = g'(\xi)$, i.e.,

$$\begin{aligned} f(\mathbf{x}_* + \mathbf{p}) - f(\mathbf{x}_*) &= \nabla f(\mathbf{x}_* + \xi\mathbf{p})^\intercal \mathbf{p} \\ &= \frac{1}{\xi} \nabla f(\mathbf{x}_* + \xi\mathbf{p})^\intercal (\xi\mathbf{p}) \\ &= \frac{1}{\xi} \nabla f(\mathbf{x}_* + \mathbf{q})^\intercal \mathbf{q} \end{aligned}$$

where we have written $\mathbf{q} \coloneqq \xi\mathbf{p}$. Since $0 < \|\mathbf{q}\| = \xi\|\mathbf{p}\| < \|\mathbf{p}\| < \delta$, we have $\nabla f(\mathbf{x}_* + \mathbf{q})^\intercal \mathbf{q} > 0$. Hence $f(\mathbf{x}_* + \mathbf{p}) > f(\mathbf{x}_*)$. Since this holds for all $\mathbf{p}$ with $\|\mathbf{p}\| < \delta$, we conclude that $\mathbf{x}_*$ is a local minimizer. □

Note an equivalent way to state the condition "$\nabla f(\mathbf{x}_* + \mathbf{p})^\intercal \mathbf{p} > 0$ whenever $\|\mathbf{p}\| < \delta$" is:

$$\nabla f(\mathbf{x})^\intercal (\mathbf{x} - \mathbf{x}_*) > 0$$

whenever $\|\mathbf{x} - \mathbf{x}_*\| < \delta$. We have simply rewritten $\mathbf{x} = \mathbf{x}_* + \mathbf{p}$.

From the previous section, a necessary condition for $\mathbf{x}_*$ to be a local minimizer of a C^2-function is

$$\nabla f(\mathbf{x}_*) = \mathbf{0} \text{ and } \nabla^2 f(\mathbf{x}_*) \succeq 0.$$

We next show that if we change $\succeq$ to $\succ$, we obtain a sufficient condition:

$$\nabla f(\mathbf{x}_*) = \mathbf{0} \text{ and } \nabla^2 f(\mathbf{x}_*) \succ 0.$$

Fact 4.15. *Let* $F\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ *be a continuous function. If* $F(\mathbf{x}_*) \succ 0$ *then* $F(\mathbf{x}) \succ 0$ *for all* $\mathbf{x}$ *in some neighborhood of* $\mathbf{x}_*$*, i.e., there exists* $\delta > 0$ *such that* $F(\mathbf{x}) \succ 0$ *whenever* $\|\mathbf{x} - \mathbf{x}_*\| < \delta$*.*

This a multivariate version of the fact that if a continuous function is positive at some point, then it is positive in some neighborhood of the point.

Theorem 4.16 (Second Order Sufficient Conditions)**.** *Let* $f \in C^2(\mathbb{R}^n)$*. If* $\nabla f(\mathbf{x}_*) = \mathbf{0}$ *and* $\nabla^2 f(\mathbf{x}_*) \succ 0$*, then* $\mathbf{x}_* \in \mathbb{R}^n$ *is strict local minimizer.*

Proof. For $\mathbf{p} \neq \mathbf{0}$,

$$f(\mathbf{x}_* + \mathbf{p}) = f(\mathbf{x}_*) + \nabla f(\mathbf{x}_*)^\intercal \mathbf{p} + \frac{1}{2}\mathbf{p}^\intercal \nabla^2 f(\mathbf{x}_* + \xi\mathbf{p})\mathbf{p}$$

for some $\xi \in (0,1)$. Since $\nabla^2 f\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ is continuous, Fact 4.15 implies that $\nabla^2 f(\mathbf{x}) \succ 0$ in some δ-neighborhood of $\mathbf{x}_*$. So if $\|\mathbf{p}\| < \delta$, then $\nabla^2 f(\mathbf{x}_* + \xi\mathbf{p}) \succ 0$. Hence

$$f(\mathbf{x}_* + \mathbf{p}) = f(\mathbf{x}_*) + \underbrace{\nabla f(\mathbf{x}_*)^\intercal \mathbf{p}}_{=0} + \underbrace{\frac{1}{2}\mathbf{p}^\intercal \nabla^2 f(\mathbf{x}_* + \xi\mathbf{p})\mathbf{p}}_{>0}.$$

Therefore

$$f(\mathbf{x}_* + \mathbf{p}) > f(\mathbf{x}_*)$$

whenever $\|\mathbf{p}\| < \delta$. Hence $\mathbf{x}_*$ is a strict local minimizer. □

There is no necessary *and* sufficient condition. Take the second order conditions for example, we can see this already in the univariate case:

- The necessary condition is not sufficient: take $f(x) = x^3$; then $f'(0) = f''(0) = 0$, but 0 is not a local minimizer
- The sufficient condition is not necessary: take $f(x) = x^4$; then 0 is a strict local minimizer but $f'(0) = f''(0) = 0$.

Remark 4.17. How can we check whether a symmetric (as Hessians invariably are) matrix $X \in \mathbb{R}^{n\times n}$ is positive definite? Checking directly by the definition is infeasible, as is the determinant characterization for large matrices (think 2000×2000). The best way is via Cholesky decomposition: a symmetric $X \succ 0$ iff there is an invertible upper triangular matrix R such that $X = R^{\mathsf{T}}R$. The Cholesky decomposition can compute R in $O(n^3)$ time.

▶ Applications of unconstrained optimization

The most common modern applications of unconstrained optimization are in machine learning, more specifically supervised learning, which is just a fancy word for data fitting. Basically the problem is that we are given data of the form

$$(\mathbf{a}_1, b_1),\ (\mathbf{a}_2, b_2), \dots, (\mathbf{a}_m, b_m) \in \mathbb{R}^n \times \mathbb{R} \tag{4.18}$$

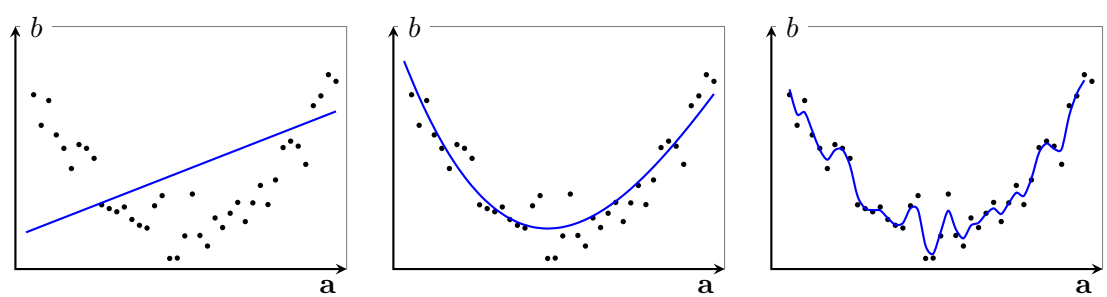
and we want find a simple function φ that sends $\mathbf{a}_i \in \mathbb{R}^n$ to $b_i \in \mathbb{R}$, $i = 1, \dots, m$. Now ideally φ should be as simple as possible so we want φ to be defined by a parameter $\mathbf{x} \in \mathbb{R}^p$ with a small p — but 'small' is a matter of opinion; we usually need $p \approx n$ but p can be as big as $p \approx kn^2$ for k-layer neural network. Whatever the case φ is going to depend on both the data $\mathbf{a} \in \mathbb{R}^n$ and the parameter $\mathbf{x} \in \mathbb{R}^p$, i.e., $\varphi : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$, and we want

$$\varphi(\mathbf{a}_1, \mathbf{x}) = b_1,\ \varphi(\mathbf{a}_2, \mathbf{x}) = b_2, \dots, \varphi(\mathbf{a}_m, \mathbf{x}) = b_m.$$

But this is too stringent — an exact fit is likely to be an overfit where you are also fitting errors in $\mathbf{a}_i$'s and b_i's so we would allow for

$$\varphi(\mathbf{a}_1, \mathbf{x}) \approx b_1,\ \varphi(\mathbf{a}_2, \mathbf{x}) \approx b_2, \dots, \varphi(\mathbf{a}_m, \mathbf{x}) \approx b_m \tag{4.19}$$

subjected to various interpretations of $\approx$.



Underfit Good fit Overfit

Example 4.20 (Regression)**.** The simplest but nonetheless very useful φ is one where $p = n$ and $\varphi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is simply given by inner product $\varphi(\mathbf{a}, \mathbf{x}) = \mathbf{a}^\mathsf{T}\mathbf{x}$. To simplify notations, we assume that the data in (4.18) is fixed and we introduce the shorthand $f_i : \mathbb{R}^p \to \mathbb{R}$,

$$f_i(\mathbf{x}) = \mathbf{a}_i^\mathsf{T}\mathbf{x} - b_i, \qquad i = 1, \dots, m,$$

and $F : \mathbb{R}^p \to \mathbb{R}^m$ with

$$F(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\mathsf{T}\mathbf{x} - b_1 \\ \mathbf{a}_2^\mathsf{T}\mathbf{x} - b_2 \\ \vdots \\ \mathbf{a}_m^\mathsf{T}\mathbf{x} - b_m \end{bmatrix} = A\mathbf{x} - \mathbf{b}$$

where

$$A = \begin{bmatrix} \mathbf{a}_1^\mathsf{T} \\ \mathbf{a}_2^\mathsf{T} \\ \vdots \\ \mathbf{a}_m^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{m \times n}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \in \mathbb{R}^m.$$

The most straightforward interpretation of $\approx$ in (4.19) is simply that for some choice of loss function $\ell : \mathbb{R} \to \mathbb{R}$, we want the total loss

$$f(\mathbf{x}) \coloneqq \sum_{i=1}^m \ell\big(f_i(\mathbf{x})\big)$$

to be small so the best $\mathbf{x}$ is obtained by solving the *empirical risk estimation problem*

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^m \ell\big(f_i(\mathbf{x})\big). \qquad \square$$

In the next few examples, we use the same notations as in Example 4.20.

Example 4.21 (Linear regression)**.** In linear regression, we choose $\ell(t) = t^2$ and so the problem becomes

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^m f_i(\mathbf{x})^2 = \min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^m (\mathbf{a}_i^\mathsf{T}\mathbf{x} - b_i)^2 = \min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|^2.$$

This minimization problem is called an *ordinary least squares* problem. Note that the objective function f is differentiable. By applying the first and second order necessary conditions we see that

$$\nabla f(\mathbf{x}_*) = \mathbf{0} \quad \Leftrightarrow \quad A^\mathsf{T}A\mathbf{x}_* = A^\mathsf{T}\mathbf{b}, \qquad \nabla^2 f(\mathbf{x}_*) = A^\mathsf{T}A \succeq 0.$$

The linear system that result from the first order condition is called the *normal equation*. In the event when $\operatorname{rank}(A) = n$, we in fact have

$$\nabla^2 f(\mathbf{x}) = A^\mathsf{T}A \succ 0$$

and so the second order sufficient condition holds. Since positive definite matrices are always invertible, we obtain the unique solution

$$\mathbf{x}_* = (A^\mathsf{T}A)^{-1}A^\mathsf{T}\mathbf{b}. \qquad \square$$

Standard linear regression is very sensitive to outliers in the data set because $\ell(t) = t^2$ blows up when t is large. One way to reduce sensitivity is to replace it with $\ell(t) = |t|$.

Example 4.22 (Least absolute deviations regression)**.** In least absolute deviations regression, we choose $\ell(t) = |t|$ and so the problem becomes

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^m |f_i(\mathbf{x})| = \min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^m |\mathbf{a}_i^\mathsf{T}\mathbf{x} - b_i| = \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_1$$

where the 1-norm or l^1-norm of $\mathbf{v} \in \mathbb{R}^m$ is defined by

$$\|\mathbf{v}\|_1 := |v_1| + |v_2| + \cdots + |v_m|.$$

For this reason, this minimization problem is also called an l^1-regression problem. Note that the objective function f is no longer differentiable. So the first and second order necessary conditions no longer apply. However, we will see that this unconstrained optimization problem can be transformed into a constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^\mathsf{T}\mathbf{y} \\ \text{subject to} \quad & -\mathbf{y} \le A\mathbf{x} - \mathbf{b} \le \mathbf{y} \end{aligned} \tag{4.23}$$

by introducing m additional variables $\mathbf{y} = (y_1, \dots, y_m)$. Here $\mathbf{1} \in \mathbb{R}^m$ denotes a vector whose every coordinate is 1 and whenever we have $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$, the inequality $\mathbf{u} \le \mathbf{v}$ will always be taken in a coordinatewise sense, i.e., $u_i \le v_i$ for all $i = 1, \dots, m$. The constrained optimization problem (4.23) is not only differentiable but is in fact a *linear programming* problem — one of the most well-studied type of optimization problem that we will discuss later. □

Insensitivity to outliers is often called *robustness* in statistics. Robustness is usually a good thing but we may imagine some uncommon scenarios where we want extreme sensitivity to a small number of outliers. In this case, we replace the sum by max,

$$\sum_{i=1}^m \quad \rightsquigarrow \quad \max_{i=1,\dots,m}$$

in the objective function of the least absolute deviations regression; the result $\max_{i=1,\dots,m}|f_i(\mathbf{x})|$ is called the maximum absolute deviation in statistics. This gives us a highly non-robust regression technique.

Example 4.24 (Maximum absolute deviation regression)**.** Maximum absolute deviation regression solves

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n} \Big(\max_{i=1,\dots,m} |f_i(\mathbf{x})|\Big) = \min_{\mathbf{x}\in\mathbb{R}^n} \Big(\max_{i=1,\dots,m} |\mathbf{a}_i^\mathsf{T}\mathbf{x} - b_i|\Big) = \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_\infty \tag{4.25}$$

where the ∞-norm or l^∞-norm of $\mathbf{v} \in \mathbb{R}^m$ is defined by

$$\|\mathbf{v}\|_\infty := \max(|v_1|, |v_2|, \dots, |v_m|).$$

For this reason, this minimization problem is also called an l^∞-regression problem or sometimes Chebyshev approximation problem. Again this nonsmooth unconstrained optimization problem can be transformed into a smooth constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & -t\mathbf{1} \leq A\mathbf{x} - \mathbf{b} \leq t\mathbf{1}, \end{aligned}$$

by introducing an additional variable t. This is also a linear programming problem. □

There is a clever way to combine the advantages of of linear regression (differentiability or smoothness or loss function) with that of least absolute deviations regression (reduced sensitivity to outliers) by choosing ℓ appropriately.

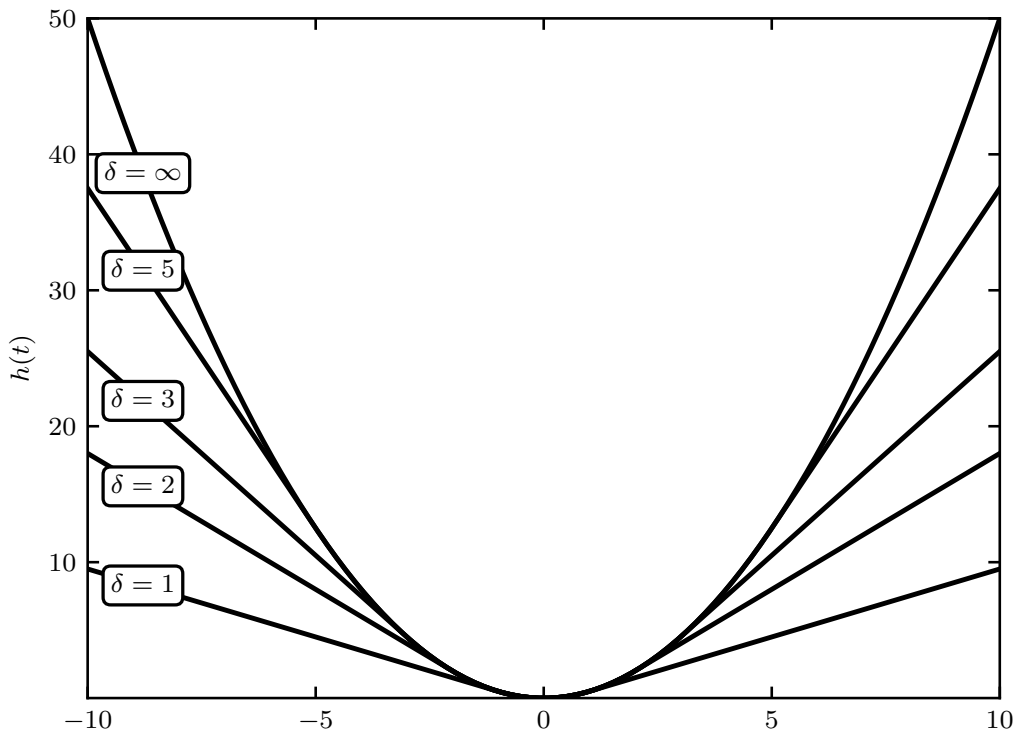


Figure 10: Huber function for different values of δ.

Example 4.26 (Robust regression)**.** In robust regression, the loss function ℓ is chosen to be the *Huber function* $h : \mathbb{R} \to \mathbb{R}$ defined by

$$h(t) = \begin{cases} t^2 & \text{if } |t| < \delta, \\ \delta(2|t| - \delta) & \text{if } |t| > \delta. \end{cases}$$

It is clear from Figure 10 that $h(t)$ resembles $\ell(t) = t^2$ when t has small magnitude and $\ell(t) = |t|$ when t has large magnitude, where 'small' and 'large' is controlled by the parameter $\delta > 0$.

So the problem becomes

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} h\big(f_i(\mathbf{x})\big) = \min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} h(\mathbf{a}_i^\mathsf{T}\mathbf{x} - b_i).$$

So we obtain a smooth unconstrained optimization problem. □

Figure 11 illustrates the effect of the different regression techniques in the presence of outliers. The Python code `regression.py` is included for you to experiment with them.

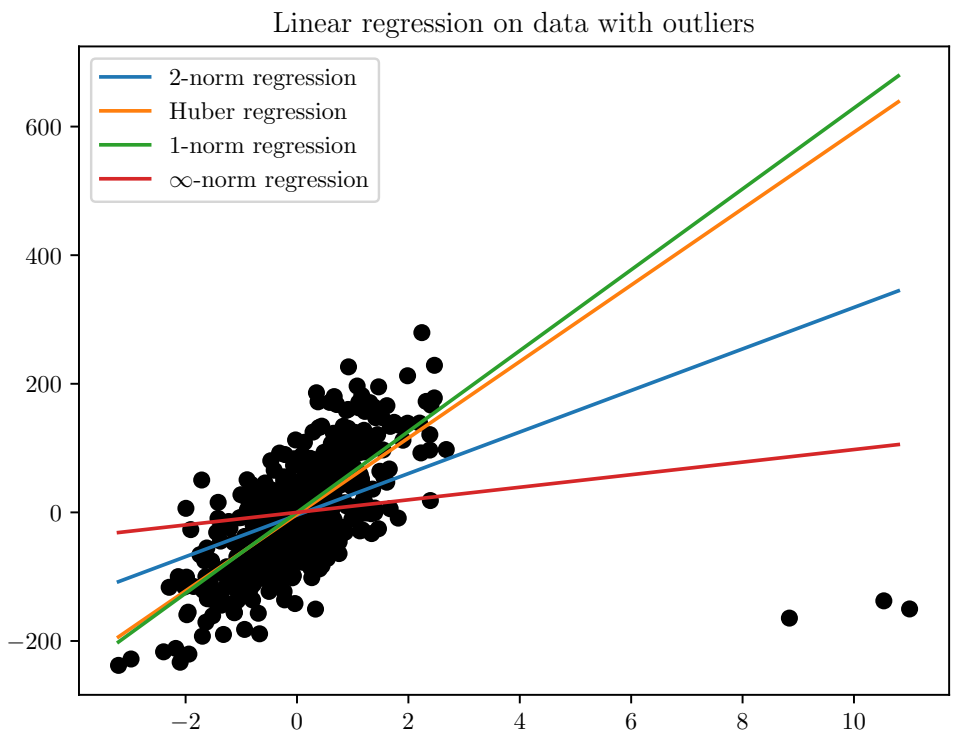


Figure 11: Different types of regression with different loss functions.

Lecture 5 (Tuesday, October 24)

▶ More applications of unconstrained optimization

Maximum absolute deviation regression is not so common but it is the close cousin of an extremely common class of approximation problem.

Example 5.1 (Piecewise linear regression)**.** Note that if we drop the absolute value in the objective function of (4.25), we obtain

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n}\Big(\max_{i=1,\dots,m} f_i(\mathbf{x})\Big) = \min_{\mathbf{x}\in\mathbb{R}^n}\Big(\max_{i=1,\dots,m} \mathbf{a}_i^\mathsf{T}\mathbf{x} - b_i\Big).$$

The objective function $f(\mathbf{x}) = \max_{i=1,\dots,m} \mathbf{a}_i^\mathsf{T}\mathbf{x} - b_i$ is a continuous piecewise linear function and in fact any continuous piecewise linear function can be expressed in this form. This is of course nondifferentiable in general but recall from the last lecture that we have

$$\max(x_1,\dots,x_m) \le \operatorname{lse}(x_1,\dots,x_m) \le \max(x_1,\dots,x_m) + \log m$$

and so the log-sum-exp function can be used as a smooth proxy for the max function. With this, we obtain an unconstrained geometric programming problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} g(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n} \log\left[\sum_{i=1}^m e^{\mathbf{a}_i^\mathsf{T}\mathbf{x}+b_i}\right].$$

As an exercise, solve this problem. Recall that we have already calculated its gradient and Hessian in the previous lecture. □

As we saw, in the design of optimization problems, we often start from a basic objective function and then modify it to have features that we want like robustness, paying some attention to the difficulty of the resulting optimization problem, with differentiability, convexity, unconstrained being desirable. As another example of this principle at work, we modify the linear regression problem to reduce the effect of having highly correlated data.

Example 5.2 (Ridge regression)**.** In ridge regression we add a *regularization*

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n}\left[\sum_{i=1}^m f_i(\mathbf{x})^2 + \gamma^2\sum_{i=1}^m x_i^2\right] = \min_{\mathbf{x}\in\mathbb{R}^n}\|A\mathbf{x}-\mathbf{b}\|^2 + \gamma^2\|\mathbf{x}\|^2. \tag{5.3}$$

A more general variant does

$$\min_{\mathbf{x}\in\mathbb{R}^n}\|A\mathbf{x}-\mathbf{b}\|^2 + \|\Gamma\mathbf{x}\|^2$$

for some given matrix $\Gamma \in \mathbb{R}^{p\times n}$ and the special case $\Gamma = \gamma I$ reduces to the basic version. Note that this is again a smooth unconstrained optimization problem and we can apply first and second order necessary conditions:

$$\nabla f(\mathbf{x}_*) = \mathbf{0} \quad\Leftrightarrow\quad (A^\mathsf{T}A + \Gamma^\mathsf{T}\Gamma)\mathbf{x}_* = A^\mathsf{T}\mathbf{b}, \qquad \nabla^2 f(\mathbf{x}_*) = A^\mathsf{T}A + \Gamma^\mathsf{T}\Gamma \succeq 0.$$

This procedure is called *Tikhonov regularization* in engineering. In this case because the matrix Γ is something we choose we have some flexibility to choose it so that the second order sufficient condition is in fact satisfied

$$\nabla^2 f(\mathbf{x}_*) = A^\mathsf{T} A + \Gamma^\mathsf{T}\Gamma \succ 0$$

and in which case we obtain a unique solution

$$\mathbf{x}_* = (A^\mathsf{T} A + \Gamma^\mathsf{T}\Gamma)^{-1} A^\mathsf{T}\mathbf{b}.$$

For instance if $\Gamma = \gamma I$, then by choosing γ large enough, we can guarantee that $A^\mathsf{T} A + \gamma I$ is invertible. One can show that for any value of γ, the unconstrained problem (5.3) is equivalent to a constrained problem

$$\begin{aligned} \text{minimize} \quad & \|A\mathbf{x} - \mathbf{b}\|^2 \\ \text{subject to} \quad & \|\mathbf{x}\|^2 = c^2 \end{aligned}$$

for some value of c. This illustrates two things: (i) this is one way of transforming a constrained problem into an unsconstrained one that we will discuss more generally under the heading of *penalty function*; (ii) it gives us another interpretation of what we are trying to achieve in (5.3), namely, we restrict the size of our solution $\mathbf{x}_*$ by making its norm fixed at c.

There are many other variations along this line. For example we might simultaneously reduce the effect of outliers and highly correlated data with

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_1 + \gamma^2\|\mathbf{x}\|^2,$$

or improve sparsity with

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|^2 + \gamma^2\|\mathbf{x}\|_1. \qquad \square$$

Note that even though we set $p = n$ in Example 4.20, this does not need to be the case. Take for example the special case when $n = 1$, i.e., $a_1, \dots, a_n \in \mathbb{R}$ are scalars, we have the following type of regression problem.

Example 5.4 (Polynomial regression)**.** We set $\varphi(a, \mathbf{x}) = x_0 + ax_1 + a^2x_2 \cdots + a^p x_p$. Using the notations in Example 4.20, the functions $f_i : \mathbb{R}^{p+1} \to \mathbb{R}$ are given by

$$f_i(\mathbf{x}) = x_0 + a_i x_1 + a_i^2 x_2 \cdots + a_i^p x_p - b_i, \qquad i = 1, \dots, m,$$

and $F : \mathbb{R}^{p+1} \to \mathbb{R}^m$ with

$$F(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^p \\ 1 & a_2 & a_2^2 & \dots & a_2^p \\ 1 & a_3 & a_3^2 & \dots & a_3^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_m & a_m^2 & \dots & a_m^p \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}.$$

The optimization problem involved is

$$\min_{\mathbf{x}\in\mathbb{R}^m} \left\| \begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^p \\ 1 & a_2 & a_2^2 & \dots & a_2^p \\ 1 & a_3 & a_3^2 & \dots & a_3^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_m & a_m^2 & \dots & a_m^p \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix} \right\|^2$$

and can be solved in the same way as Example 4.21. The matrix involved is called a Vandermonde matrix and there are special fast algorithms that exploits this structure. We show an example in Figure 12. Polynomial regression allows one to fit every nook and cranny of a

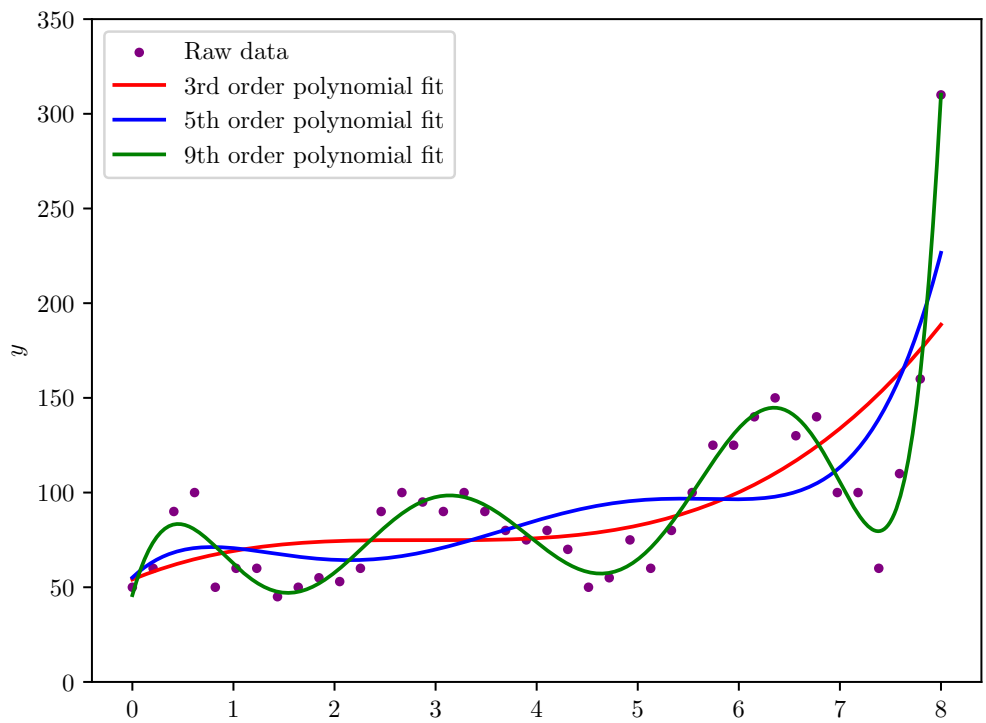


Figure 12: Polynomial regression for $p = 3, 5, 9$.

data set by setting p high enough, but of course at the risk of overfitting. The Python code `poly.py` is included for you to experiment with them. □

The problems that we have considered up to this point are all regression problems, basically meaning that $b_1, \dots, b_m \in \mathbb{R}$ can be any real values. Another equally important class of problems are *classification* problems, where $b_1, \dots, b_m$ take values from a prespecified finite subset of $\mathbb{R}$, e.g., $\{-1, +1\}$ representing yes or no; $\{-1, 0, +1\}$ representing buy, hold, or sell; $\{1, 2, 3, 4\}$ representing A, C, G, T in DNA molecules; $\{1, 2, 3, 4, 5\}$ representing number of stars in movie ratings; $\{0, 1, \dots, 9\}$ representing the digits; $\{0, 1, \dots, 255\}$ representing the ASCII characters; etc. Here $b_1, \dots, b_m$ are often called *labels* and $\mathbf{a}_1, \dots, \mathbf{a}_m$ are often called *feature vectors*.

Example 5.5 (Classification)**.** For simplicity we consider the case where the labels are in $\{-1,+1\}$ or *binary classification*. Here the simplest φ is one where $p = n+1$ and $\varphi : \mathbb{R}^n \times \mathbb{R}^{n+1} \to \mathbb{R}$ is simply given by $\varphi(\mathbf{a},\mathbf{x},y) = \mathbf{a}^\mathsf{T}\mathbf{x} + y$. Note that here the parameter is $(\mathbf{x},y) \in \mathbb{R}^n \times \mathbb{R} = \mathbb{R}^{n+1}$ and the labels are $b_i \in \{-1,+1\}$. To simplify notations, we assume that the data in (4.18) is fixed and we introduce the shorthand $f_i : \mathbb{R}^p \to \mathbb{R}$,

$$f_i(\mathbf{x},y) = b_i\varphi(\mathbf{a}_i,\mathbf{x},y) = b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y), \qquad i = 1,\dots,m.$$

The intuition here is basically that we want the sign of $\mathbf{a}_i^\mathsf{T}\mathbf{x} + y$ to agree with $b_i \in \{-1,+1\}$ and the loss function $\ell : \mathbb{R} \to \mathbb{R}$ will be chosen with this mind. As in the case of Example 4.20, we want the total loss

$$f(\mathbf{x},y) \coloneqq \sum_{i=1}^m \ell\big(f_i(\mathbf{x},y)\big)$$

to be small and obtain the best $(\mathbf{x},y)$ by solving the empirical risk estimation problem

$$\min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} f(\mathbf{x},y) = \min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} \sum_{i=1}^m \ell\big(f_i(\mathbf{x},y)\big). \qquad \square$$

We want the sign of $\mathbf{a}_i^\mathsf{T}\mathbf{x} + y$ to agree with $b_i \in \{-1,+1\}$; so here is the most straightforward way to choose our loss function ℓ:

- if $b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y) \geq 0$, there is no error, so the loss is zero;
- if $b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y) < 0$, then the error is proportional to how far the value is from zero, so the loss is $|b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y)| = -b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y)$.

In other words, a simple loss for the ith feature/label is $\max\big(0, -b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y)\big)$ and so the total loss is

$$\sum_{i=1}^m \max\big(0, 1 - b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y)\big),$$

corresponding to a loss function of $\ell(t) = \max(0,t)$. There are two variants along this line of reasoning.

Example 5.6 (Support vector machines)**.** In a well-known technique called support vector machines or SVM, instead of $\ell(t) = \max(0,-t)$, we choose $\ell(t) = \max(0,1-t)$, called the *hinge loss*, and optimize

$$\min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} f(\mathbf{x},y) = \min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} \left[\sum_{i=1}^m \max\big(0, 1 - b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y)\big)\right].$$

The hinge loss is not differentiable but the unconstrained problem above is equivalent to the linear programming problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^\mathsf{T}\mathbf{e} \\ \text{subject to} \quad & 1 - b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y) \leq e_i, \quad i = 1,\dots,m, \\ & \mathbf{e} \geq \mathbf{0}, \end{aligned}$$

where we have introduced m extra variables $\mathbf{e} = (e_1,\dots,e_m)$. When we discuss constrained optimization, we will see the interpretation of this problem and why it is called support vector machine. $\square$

The following is an alternative way to deal with the nondifferentiability of the hinge loss.

Example 5.7 (Logistic regression)**.** As in the case of piecewise linear regression, we may use the log-sum-exp function $\ell(t) = \log(1 + e^{-t})$ to obtain a smooth proxy for $\ell(t) = \max(0, -t)$. This gives us

$$\min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} f(\mathbf{x}, y) = \min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} \left[\sum_{i=1}^{m} \log\bigl(1 + e^{-b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x}+y)}\bigr)\right],$$

which is differentiable. This is called logistic regression and has a nice probabilistic interpretation that we will not go into. It has the advantage of being readily generalizable from binary classification to a multiclass classification. □

Example 5.8 (Regularization and kernelization)**.** The regularization mentioned earlier also applies to support vector machines and logistic regression, so that we optimize

$$\min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} \left[\sum_{i=1}^{m} \max\bigl(0, 1 - b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x} + y)\bigr) + \gamma^2\|\mathbf{x}\|^2\right],$$

or

$$\min_{(\mathbf{x},y)\in\mathbb{R}^{n+1}} \left[\sum_{i=1}^{m} \log\bigl(1 + e^{-b_i(\mathbf{a}_i^\mathsf{T}\mathbf{x}+y)}\bigr) + \gamma^2\|\mathbf{x}\|^2\right],$$

or with yet other regularization term. There is also a powerful twist called *kernelization*, which is likely the most dominant technique in machine learning before deep neural networks had its clean sweep of the field. I briefly described it during lecture but now realize that it will take us too far afield.

▶ Algorithms for nonlinear/convex optimization: line search

In nonlinear optimization problems, the objective function f is not guaranteed to be convex. In convex optimization problems, f is convex. The algorithms for nonlinear and convex optimization are identical, however.

The general scheme for *line search* methods in $\mathbb{R}^n$ is as follows:

(i) Choose a *starting point* $\mathbf{x}_0 \in \mathbb{R}^n$

(ii) Choose a *search direction* $\mathbf{p}_k \in \mathbb{R}^n$

(iii) Choose a *step size/length* $\alpha_k \in \mathbb{R}$

(iv) Do $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

(v) Repeat items 2 – 4 until *stopping condition* is met

The major question we answer in the next few lectures is: *how do we find search direction p_k and step size α_k?* If f is not differentiable, use non-differentiable optimization techniques. If f is differentiable, we can use Taylor's theorem.

Recall the first and second order approximations for a C^2 function f:

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \approx f(\mathbf{x}_k) + \alpha_k \nabla f(\mathbf{x}_k)^\mathsf{T} \mathbf{p}_k \tag{linear}$$

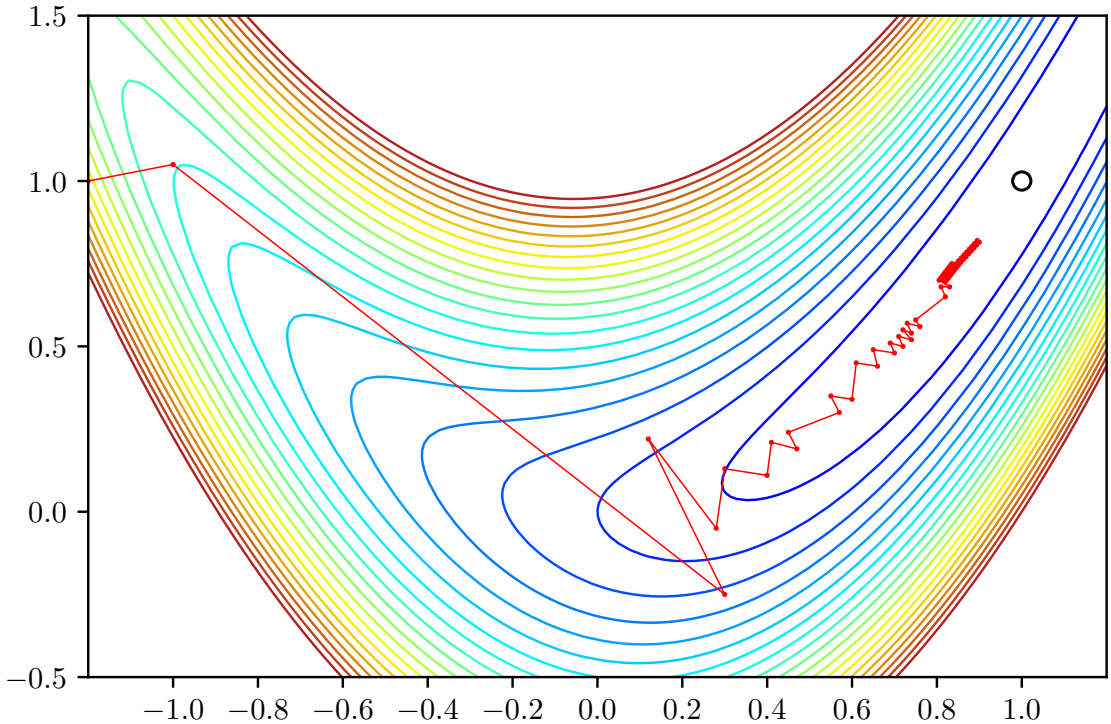


Figure 13: Line search method — red points are $\mathbf{x}_k$, red lines are $\alpha_k \mathbf{p}_k$. This is an example of a contour plot — colored curves are level curves $f(\mathbf{x}) = c$ for various values of c with red > orange > yellow > green > blue > indigo > violet.

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \approx f(\mathbf{x}_k) + \alpha_k \nabla f(\mathbf{x}_k)^\mathsf{T} \mathbf{p}_k + \frac{1}{2}\alpha_k^2 \mathbf{p}_k^\mathsf{T} \nabla^2 f(\mathbf{x}_k) \mathbf{p}_k \tag{quadratic}$$

First order approximations are used in steepest descent and conjugate gradient methods; second order approximations are used in Newton's method, interior point methods (IPM), and quasi-Newton methods. Second order methods require the computation of the Hessian, which is $O(n^2)$. In contrast, evaluating a gradient is only $O(n)$.

▶ Search direction from linear approximation

We start by describing some approaches for computing α_k and $\mathbf{p}_k$. First, suppose $\alpha_k = 1$ for all $k = 0, 1, \ldots$. Using the linear approximation, we find $\mathbf{p}_k$ by solving

$$\min_{\|\mathbf{p}\|=1} (f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\mathsf{T} \mathbf{p}).$$

Note that without the $\|\mathbf{p}\| = 1$ condition, the minimum will be negative infinity if the gradient is not 0. By Cauchy–Schwarz,

$$-\|\nabla f(\mathbf{x}_k)\| \, \|\mathbf{p}\| \le \nabla f(\mathbf{x}_k)^\mathsf{T} \mathbf{p} \le \|\nabla f(\mathbf{x}_k)\| \, \|\mathbf{p}\|.$$

From this, we see that the minimum is attained when

$$\mathbf{p}_k = \frac{-\nabla f(x_k)}{\|\nabla f(x_k)\|}.$$

The resulting algorithm is the *normalized steepest descent* algorithm.

Algorithm 3 Normalized steepest descent

Require: $\mathbf{x}_0 \in \mathbb{R}^n$, tolerance $\varepsilon > 0$ small

```
procedure DESCENT(x_0, ε)
    x ← x_0
    while ‖∇f(x)‖ > ε do
        p ← −∇f(x)/‖∇f(x)‖
        find step size α > 0
        x ← x + αp
    end while
    return x
end procedure
```

We next discuss how to find α to guarantee convergence to a stationary point.

▶ Search direction from quadratic approximation

Using the quadratic approximation, we find $\mathbf{p}_k$ by solving

$$\min_{\mathbf{p}\in\mathbb{R}^n} \left(f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\intercal \mathbf{p} + \frac{1}{2}\mathbf{p}^\intercal \nabla^2 f(\mathbf{x}_k)\mathbf{p} \right).$$

By the second-order sufficient condition for optimality, if $\nabla^2 f(\mathbf{x}_k) \succ 0$ (e.g., if f strictly convex), then the minimizer is $\mathbf{p}_k = -(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k)$. This direction is called the *Newton direction.*

▶ Descent direction

Definition 5.9 (Descent direction)**.** If the search direction satisfies

$$f(\mathbf{x}_k + \alpha \mathbf{p}_k) < f(\mathbf{x}_k)$$

for some $\alpha > 0$ then $\mathbf{p}_k$ is called a *descent direction.*

Algorithms in optimization are generally based on one principle: finding a descent direction and taking a step in that direction. At the next iteration, the value of the function decreases. Ideally, the algorithm will reach a local minimum.

Some optimization specialists take the condition $\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k < 0$ to be the definition of descent direction. The reason is as follows.

Theorem 5.10 (First Order Sufficient Condition for Descent)**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be C^1. If $\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k < 0$, then there is some $\alpha > 0$ such that $f(\mathbf{x}_k + \alpha\mathbf{p}_k) < f(\mathbf{x}_k)$.*

Proof. If $\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k < 0$, the continuity of ∇f implies that $\nabla f(\mathbf{x}_k + \alpha\mathbf{p}_k)^\intercal \mathbf{p}_k < 0$ for sufficiently small α. For all such α, Taylor's theorem furnishes $\xi \in (0,1)$ such that

$$f(\mathbf{x}_k + \alpha\mathbf{p}_k) = f(\mathbf{x}_k) + \alpha \nabla f(\mathbf{x}_k + \xi\alpha\mathbf{p}_k)^\intercal \mathbf{p}_k < f(\mathbf{x}_k). \qquad \square$$

Notation 5.11. Common shorthand in optimization:

$$f_k = f(\mathbf{x}_k), \qquad \mathbf{g}_k = \nabla f(\mathbf{x}_k), \qquad H_k = \nabla^2 f(\mathbf{x}_k).$$

These are useful for naming your variables when you code up optimization algorithms.

Theorem 5.12.

(i) *If $\nabla^2 f(\mathbf{x}_k) \succ 0$, then the Newton step $\mathbf{p}_k = -\nabla^2 f(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$ is a descent direction.*

(ii) *The steepest descent direction, $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$, is always a descent direction.*

Proof. By Theorem 5.10, if $\mathbf{g}_k^\intercal \mathbf{p}_k < 0$, then $\mathbf{p}_k$ is a descent direction:

(i) $\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k = \mathbf{g}_k^\intercal \mathbf{p}_k = -\mathbf{g}_k^\intercal H_k^{-1}\mathbf{g}_k < 0$

(ii) $\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k = -\mathbf{g}_k^\intercal \mathbf{g}_k = -\|\mathbf{g}_k\|^2 < 0.$ □

In general we will not have $\nabla^2 f(\mathbf{x}_k) \succ 0$ for all $\mathbf{x}_k$, so the Newton direction may not be a descent direction. In this case, we find $\mathbf{p}_k$ such that $\mathbf{p}_k^\intercal \nabla^2 f(\mathbf{x}_k)^{-1}\mathbf{p}_k < 0$ (direction of negative curvature). If $\nabla^2 f(\mathbf{x}_k)$ has a positive eigenvalue, we may pick $\mathbf{p}_k$ to be in any of the eigenspaces corresponding to positive eigenvalues.

For *strictly* convex C^2 functions, however, the Newton direction will always be a descent direction.

▶ Wolfe conditions

Recall the iteration step in the line search method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

where α_k is the step size and $\mathbf{p}_k$ is the (descent) direction. Getting to $\mathbf{x}_{k+1}$ from $\mathbf{x}_k$ requires knowing which direction to take ($\mathbf{p}_k$) and knowing how much to step in that direction (α_k). We have seen two descent directions:

$$\mathbf{p}_k = -\mathbf{g}_k, \qquad \mathbf{p}_k = -H_k^{-1}\mathbf{g}_k.$$

We will discuss how to choose the step size α_k. In modern machine learning parlance, step size α_k is called the *learning rate*.

The more important and more expensive part is to find $\mathbf{p}_k$. Finding α_k is of secondary importance. For Newton's method we can in fact take $\alpha_k = 1$ for all k. But in general this will not work.

Why does the choice of α_k matter? The descent direction only guarantees $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$. This is not enough to guarantee convergence to a stationary point or optimal point. For example, if $f(\mathbf{x}_*) = -1$ and $f(\mathbf{x}_k) = \frac{1}{k}$, then $(f(\mathbf{x}_k))_{k=1}^\infty$ converges as a sequence, but the algorithm may not converge to the solution. See Figure 14 for an illustration.

Just having a descent direction is not enough and we need sufficient decrease conditions. The best known sufficient decrease conditions are the *Wolfe conditions* and these tell us how to choose α_k.

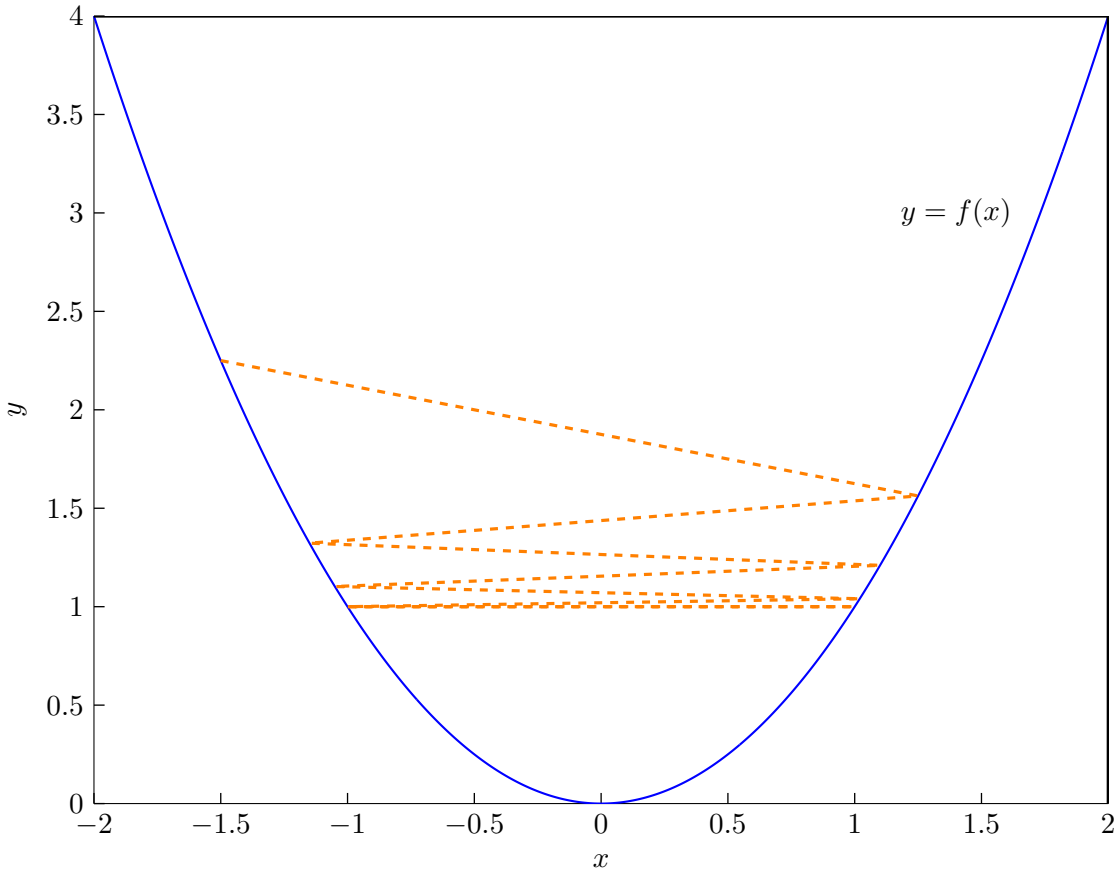


Figure 14: Value of f is reduced at every step and yet does not converge to a minimizer. The problem with is that each step achieves too little reduction relative to the step size.

Definitions 5.13 (Wolfe conditions)**.** A step size $\alpha_k > 0$ satisfies the *Wolfe conditions* if for some $0 < c_1 < c_2 < 1$, the following inequalities hold:

Armijo rule:

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla f(\mathbf{x}_k)^\mathsf{T} \mathbf{p}_k \tag{1}$$

curvature condition:

$$\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^\mathsf{T} \mathbf{p}_k \geq c_2 \nabla f(\mathbf{x}_k)^\mathsf{T} \mathbf{p}_k \tag{2}$$

The *strong Wolfe conditions* require α_k to satisfy the (1) and

$$\left|\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^\mathsf{T} \mathbf{p}_k\right| \leq c_2 \left|\nabla f(\mathbf{x}_k)^\mathsf{T} \mathbf{p}_k\right| \tag{2'}$$

The Armijo condition guarantees sufficient decrease, i.e., $f(\mathbf{x}_{k+1})$ is sufficiently smaller than $f(\mathbf{x}_k)$. The curvature condition guarantees reasonable progress, i.e., $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ is reasonably far from $\mathbf{x}_k$ — so α_k cannot be too small.

Although the Wolfe conditions seem restrictive, the following theorem gives reasonable conditions under which the Wolfe conditions are satisfied.

Theorem 5.14. *Suppose*

(i) $f \in C^1(\mathbb{R}^n)$

(ii) $\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k < 0$

(iii) $g(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ *is bounded below for* $\alpha > 0$

(iv) c_1 *and* c_2 *any constant with* $0 < c_1 < c_2 < 1$.

Then there exists $\alpha_k > 0$ *satisfying the Wolfe conditions and strong Wolfe conditions.*

Proof. The picture to keep in mind here is Figure 15. Note that $g(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ bounded below and the line $\ell(\alpha) = f(\mathbf{x}_k) + \alpha c_1 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k$ is unbounded below. Since

$$g'(0) = \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k < c_1 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k = \ell'(0) < 0,$$

g decreases faster than ℓ in a neighborhood of 0, i.e. the graph of g lies below graph of ℓ.

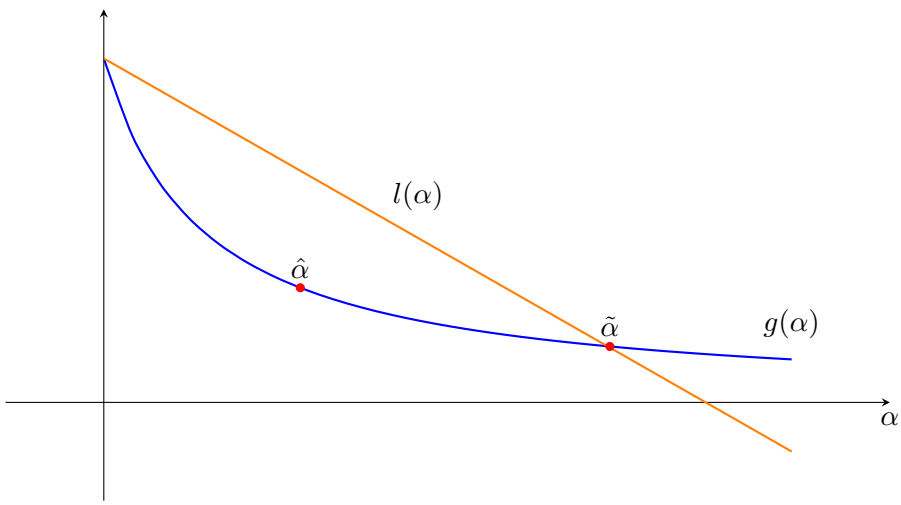


Figure 15: Simple pictorial illustration of the relation between g and *ell*.

Then there exists a smallest $\widetilde{\alpha} > 0$ such that $g(\widetilde{\alpha}) = \ell(\widetilde{\alpha})$, i.e.,

$$f(\mathbf{x}_k + \widetilde{\alpha} \mathbf{p}_k) = f(\mathbf{x}_k) + \widetilde{\alpha} c_1 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k \tag{$\star$}$$

The Armijo condition clearly holds for all $\alpha < \widetilde{\alpha}$. By the mean value theorem, there exists $\hat{\alpha} \in (0, \widetilde{\alpha})$ such that

$$g'(\hat{\alpha}) = \frac{g(\widetilde{\alpha}) - g(0)}{\widetilde{\alpha} - 0}.$$

By the chain rule,

$$f(\mathbf{x}_k + \widetilde{\alpha} \mathbf{p}_k) - f(\mathbf{x}_k) = \widetilde{\alpha} \underbrace{\nabla f(\mathbf{x}_k + \hat{\alpha} \mathbf{p}_k)^\intercal \mathbf{p}_k}_{g'(\hat{\alpha})} \tag{$\star\star$}$$

By ($\star$) and ($\star\star$) (and dividing by $\widetilde{\alpha}$),

$$\nabla f(\mathbf{x}_k + \hat{\alpha} \mathbf{p}_k)^\intercal \mathbf{p}_k = c_1 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k > c_2 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k \tag{$\star\star\star$}$$

since $c_1 < c_2$ and $\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k < 0$. Then $\hat{\alpha}$ satisfies the curvature condition (since $\hat{\alpha} < \widetilde{\alpha}$) and the Armijo condition. Note that $(\star\star\star)$ also implies the strong Wolfe condition since both sides are negative. $\square$

The curvature condition in the Wolfe conditions is

$$\nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^\intercal \mathbf{p}_k > c_2 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k,$$

whereas the curvature condition in the strong Wolfe conditions is

$$\left|\nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^\intercal \mathbf{p}_k\right| < c_2 \left|\nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k\right|.$$

Note that these are the same if

$$0 > \nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^\intercal \mathbf{p}_k > c_2 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k$$

but can be different if

$$\nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^\intercal \mathbf{p}_k > 0 > c_2 \nabla f(\mathbf{x}_k)^\intercal \mathbf{p}_k.$$

When discussing convergence of optimization algorithms, it is important to make precise *what* exactly is converging – after all, this is what makes all of these a science. Given conditions on $\mathbf{x}_0$ and f, we can try to show that the sequence $(\mathbf{x}_k)$ produced by our algorithm converges in the following senses:

(i) Stationary point: $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_*$, where $\nabla f(\mathbf{x}_*) = \mathbf{0}$;

(ii) Candidate for local minimum: $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_*$, where $\nabla f(\mathbf{x}_*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}_*) \succeq 0$;

(iii) Local minimum: $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_*$ where $\nabla f(\mathbf{x}_*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}_*) \succ 0$.

These are all too difficult to guarantee in general. Instead, we can prove weaker statements concerning the sequence of gradients and Hessians:

(i) $\lim_{k\to\infty} \nabla f(\mathbf{x}_k) = \mathbf{0}$ (or $\liminf_{k\to\infty} \nabla f(\mathbf{x}_*) = \mathbf{0}$);

(ii) $\lim_{k\to\infty} \nabla f(\mathbf{x}_k) = \mathbf{0}$ and $\lim_{k\to\infty} \nabla^2 f(\mathbf{x}_k) \succeq 0$;

(iii) $\lim_{k\to\infty} \nabla f(\mathbf{x}_k) = \mathbf{0}$ and $\lim_{k\to\infty} \nabla^2 f(\mathbf{x}_k) \succ 0$.

Items 1–3 are called *convergence*; items 4–6 are called *functional convergence*. For certain functions these are equivalent, but in general this is not true. Functional convergence is typically easier to establish.

We now show that line search methods with step size satisfying the Wolfe conditions will converge in the functional sense.

Theorem 5.15. *Suppose*

(i) $f \in C^1(\mathbb{R}^n)$,

(ii) $\nabla f \in \mathrm{Lip}_\gamma(\mathbb{R}^n)$ (*i.e.*, $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \le \gamma \|\mathbf{x} - \mathbf{y}\|$ *for all* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$),

(iii) f *is bounded below,*

(iv) $\mathbf{p}_k$ *and* α_k *satisfy Wolfe conditions:*

$$\begin{cases} f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) + c_1\alpha_k \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k \\ \nabla f(\mathbf{x}_{k+1})^\top \mathbf{p}_k \ge c_2 \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k, \end{cases}$$

(v) *there exists* δ *such that* $\cos\theta_k = -\dfrac{\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k}{\|\nabla f(\mathbf{x}_k)\|\|\mathbf{p}_k\|} \ge \delta > 0$ *for all* $k \in 0, 1, \ldots$.

[(i)] Then $\lim_{k\to\infty} \nabla f(\mathbf{x}_k) = \mathbf{0}$.

Proof. Subtracting $\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k$ from both sides of the curvature inequality,

$$[\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)]^\top \mathbf{p}_k \ge (c_2 - 1)\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k.$$

By the Lipschitz condition and the Cauchy–Schwarz inequality,

$$[\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)]^\top \mathbf{p}_k \le \|\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\|\|\mathbf{p}_k\| \le \alpha_k \gamma \|\mathbf{p}_k\|^2.$$

Combining the two inequalities, we obtain

$$\alpha_k \gamma \|\mathbf{p}_k\|^2 \ge (c_2 - 1)\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k, \qquad \Rightarrow \qquad \alpha_k \ge \frac{c_2 - 1}{\gamma} \frac{\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k}{\|\mathbf{p}_k\|^2}.$$

Substituting this inequality into the Armijo condition, noting that $\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k < 0$,

$$f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) - \frac{c_1(1 - c_2)}{\gamma} \frac{(\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k)^2}{\|\mathbf{p}_k\|^2}.$$

Letting $c = c_1(1 - c_2)/\gamma$ and $\delta_k = \cos\theta_k$, we can rewrite this as

$$f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) - c\,\delta_k^2 \|\nabla f(\mathbf{x}_k)\|^2.$$

Summing over all $j \le k$,

$$f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_0) - c \sum_{j=0}^{k} \delta_j^2 \|\nabla f(\mathbf{x}_j)\|^2$$

Since f is bounded below, there exists some $M > 0$ such that

$$f(\mathbf{x}_0) - f(\mathbf{x}_{k+1}) \le M$$

for all $k \in \mathbb{N}$, so

$$\sum_{j=0}^{k} \delta_j^2 \|\nabla f(\mathbf{x}_j)\|^2 \le \frac{M}{c}.$$

Since each term in the sum is nonnegative and the sequence of partial sums is bounded above,

$$\sum_{j=0}^{\infty} \delta_j^2 \|\nabla f(\mathbf{x}_j)\|^2$$

is convergent. It follows that

$$\lim_{k\to\infty} \delta_k^2 \|\nabla f(\mathbf{x}_k)\|^2 = 0.$$

Since $\delta_k \ge \delta > 0$

$$0 \le \delta^2 \|\nabla f(\mathbf{x}_k)\|^2 \le \delta_k^2 \|\nabla f(\mathbf{x}_k)\|^2 \to 0$$

Hence $\|\nabla f(\mathbf{x}_k)\| \to 0$. □

▶ Finding step size

The procedure for finding an appropriate step size is usually the so-called *line search*.

Notation 5.16. If $\min f(\mathbf{x}) = f(\mathbf{x}_*)$, then $\operatorname{argmin} f(\mathbf{x}) = \mathbf{x}_*$. Typically argmin is a *set* of global minimizers, but we will occasionally abuse notation.

Here, we discuss three methods to compute α_k at each iteration.

Exact line search: find the best α along the ray $\{\mathbf{x}_k + \alpha\mathbf{p}_k : \alpha > 0\}$ in the sense of

$$\alpha_k = \operatorname{argmin}_{\alpha \geq 0} f(\mathbf{x}_k + \alpha\mathbf{p}_k).$$

Note that this is a univariate minimization problem: define

$$g\colon [0, \infty) \to \mathbb{R}, \qquad g(\alpha) = f(\mathbf{x}_k + \alpha\mathbf{p}_k).$$

If we minimize $g(\alpha)$, then minimizer α_* is the step size α_k. However, there is a problem associated with solving a univariate optimization problem every step: it can be very expensive. We use exact line search only when the univariate minimization is computationally cheaper than finding $\mathbf{p}_k$ (it is better to spend energy finding the right direction).

Backtracking line search: α_k approximately so that Armijo satisfied.

Algorithm 4 Backtracking line search (BTLS)

Require: $\mathbf{p}_k \in \mathbb{R}^n,\ \mathbf{x}_k \in \mathbb{R}^n,\ c \in (0,1),\ \rho \in (0,1)$
  procedure BTLS($\mathbf{p}_k$, x_k, c, ρ)
    $\alpha \leftarrow 1$
    while $f(\mathbf{x}_k + \alpha\mathbf{p}_k) > f(\mathbf{x}_k) + c\alpha\nabla f(\mathbf{x}_k)^\mathsf{T}\mathbf{p}_k$ **do** // Armijo not satisfied
      $\alpha \leftarrow \rho\alpha$
    end while
    $\alpha_k = \alpha$ // this is the best step size
    return α_k
  end procedure

You may of course modify the 'while' condition so that you also get the curvature condition but practioners often ignore that because it is more expensive. Observe that searching for an α that satisfies the Armijo condition only requires us to evaluate $f(\mathbf{x}_k + \alpha\mathbf{p}_k)$ at various values of α; on the other hand searching for an α that satisfies the curvature condition requires that we evaluate $\nabla f(\mathbf{x}_k + \alpha\mathbf{p}_k)$ at various values of α, which is much more expensive since each gradient requires n function evaluations.

Barzilai–Borwein step size: One may avoid line searches entirely by using a fixed formula for step size:

$$\alpha_{k+1} = \frac{\left|\left(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\right)^\mathsf{T}(\mathbf{x}_{k+1} - \mathbf{x}_k)\right|}{\|\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\|^2}.$$

This has become popular recently when used in conjunction with stochastic gradient descent. We will discuss it later after we dicussed quasi-Newton methods. Note that

this is a two-step method, unlike line searches, it requires us to store two successive gradients.

With these tools, we can give the complete algorithm for steepest descent.

Algorithm 5 Steepest descent

Require: $\mathbf{x}_0 \in \mathbb{R}^n$, $\varepsilon > 0$ (small)
  procedure SDBTLS($\mathbf{x}_0$, ε)
    $k \leftarrow 0$
    repeat
      Descent direction: $\mathbf{p}_k \leftarrow -\nabla f(\mathbf{x}_k)$
      Step size: $\alpha_k \leftarrow |(\mathbf{p}_k - \mathbf{p}_{k-1})^\intercal(\mathbf{x}_k - \mathbf{x}_{k-1}|/\|\mathbf{p}_k - \mathbf{p}_{k-1}\|^2$
      Update: $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$; $\quad k \leftarrow k+1$
    until *Stopping condition*: $\|\nabla f(\mathbf{x}_k)\| < \varepsilon$.
    $\mathbf{x}_* \leftarrow \mathbf{x}_k$
    return $\mathbf{x}_*$
  end procedure

Alternatively the step size can be determined through exact or backtracking line search.

▶ Newton method for optimization

Newton method for optimizing multivariate functions is best with $\alpha_k = 1$ with one caveat — $\mathbf{x}_k$ must be near to $\mathbf{x}_*$. Because of this caveat, it is not uncommon for practioners to include an α_k found by backtracking line search. One key difference with steepest descent is that the stopping condition is evaluated after computing the Newton decrement, not after the update.

Algorithms requiring higher order derivatives, such as Newton method, are computationally expensive because one needs to evaluate $n(n+1)/2$ functions to get $\nabla^2 f(\mathbf{x}_k)$ and to a lesser extent, one also needs to solve large system of linear equations $\nabla^2 f(\mathbf{x}_k)\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ for $\mathbf{p}_k$. For this reason, optimization is typically done in a "cheap to expensive" fashion, whereby one uses cheap methods until the iteration is sufficiently close to the optimum and then invoking expensive algorithms like Newton method. In other words, the starting point $\mathbf{x}_0$ for Newton method is usually the output of some cheaper algorithm like steepest descent.

In the following we will assume that $f : \mathbb{R}^n \to \mathbb{R}$ is a strictly convex function although the method also works (with slight modification of the stopping condition) as long as f has invertible Hessian.

The stopping condition in terms of λ_k^2 seems a bit strange but it is actually entirely consistent with the stopping condition in steepest descent. Note that in steepest descent, we used the norm of the gradient

$$\|\nabla f(\mathbf{x}_k)\| = \sqrt{\nabla f(\mathbf{x}_k)^\intercal \nabla f(\mathbf{x}_k)}$$

to decide when to stop whereas in Newton method, we used the Newton decrement

$$\lambda_k = \sqrt{\nabla f(\mathbf{x}_k)^\intercal [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)}$$

to decide when to stop. When $\nabla^2 f(\mathbf{x}_k) = I$, it reduces to the norm of the gradient.

Algorithm 6 Multivariate Newton Method

Require: $\mathbf{x}_0 \in \mathbb{R}^n$, $\varepsilon > 0$ (small)
  procedure NEWTON($\mathbf{x}_0$, ε)
    $k \leftarrow 0$
    repeat
      Newton step: $\mathbf{p}_k \leftarrow -[\nabla^2 f(\mathbf{x}_k)]^{-1}\nabla f(\mathbf{x}_k)$
      Newton decrement: $\lambda_k^2 \leftarrow \nabla f(\mathbf{x}_k)^{\mathsf{T}}[\nabla^2 f(\mathbf{x}_k)]^{-1}\nabla f(\mathbf{x}_k)$
      Stopping condition: **quit** if $\lambda_k^2/2 \le \varepsilon$.
      (Optional) line search: call backtracking line search to find α_k, or just use $\alpha_k = 1$
      Update: $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$; $\quad k \leftarrow k+1$
    return $\mathbf{x}_* \coloneqq \mathbf{x}_k$
  end procedure

One very important properrty if Newton method is that it is affine invariant, i.e., if we do a change of coordinates $\mathbf{y} = A\mathbf{x}+\mathbf{b}$ and write $g(\mathbf{y}) = f(A\mathbf{x}+\mathbf{b})$, then applying Newton method to g gives iterates $\mathbf{y}_k$ that are exactly related to $\mathbf{x}_k$, the iterates of Newton method applied to f, by exactly the same relation $\mathbf{y}_k = A\mathbf{x}_k + \mathbf{b}$. Furthermore the Newton decrements have exactly the same because of chain rule. No other standard optimization method — steepest descent, Gauss–Newton, quasi-Newton, etc — has this property.

▶ Starting and stopping

Finding an initial point $\mathbf{x}_0$ can be a nontrivial task. One guideline is that you should always initialize an expensive algorithm with cheaper algorithms. Nobody would ever generate $\mathbf{x}_0$ randomly for Newton method because you would just be wasting expensive Newton iterations when you are still far from $\mathbf{x}_*$, when Newton method is most likely to become quadratically convergent. What you should do is to set the input $\mathbf{x}_0$ of Newton method to be the output $\mathbf{x}_*$ of steepest decent, i.e., you initialize Newton method with steepest descent.

Terminating your algorithm is also an art. In our pseudocodes we simply used the norm of the gradient to ascertain when to stop. Basically when the first order necessary condition is satisfied approximately, i.e., $\nabla f(\mathbf{x}_k) \approx \mathbf{0}$, we output $\mathbf{x}_* \coloneqq \mathbf{x}_k$. But realistic stopping condition usually involves more than one criterion:

(i) $|f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})| \le \varepsilon_1$,

(ii) $\|\mathbf{x}_k - \mathbf{x}_{k-1}\| \le \varepsilon_2$,

(iii) $\|\nabla f(\mathbf{x}_k)\| \le \varepsilon_3$,

with small numbers $\varepsilon_1, \varepsilon_2, \varepsilon_3 > 0$. The first two conditions basically tell you when your algorithm has stopped making progress. All three conditions must be satisfied before we output $\mathbf{x}_* \coloneqq \mathbf{x}_k$. A standard choice for $\varepsilon_1, \varepsilon_2, \varepsilon_3$, based on decades of experimentation by practitioners, is

$$\varepsilon_1 = \varepsilon(1 + |f(\mathbf{x}_k)|), \quad \varepsilon_2 = \sqrt{\varepsilon}(1 + \|\mathbf{x}_k\|), \quad \varepsilon_3 = \sqrt[3]{\varepsilon}(1 + |f(\mathbf{x}_k)|).$$

Lecture 6 (Tuesday, October 31)

▶ Newton method for systems of nonlinear equations

Let $F\colon \mathbb{R}^n \to \mathbb{R}^n$; Newton method can be used to numerically find $\mathbf{x}_* \in \mathbb{R}^n$ such that $F(\mathbf{x}_*) = \mathbf{0}$ by setting

$$\mathbf{p}_k \leftarrow -[\mathrm{J}F(\mathbf{x}_k)]^{-1}F(\mathbf{x}_k),$$

where $\mathrm{J}F(\mathbf{x}_k) \in \mathbb{R}^{n\times n}$ is the Jacobian matrix. So the Newton iteration is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathrm{J}F(\mathbf{x}_k)]^{-1}F(\mathbf{x}_k).$$

Newton method for optimization is just a special case: $F(\mathbf{x}) = \nabla f(\mathbf{x})$ where f is the objective function. Thus $\mathrm{J}F = \mathrm{J}(\nabla f) = \nabla^2 f$.

Example 6.1. Define $F\colon \mathbb{R}^2 \to \mathbb{R}^2$. by

$$F(x,y) = \begin{bmatrix} x+y-3 \\ x^2+y^2-9 \end{bmatrix}.$$

Then

$$\mathrm{J}F\colon \mathbb{R}^2 \to \mathbb{R}^{2\times 2}, \qquad \mathrm{J}F(x,y) = \begin{bmatrix} 1 & 1 \\ 2x & 2y \end{bmatrix}.$$

Suppose we start at $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$. Then $\mathbf{p}_0$ solves

$$\mathrm{J}F(x_0,y_0)\mathbf{p}_0 = -F(x_0,y_0).$$

Solving the system of equations,

$$\mathbf{p}_0 = \begin{bmatrix} -13/8 \\ -11/8 \end{bmatrix}.$$

The iteration step is

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \mathbf{p}_0 = \begin{bmatrix} -0.625 \\ 3.625 \end{bmatrix}$$

Now $\mathbf{p}_1$ solves

$$\mathrm{J}F(x_1,y_1)\mathbf{p}_1 = -F(x_1,y_1), \qquad \Rightarrow \qquad \begin{bmatrix} 1 & 1 \\ -5/4 & 29/4 \end{bmatrix}\mathbf{p}_1 = \begin{bmatrix} 0 \\ 145/32 \end{bmatrix}.$$

Hence

$$\mathbf{p}_1 = \begin{bmatrix} 145/272 \\ -145/272 \end{bmatrix},$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \mathbf{p}_1 \approx \begin{bmatrix} 0.0092 \\ 3.092 \end{bmatrix} \approx \begin{bmatrix} 0 \\ 3 \end{bmatrix}.$$

▶ Gauss–Newton method for nonlinear least squares

In a linear least squares problem, aka linear regression, we optimize

$$\min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} (\mathbf{a}_i^{\mathsf{T}}\mathbf{x} - b_i)^2 = \min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|^2$$

where $A \in \mathbb{R}^{m\times n}$ has rows $\mathbf{a}_1^{\mathsf{T}}, \dots, \mathbf{a}_m^{\mathsf{T}}$ and $\mathbf{b} \in \mathbb{R}^m$ has entries $b_1, \dots, b_m$. We saw how to reduce this to a linear system

$$A^{\mathsf{T}}A\mathbf{x} = A^{\mathsf{T}}\mathbf{b}. \tag{6.2}$$

A nonlinear least squares problem is one where we replace the *residual* $F(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$ by a nonlinear function:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} f_i(\mathbf{x})^2 = \min_{\mathbf{x}\in\mathbb{R}^n} \|F(\mathbf{x})\|^2$$

where $F : \mathbb{R}^n \to \mathbb{R}^m$ is

$$F(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix}.$$

Note that

$$F(\mathbf{x}) = \mathbf{0} \quad \Leftrightarrow \quad \min_{\mathbf{x}\in\mathbb{R}^n} \|F(\mathbf{x})\|^2 = 0$$

so solving a system of nonlinear equations is also an optimization problem, in fact, a nonlinear least squares problem. This is hardly surprising since every problem in the world is an optimization problem.

Gauss–Newton method is a special algorithm designed specifically for objective function of the form $f(\mathbf{x}) = \|F(\mathbf{x})\|^2$, i.e., for nonlinear least squares problems. If $m = n$, then the Gauss–Newton iteration is simply

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathrm{J}F(\mathbf{x}_k)^{-1}F(\mathbf{x}_k), \tag{6.3}$$

assuming that the Jacobian $\mathrm{J}F(\mathbf{x}_k) \in \mathbb{R}^{n\times n}$ is invertible.

More generally for any $m \geq n$, the Gauss–Newton iteration is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathrm{J}F(\mathbf{x}_k)^{\mathsf{T}}\mathrm{J}F(\mathbf{x}_k)]^{-1}\mathrm{J}F(\mathbf{x}_k)^{\mathsf{T}}F(\mathbf{x}_k), \tag{6.4}$$

assuming that the Jacobian $\mathrm{J}F(\mathbf{x}_k) \in \mathbb{R}^{m\times n}$ has rank n. Note that $A \in \mathbb{R}^{m\times n}$ has rank n iff $A^{\mathsf{T}}A \in \mathbb{R}^{n\times n}$ is invertible and the matrix $(A^{\mathsf{T}}A)^{-1}A^{\mathsf{T}} \in \mathbb{R}^{n\times m}$ is called the Moore–Penrose pseudoinverse of A, a sort of "inverse" for a rectangular matrix. This is very much related to (6.2). In particular, if $m = n$ and A is invertible, then $(A^{\mathsf{T}}A)^{-1}A^{\mathsf{T}} = A^{-1}$, so (6.4) reduces to (6.3).

Gauss–Newton method is an approximation of Newton method and can be derived as follows. Note that by chain rule,

$$\nabla f(\mathbf{x}) = 2\mathrm{J}F(\mathbf{x})^{\mathsf{T}}F(\mathbf{x}) \qquad \text{and} \qquad \nabla^2 f(\mathbf{x}) \approx 2\mathrm{J}F(\mathbf{x})^{\mathsf{T}}\mathrm{J}F(\mathbf{x}) \tag{6.5}$$

where the approximation is obtained by dropping the term in red:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = 2\sum_{k=1}^{m}\left(\frac{\partial f_k}{\partial x_i}\frac{\partial f_k}{\partial x_j} + f_k \frac{\partial^2 f_k}{\partial x_i \partial x_j}\right),$$

which is valid if the term in red is small (e.g., when f_k is nearly linear). Now observe that by (6.5), the Newton direction

$$\nabla^2 f(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k) \approx [\mathrm{J}F(\mathbf{x}_k)^{\mathsf{T}}\mathrm{J}F(\mathbf{x}_k)]^{-1}\mathrm{J}F(\mathbf{x}_k)^{\mathsf{T}}F(\mathbf{x}_k)$$

which gives us (6.4).

▶ Stochastic gradient descent

As we saw in the last lecture, many optimization problems in machine learning take a form like

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} f_i(\mathbf{x})$$

where m is the size of the *sample* or *training set* or *observations* (huge in modern applications) and n is the number of *parameters* (also huge in modern applications) to be estimated from the observations. Take a simple binary classification problem into two classes represented by -1 and $+1$ respectively. Examples include regularized least squares problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{m}\sum_{i=1}^{m}(b_i - \mathbf{a}_i^{\mathsf{T}}\mathbf{x})^2 + \frac{\lambda}{2}\|\mathbf{x}\|^2;$$

regularized binary logistic regression:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{m}\sum_{i=1}^{m}\log\bigl(1 + e^{-b_i\mathbf{a}_i^{\mathsf{T}}\mathbf{x}}\bigr) + \frac{\lambda}{2}\|\mathbf{x}\|^2;$$

regularized squared hinge loss SVM:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{m}\sum_{i=1}^{m}\max(0, 1 - b_i\mathbf{a}_i^{\mathsf{T}}\mathbf{x})^2 + \frac{\lambda}{2}\|\mathbf{x}\|_2^2;$$

nonconvex SVM with a sigmoid loss:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{m}\sum_{i=1}^{m}(1 - \tanh(b_i\mathbf{a}_i^{\mathsf{T}}\mathbf{x})) + \frac{\lambda}{2}\|\mathbf{x}\|_2^2;$$

and many more. Recall that $\lambda > 0$ is a *regularization parameter*, $b_1, \dots, b_m \in \{-1, +1\}$ are *training labels*, and $\mathbf{a}_1, \dots, \mathbf{a}_m \in \mathbb{R}^n$ are *feature vectors* — all these values are given but m is prohibitively large in modern machine learning applications.

Stochastic gradient descent is the simple procedure where the usual updates

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\nabla f(\mathbf{x}_k) = \mathbf{x}_k - \eta_k\left[\frac{1}{m}\sum_{i=1}^{m}\nabla f_i(\mathbf{x}_k)\right], \qquad \alpha_k = \eta_k/m,$$

is replaced by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \left[\frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(\mathbf{x}_k) \right]$$

where $S_k \subseteq \{1, \dots, m\}$ is a small randomly chosen subset called a *minibatch*. In this context, the stepsize η_k is often called the *learning rate*. In usual steepest descent, we want

$$\lim_{k \to \infty} \nabla f(\mathbf{x}_k) = \mathbf{0}.$$

In stochastic gradient descent, noting that $\nabla f(\mathbf{x}_k)$ is now a random variable, we want

$$\lim_{k \to \infty} \mathbb{E}[\nabla f(\mathbf{x}_k)] = \mathbf{0}.$$

▶ Quasi-Newton methods

Recall that the steepest descent algorithm has search direction $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ and gives linear convergence. Newton method has search direction $\mathbf{p}_k = -\nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$ but requires evaluation of a Hessian ($O(n^2)$ evaluations) and matrix inversion at each iteration, both of which are costly. Quasi-Newton methods are an "in-between" alternative to Newton method, requiring fewer function evaluations and supplying superlinear convergence.

Quasi-Newton methods use an approximation H_k to the Hessian $\nabla^2 f(\mathbf{x}_k)$; however, instead of requiring

$$\|H_k - \nabla^2 f(\mathbf{x}_k)\| \to 0,$$

it is enough to have

$$\|H_k^{-1} \nabla f(\mathbf{x}_k) - \nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)\| \to 0.$$

We do not care about the actual value of the Hessian since we just need the product, i.e., just need the search direction to be close. We discuss two methods of computing H_k.

Quasi-Newton methods can also be randomized in the same way steepest descent can be randomized into SGD. One of the most popular randomized quasi-Newton method is the so-called SLBFGS — stochastic limited-memory BFGS, which we will see below.

▶ Davidon–Fletcher–Powell

The H_k are defined inductively. Suppose H_k has been computed — we set H_{k+1} to be the minimizer of the following problem:

$$\begin{aligned} \text{minimize} \quad & \|H - H_k\| \\ \text{subject to} \quad & H = H^\mathsf{T}, \quad \underbrace{H\mathbf{s}_k = \mathbf{y}_k}_{\text{secant equation}}, \end{aligned} \tag{$\star$}$$

where $s_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$. Intuitively, the solution to the secant equation can be thought of as

$$H_k = \frac{\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)}{\mathbf{x}_{k+1} - \mathbf{x}_k} \approx \nabla^2 f(\mathbf{x}_{k+1}).$$

This, of course, is not allowed since the numerator and denominator are vectors.

Ideally, we would like

(i) H_{k+1} to be symmetric

(ii) H_{k+1} to be positive definite

(iii) H_{k+1} to satisfy the secant equation.

The solution to ($\star$) is given by the Davidon–Fletcher–Powell (DFP) update:

$$H_{k+1} = (I - \rho_k \mathbf{y}_k \mathbf{s}_k^\top) H_k (I - \rho_k \mathbf{s}_k \mathbf{y}_k^\top) + \rho_k \mathbf{y}_k \mathbf{y}_k^\top,$$

where $\rho_k = 1/(\mathbf{y}_k^\top \mathbf{s}_k)$.

The DFP updates for the inverse Hessian $B_k := H_k^{-1}$ can be computed efficiently using what's called the Sherman–Woodbury–Morisson formula:

$$B_{k+1} = B_k - \frac{B_k \mathbf{y}_k \mathbf{y}_k^\top B_k}{\mathbf{y}_k^\top B_k \mathbf{y}_k} + \frac{\mathbf{s}_k \mathbf{s}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k}.$$

Note that positive definiteness of H_k (resp. B_k) implies the positive definiteness of H_{k+1} (resp. B_{k+1}) is positive definite. In particular, if we choose H_0 or B_0 to be positive definite, all subsequent H_k and B_k will be positive definite. The most common choices are $B_0 = I$ or $H_0 = I$, though one can also choose $H_0 = \nabla^2 f(\mathbf{x}_0)$ (provided the Hessian is positive definite) or $H_0 = \operatorname{diag}(\nabla^2 f(\mathbf{x}_0))$, with sign changes to ensure positive definiteness.

Algorithm 7 Quasi-Newton using DFP updates

Require: starting point $\mathbf{x}_0 \in \mathbb{R}^n$, tolerance $\varepsilon > 0$, inverse Hessian approximation $B_0 \succ 0$

procedure DFP(x_0, ε, B_0)
  $k \leftarrow 0$
  while $\|\nabla f(\mathbf{x}_k)\| > \varepsilon$ **do**
    Quasi-Newton step: $\mathbf{p}_k \leftarrow -B_k \nabla f(\mathbf{x}_k)$
    Line search: call backtracking line search to find suitable α_k
    Update: $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    Update inverse Hessian:

$$\begin{aligned} \mathbf{s}_k &\leftarrow \mathbf{x}_{k+1} - \mathbf{x}_k \\ \mathbf{y}_k &\leftarrow \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \\ B_{k+1} &\leftarrow B_k - \frac{B_k \mathbf{y}_k \mathbf{y}_k^\top B_k}{\mathbf{y}_k^\top B_k \mathbf{y}_k} + \frac{\mathbf{s}_k \mathbf{s}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k} \end{aligned}$$

    $k \leftarrow k + 1$
  end while
  return $\mathbf{x}_* := \mathbf{x}_k$
end procedure

▶ Broyden–Fletcher–Goldfarb–Shanno

Recall that our goal is to approximate the inverse Hessian $B_k \approx \nabla^2 f(\mathbf{x}_k)^{-1}$. Instead of approximating the Hessian and inverting, we could have started the other way around by finding a formula for the approximate inverse Hessian. If B_k has been computed, we set B_{k+1} to be the minimizer of the following problem:

$$\begin{aligned} \text{minimize} \quad & \|B - B_k\| \\ \text{subject to} \quad & B = B^{\mathsf{T}}, \quad B\mathbf{y}_k = \mathbf{s}_k. \end{aligned} \tag{$\star\star$}$$

Note that this is the dual of ($\star$): the equation $B\mathbf{y}_k = \mathbf{s}_k$ is solved by the inverse to the solution to the secant equation $H\mathbf{s}_k = \mathbf{y}_k$. The solution to ($\star\star$) is given by the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update:

$$B_{k+1} = (I - \rho_k \mathbf{s}_k \mathbf{y}_k^{\mathsf{T}}) B_k (I - \rho_k \mathbf{y}_k \mathbf{s}_k^{\mathsf{T}}) + \rho_k \mathbf{s}_k \mathbf{s}_k^{\mathsf{T}}$$

where $\rho_k = 1/(\mathbf{y}_k^{\mathsf{T}} \mathbf{s}_k)$. If desired, the approximate Hessian $H_k := B_k^{-1}$ can be computed with Sherman–Morisson:

$$H_{k+1} = H_k - \frac{H_k \mathbf{s}_k \mathbf{s}_k^{\mathsf{T}} H_k}{\mathbf{s}_k^{\mathsf{T}} H_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^{\mathsf{T}}}{\mathbf{y}_k^{\mathsf{T}} \mathbf{s}_k}.$$

Both BFGS and DFP are special cases of *Broyden class updates*:

$$H_{k+1} = (1 - \phi_k) H_{k+1}^{\mathrm{BFGS}} + \phi_k H_{k+1}^{\mathrm{DFP}}.$$

In practice, the matrices H_k or B_k are never actually formed in quasi-Newton methods. Instead one simply stores the vectors $\mathbf{y}_1, \dots, \mathbf{y}_k$; $\mathbf{s}_1, \dots, \mathbf{s}_k$; $\mathbf{x}_1, \dots, \mathbf{x}_k$. The reason is that forming these matrices can take up too much storage. Nevertheless as k gets larger, the number of vectors we need to store gets larger too. The so-called *limited memory* quasi-Newton method simply drops the earlier vectors and just use the recently computed ones. For example, we can just store the most recent three vectors $\mathbf{y}_{k-2}, \mathbf{y}_{k-1}, \mathbf{y}_k$; $\mathbf{s}_{k-2}, \mathbf{s}_{k-1}, \mathbf{s}_k$; $\mathbf{x}_{k-2}, \mathbf{x}_{k-1}, \mathbf{x}_k$ and and form our approximate Hessian or inverse Hessian using these, as if we are at the third step of our algorithm. If we use BFGS updates, then the limited-memory variant is called LBFGS. If we randomize this in the same way as we randomized steepest descent, then we obtain the SLBFGS algorithm mentioned earlier.

▶ Barzilai–Borwein step size

The key idea behind the Barzilai–Borwein step size described earlier is from quasi-Newton method. Here we assume that the approximate Hessian B_k takes the form $B_k = (1/\alpha_k)I$ for some scalar $\alpha_k > 0$. So we solve

$$\min_{\alpha > 0} \left\| \frac{1}{\alpha} \mathbf{y}_k - \mathbf{s}_k \right\|^2$$

which gives us

$$\alpha_{k+1} = \frac{\mathbf{y}_k^{\mathsf{T}} \mathbf{s}_k}{\mathbf{y}_k^{\mathsf{T}} \mathbf{y}_k} = \frac{\left| \left(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \right)^{\mathsf{T}} (\mathbf{x}_{k+1} - \mathbf{x}_k) \right|}{\|\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\|^2},$$

the Barzilai–Borwein step size defined earlier.

Note that the Barzilai–Borwein step size and quasi-Newton methods are all *two-step methods*, unlike steepest descent and Newton, which only involves the gradient (or Hessian) at step k, they require that we use the gradients at two successive steps since we need both to compute

$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

We now move on to the next topic:

$$\begin{tikzcd}
\text{univariate} \arrow[r] & \text{multivariate unconstrained} \arrow[d] & \\
& \boxed{\text{multivariate equality constrained}} \arrow[d] & \\
\text{convex} & \text{multivariate inequality constrained} \arrow[l] \arrow[r] \arrow[d] & \text{nonsmooth} \\
& \begin{array}{c}\text{multiobjective, combinatorial,}\\ \text{dynamic, stochastic, integer, etc}\end{array} &
\end{tikzcd}$$

▶ Constrained optimization

The most common optimization problems come with constraints. The additional requirements on a program from the inclusion of constraints can make a big difference in the algorithm choice, runtime, and solvability of a problem. We illustrate the distinction in the following two examples.

Example 6.6 (Unconstrained quadratic programming)**.** Let $A \in \mathbb{S}^n$, $\mathbf{b} \in \mathbb{R}^n$, $c \in \mathbb{R}$. In Homework 4, we showed that

$$\min_{\mathbf{x}\in\mathbb{R}^n} \ \frac{1}{2}\mathbf{x}^\mathsf{T} A\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x} + c$$

has a unique solution iff $A \succeq 0$, and that a closed-form solution was available in that case. □

Example 6.7 (Linearly constrained quadratic programming)**.** Let $A, \mathbf{b}, c$ be as above, and let $\mathbf{h} \in \mathbb{R}^m$ and $G \in \mathbb{R}^{m\times n}$. The problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^\mathsf{T} A\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x} + c \\ \text{subject to} \quad & G\mathbf{x} \le \mathbf{h}. \end{aligned}$$

is NP-hard unless A is positive semidefinite. An example of an NP-hard problem that can be written in this form is the Facebook friendship clique problem. If A is positive semidefinite, then the problem can be solved in polynomial time, though there is no closed-form solution. Another important special case is when $A = 0$ and which case we obtain *linear programming* or LP

$$\begin{aligned} \text{minimize} \quad & \mathbf{b}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & G\mathbf{x} \le \mathbf{h}. \end{aligned}$$

□

Although constrained optimization appears easier to solve because of the smaller domain, the above examples illustrate the difficulties associated with adding constraints. We now formally introduce equality constrained optimization and inequality constrained optimization problems.

Definition 6.8 (Constrained problem)**.** Let $f, g_i, h_i \colon \mathbb{R}^n \to \mathbb{R}$. A problem of the form

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) = 0, \qquad i = 1, \ldots, m \end{aligned}$$

is called an *equality constrained optimization problem.* A problem of the form

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \begin{aligned} g_i(\mathbf{x}) &= 0, \qquad i = 1, \ldots, m, \\ h_i(\mathbf{x}) &\leq 0, \qquad i = 1, \ldots, p. \end{aligned} \end{aligned}$$

is called an *inequality constrained optimization problem.*

Note that

$$g_i(\mathbf{x}) = 0 \qquad \Leftrightarrow \qquad -g_i(\mathbf{x}) \leq 0 \text{ and } g_i(\mathbf{x}) \leq 0,$$

so every constrained problem can be written as an inequality constrained problem.

There is a special case of an inequality constrained optimization problem where the inequality constraints are all nonnegativity constraints on the variables x_i's, for example,

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \begin{aligned} g_i(\mathbf{x}) &= 0, \qquad i = 1, \ldots, m, \\ \mathbf{x} &\geq \mathbf{0}, \end{aligned} \end{aligned} \tag{6.9}$$

i.e., with $h_i(\mathbf{x}) = x_i$. In this special case, the problem essentially behaves like an equality constrained optimization problem for reasons that we will explain later. We don't need all variables to be nonnegative — for example we may have just one constraint $x_1 \geq 0$ and the other variables $x_2, \ldots, x_n$ can take any real value. We will call any such problem an *equality constrained optimization problem with nonnegative variables.*

▶ Equality constrained optimization

We start by discussing the equality constrained optimization problem and generalize it. How do we deal with constraints? The standard approach is the *method of Lagrange multipliers.*

Define $G \colon \mathbb{R}^n \to \mathbb{R}^m$ by

$$G(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_m(\mathbf{x}) \end{bmatrix}.$$

Then the equality constrained optimization problem can be written as

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & G(\mathbf{x}) = \mathbf{0}. \end{aligned} \tag{$\star$}$$

We assume $m \leq n$ or otherwise we may have $\{\mathbf{x} \in \mathbb{R}^n : G(\mathbf{x}) = \mathbf{0}\} = \varnothing$ (overdetermined systems of equations). Recall that the Jacobian $\mathrm{J}G(\mathbf{x})$ is an $m \times n$ matrix for any $\mathbf{x} \in \mathbb{R}^n$.

Definition 6.10. The function defined by

$$L(\mathbf{x}, \boldsymbol{\lambda}) \coloneqq f(\mathbf{x}) + \boldsymbol{\lambda}^{\mathsf{T}} G(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x})$$

is called the *Lagrangian* of the optimization problem $(\star)$. The variables:

$$\mathbf{x} = (x_1, \ldots, x_n)$$

are called *primal variables* and the variables:

$$\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)$$

are called *dual variables*. An alternative old fashioned name for dual variables is *Lagrange multipliers*.

Just like the case of unconstrained optimization, we are going to need optimality conditions in order to design algorithms.

Theorem 6.11 (First order necessary condition)**.** *Let $f, g_1, \ldots, g_m \in C^1(\mathbb{R}^n)$. If $\mathbf{x}_*$ is a local minimizer of $(\star)$ and $\operatorname{rank}\big(\mathrm{J}G(\mathbf{x}_*)\big) = m$, then there exist $\lambda_1^*, \ldots, \lambda_m^*$ such that*

$$\nabla f(\mathbf{x}_*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\mathbf{x}_*) = \mathbf{0},$$

or, alternatively,

$$\nabla f(\mathbf{x}_*) + \mathrm{J}G(\mathbf{x}_*)^{\mathsf{T}} \boldsymbol{\lambda}_* = \mathbf{0},$$

or, in terms of the Lagrangian,

$$\nabla L(\mathbf{x}_*, \boldsymbol{\lambda}_*) = \mathbf{0}. \tag{6.12}$$

Proof. The proof relies on the implicit function theorem but it is more important to get an idea of why it is true. Consider a problem where there is only one constraint as depicted in Figure 16:

$$\begin{aligned} \text{minimize} \quad & f(x, y) \\ \text{subject to} \quad & h(x, y) = c. \end{aligned} \tag{6.13}$$

So $g(x, y) = h(x, y) - c$ and $\nabla g(x, y) = \nabla h(x, y)$. Suppose the minimum value of (6.13) is c_3 and the minimizer is (x_*, y_*). By examining at the contour of f, we see that at the minimizer, the curve $g(x, y) = 0$ or equivalently $h(x, y) = c$ must touch the level curve $f(x, y) = c_3$ *tangentially*. Otherwise if the curve $h(x, y) = c$ cuts across (the formal term is *transversally*) the level curve, for example, like what it does to $f(x, y) = c_5$, then one can still move the point along $h(x, y) = c$ in the direction of $\nabla f_{/\!/}(x, y)$ so as to further decrease the value of f. The statement that "$h(x, y) = c$ must touch the level curve $f(x, y) = c_3$ tangentially at the minimizer (x_*, y_*)" is equivalently to the statement "$\nabla f(x_*, y_*)$ and $\nabla h(x_*, y_*)$ are parallel"

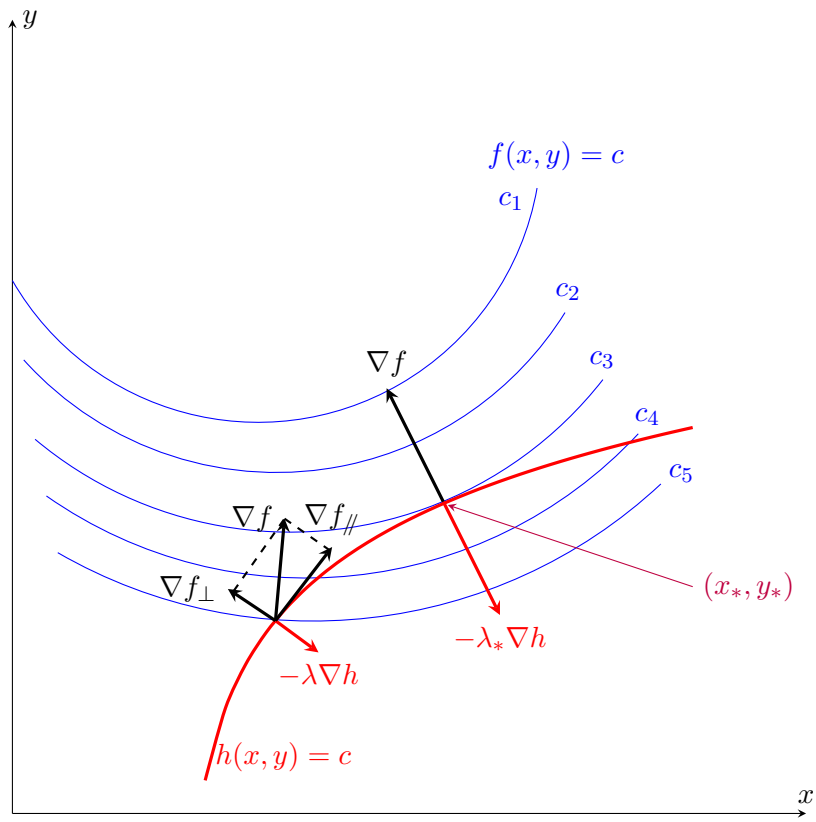


Figure 16: Pictorial proof of the first order condition.

or "$\nabla f(x_*, y_*) = -\lambda_* \nabla h(x_*, y_*)$ for some $\lambda_* \in \mathbb{R}$." Since $\nabla g(x_*, y_*) = \nabla h(x_*, y_*)$, we obtain the first order condition

$$\nabla f(x_*, y_*) + \lambda_* \nabla g(x_*, y_*) = 0.$$

Note that our discussion assumes that $\nabla h(x_*, y_*) \neq 0$ but this is guaranteed by

$$\operatorname{rank} \nabla g(x_*, y_*) = \operatorname{rank} \mathrm{J}g(x_*, y_*) = m = 1. \qquad \square$$

The same argument in the proof works for maximization but in this case Figure 16 should be modified to have $\lambda \nabla h$ pointing in same direction as $\nabla f_\perp$ and $\lambda_* \nabla h$ pointing in the same direction as ∇f. The gradients in the first order condition (as well as those in the second order conditions below) are all with respect to $\mathbf{x}$. Note that the gradient with respect to $\boldsymbol{\lambda}$ simply gives us back the constraint $G(\mathbf{x}) = \mathbf{0}$ and adds no information.

In $(\star)$, the point $\boldsymbol{\lambda}_* = (\lambda_1^*, \ldots, \lambda_m^*) \in \mathbb{R}^m$ is called *dual stationary point* or some old-fashioned people would call $\lambda_1^*, \ldots, \lambda_n^*$ the *critical Lagrange multipliers*. What is the meaning of these? There are various interpretations; one of the most intuitive comes from the perspective of economics, which we will see below when we discuss applications. To emphasize the difference, the stationary point $\mathbf{x}_* \in \mathbb{R}^n$ is called the *primal stationary point*.

When it works, the first order necessary condition is straightforward to apply — you have probably seen plenty of basic examples in your calculus class. We will see some examples

when we discuss applications later and in Homework 3. Here it is more important to see examples where it fails, i.e., you should not apply the first order necessary condition blindly.

Example 6.14 (Not sufficient)**.** Consider the bivariate optimization problem (note that when we write 'optimize' that means we want both maximizers and minimizers):

$$\begin{aligned} \text{optimize} \quad & x^3 + y^3 \\ \text{subject to} \quad & x - y = 0. \end{aligned}$$

So (6.12) is

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \nabla L(x, y, \lambda) = \begin{bmatrix} 3x^2 + \lambda \\ 3y^2 - \lambda \end{bmatrix}$$

and clearly $(x_*, y_*, \lambda_*) = (0, 0, 0)$ satisfies it. Furthermore

$$\mathrm{J}g(x, y) = [1, -1]$$

and so has rank one for any (x, y) including $(x_*, y_*) = (0, 0)$ so first order condition applies. But $(x_*, y_*) = (0, 0)$ is neither a local minimizer nor maximizer since for any $c > 0$,

$$f(-c, -c) = -2c^3 < 0 = f(x_*, y_*), \qquad f(c, c) = 2c^3 > 0 = f(x_*, y_*)$$

and $(-c, -c)$ and $(c, c,)$ clearly satisfies the constraint $x - y = 0$. The reason is that the first order condition is necessary but not sufficient. It gives you all *candidates* for local minimizers and local maximizers but it can include points that are neither. □

Unlike the unconstrained case, note that we have an additional condition on the rank of $\mathrm{J}G(\mathbf{x}_*)$.

Example 6.15 (Rank condition cannot be omitted)**.** The bivariate minimization problem

$$\begin{aligned} \text{minimize} \quad & y \\ \text{subject to} \quad & y^3 - x^2 = 0 \end{aligned}$$

has global minimizer $(x_*, y_*) = (0, 0)$ since $y^3 = x^2 \geq 0$ means that we must have $y \geq 0$ and so the the minimum is attained at $y_* = 0$, which implies that $x_* = 0$. A global minimizer is always a local minimizer. But if we simply apply $\nabla L(x_*, y_*, \lambda_*) = \mathbf{0}$, we get

$$-2\lambda x = 0, \qquad 1 + 3\lambda y^2 = 0, \qquad -x^2 + y^3 = 0.$$

The first equation implies that either $\lambda = 0$ or $x = 0$. If $\lambda = 0$, it contradicts the second equation. If $x = 0$, then thrid equation implies that $y = 0$, which again contradicts the second equation. So $\nabla L(x_*, y_*, \lambda_*) = \mathbf{0}$ has no solution. What went wrong? Note that

$$\mathrm{J}G(x, y) = \left[\frac{\partial}{\partial x}(y^3 - x^2), \frac{\partial}{\partial y}(y^3 - x^2)\right] = [-2x, 3y^2]$$

and so $\mathrm{J}G(x_*, y_*) = (0, 0)$, i.e., $\operatorname{rank}\big(\mathrm{J}G(x_*, y_*)\big) = 0 < 1 = m$. □

In Homework 3, we will see more examples where the first order condition fails.

We did not discuss second order conditions in the lecture because it requires a bit more linear algebra — recall that the nullspace or kernel of a matrix $A \in \mathbb{R}^{m\times n}$ is

$$\ker(A) \coloneqq \{\mathbf{z} \in \mathbb{R}^n : A\mathbf{z} = \mathbf{0}\}.$$

Theorem 6.16 (Second order necessary condition)**.** *Let $f, g_1, \dots, g_m \in C^2(\mathbb{R}^n)$ and $\mathbf{x}_* \in \mathbb{R}^n$ be such that* $\operatorname{rank}\big(\mathrm{J}G(\mathbf{x}_*)\big) = m$*. If $\mathbf{x}_*$ is a local minimizer of $(\star)$, then there exists $\boldsymbol{\lambda}_* \in \mathbb{R}^m$ such that*

$$\nabla L(\mathbf{x}_*, \boldsymbol{\lambda}_*) = \mathbf{0}$$

and

$$\mathbf{z}^{\mathsf{T}}\nabla^2 L(\mathbf{x}_*, \boldsymbol{\lambda}_*)\mathbf{z} \geq 0 \quad \textit{for all } \mathbf{z} \in \ker\big(\mathrm{J}G(\mathbf{x}_*)\big).$$

The conditions discussed earlier were all necessary conditions. Here is a sufficient condition.

Theorem 6.17 (Second order sufficient condition)**.** *Let $f, g_1, \dots, g_m \in C^2(\mathbb{R}^n)$ and $\mathbf{x}_* \in \mathbb{R}^n$ be such that* $\operatorname{rank}\big(\mathrm{J}G(\mathbf{x}_*)\big) = m$*. If there exists $\boldsymbol{\lambda}_* \in \mathbb{R}^m$ such that*

$$\nabla L(\mathbf{x}_*, \boldsymbol{\lambda}_*) = \mathbf{0}$$

and

$$\mathbf{z}^{\mathsf{T}}\nabla^2 L(\mathbf{x}_*, \boldsymbol{\lambda}_*)\mathbf{z} > 0 \quad \textit{for all } \mathbf{0} \neq \mathbf{z} \in \ker\big(\mathrm{J}G(\mathbf{x}_*)\big),$$

then $\mathbf{x}_*$ is a strict local minimizer of $(\star)$.

Earlier when we said that the equality constrained optimization problem with nonnegative variables (6.9) behaves like a equality constrained optimization problem, we meant that Definition 6.10 and Theorems 6.11, 6.16, 6.17 apply verbatim to (6.9), i.e., the nonnegative constraints $x_i \geq 0$ do not require additional Lagrange multipliers (general inequality constraints would require them, as we will see in the next lecture) and do not appear in the Lagrangian or optimality conditions. We will see why in the next lecture. The bottom line is nonnegative variables in an optimization problem do not need to be treated as nonnegative constraints.

Lecture 7 (Tuesday, November 7)

▶ Methods for equality constrained optimization

For simplicity, we will write an equality constrained optimization problem in the form

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & G(\mathbf{x}) = \mathbf{0}, \end{aligned}$$

where $G : \mathbb{R}^n \to \mathbb{R}^m$ is the function $G(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))$.

The most widely used strategy for constrained optimization is to convert the problem to an unconstrained optimization problem and then use one of the algorithms for the latter: steepest descent, quasi-Newton, Newton, etc. More specifically, we find an F or a sequence of F_k's depending on f and G (or H) such that the *unconstrained* minimizer of F equals the desired constrained minimizer or that the limit of the sequence *unconstrained* minimizers of F_k converges to the desired constrained minimizer.

We will start with the method of *penalty function*. Note that this depend on the optimization problem, i.e., on f and G.

Quadratic penalty function

The *quadratic penalty function* or l^2-penalty function is

$$P_2(\mathbf{x}, \rho) = f(\mathbf{x}) + \frac{\rho}{2} G(\mathbf{x})^{\mathsf{T}} G(\mathbf{x}) = f(\mathbf{x}) + \frac{\rho}{2} \|G(\mathbf{x})\|_2^2 = f(\mathbf{x}) + \frac{\rho}{2} \sum_{i=1}^{m} g_i(\mathbf{x})^2$$

where $\rho \geq 0$. Note that if any of the $g_i(\mathbf{x}) \neq 0$, then $P_2(\mathbf{x}, \rho)$ blows up when $\rho \to +\infty$. So any *minimization* algorithm would avoid this by making sure that $\mathbf{x}$ is such that $g_i(\mathbf{x}) = 0$ for all $i = 1, \dots, m$, i.e., $G(\mathbf{x}) = \mathbf{0}$. This can be formalized and rigorously established.

Theorem 7.1. *Let*

$$\mathbf{x}_*(\rho) = \operatorname{argmin} P_2(\mathbf{x}, \rho) \quad \textit{and} \quad \mathbf{x}_* = \operatorname{argmin}\{f(\mathbf{x}) : G(\mathbf{x}) = \mathbf{0}\}.$$

Under reasonably general conditions,

$$\lim_{\rho \to +\infty} \mathbf{x}_*(\rho) = \mathbf{x}_*.$$

Example 7.2. The univariate optimization problem

$$\begin{aligned} \text{minimize} \quad & x^3 \\ \text{subject to} \quad & x - 1 = 0, \end{aligned}$$

has obvious minimizer $x_* = 1$. The quadratic penalty function

$$P_2(x, \rho) = x^3 + \frac{\rho}{2}(x-1)^2$$

has unconstrained minimizer (exercise)

$$x_*(\rho) = \frac{-\rho + \sqrt{\rho^2 + 12\rho}}{6}.$$

We see that (another exercise)

$$\lim_{\rho \to \infty} x_*(\rho) = 1 = x_*. \qquad \square$$

We will discuss two more methods via *penalty function* or *augmented Lagrangian*. These depend on the optimization problem, i.e., on f and G. We have already discussed quadratic penalty function in the last lecture.

Absolute value penalty function

An issue with the quadratic penalty function is that we need $\rho \to \infty$ in order to guarantee that we obtain $\mathbf{x}_*$. The *absolute value penalty function* or l^1-penalty function, defined by

$$P_1(\mathbf{x}, \rho) = f(\mathbf{x}) + \rho \|G(\mathbf{x})\|_1 = f(\mathbf{x}) + \rho \sum_{i=1}^{m} |g_i(\mathbf{x})|$$

where $\rho \geq 0$, does not have this issue — ρ need not be arbitrarily large for $\mathbf{x}_*(\rho)$ to become $\mathbf{x}_*$. However it suffers from the problem of not being differentiable in general — requiring us to use nondifferentiable optimization algorithms. Roughly speaking, these are algorithms that use subgradient in place of gradient.

Theorem 7.3. *Let*

$$\mathbf{x}_*(\rho) = \operatorname{argmin} P_1(\mathbf{x}, \rho) \quad \textit{and} \quad \mathbf{x}_* = \operatorname{argmin}\{f(\mathbf{x}) : G(\mathbf{x}) = \mathbf{0}\}.$$

Under mild conditions, there exists $\bar{\rho} > 0$ such that

$$\mathbf{x}_*(\rho) = \mathbf{x}_*$$

whenever $\rho > \bar{\rho}$.

Example 7.4. The univariate optimization problem

$$\begin{aligned} \text{minimize} \quad & x^2 \\ \text{subject to} \quad & x - 1 = 0, \end{aligned}$$

has obvious minimizer $x_* = 1$ and critical Lagrange multiplier $\lambda_* = -2$. The l^1-penalty function

$$P_1(x, \rho) = x^2 + \rho |x - 1|$$

has unconstrained minimizer $x_*(\rho) = 1$ if $\rho > 2 =: \bar{\rho}$.

There are also variants of the penalty function method where each constraint g_i is given its own penalty parameter ρ_i:

$$\begin{aligned} P_1(\mathbf{x}, \boldsymbol{\rho}) &= f(\mathbf{x}) + \sum_{i=1}^{m} \rho_i |g_i(\mathbf{x})|, \\ P_2(\mathbf{x}, \boldsymbol{\rho}) &= f(\mathbf{x}) + \frac{\rho}{2} \|G(\mathbf{x})\|_2^2 = f(\mathbf{x}) + \frac{1}{2} \sum_{i=1}^{m} \rho_i g_i(\mathbf{x})^2, \end{aligned}$$

allowing each $\rho_i \to +\infty$ at different rates.

Augmented Lagrangian

For an equality constrained optimization problem, the Lagrangian takes the form

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^\intercal G(\mathbf{x}).$$

When $\boldsymbol{\lambda} = \boldsymbol{\lambda}_*$, the critical Lagrange multiplier, the stationarity condition in KKT conditions says that

$$\nabla L(\mathbf{x}_*, \boldsymbol{\lambda}_*) = \mathbf{0},$$

i.e., $\mathbf{x}_*$ is a *stationary point* of $L(\mathbf{x}, \boldsymbol{\lambda}_*)$. However,

$$\nabla^2 L(\mathbf{x}_*, \boldsymbol{\lambda}_*) = \nabla^2 f(\mathbf{x}_*) + \sum_{i=1}^m \lambda_i^* \nabla^2 g_i(\mathbf{x}_*)$$

may not be positive semidefinite. So $\mathbf{x}_*$ is generally *not a minimizer* of $L(\mathbf{x}, \boldsymbol{\lambda}_*)$.

However, if we add a quadratic penalty term to $L(\mathbf{x}, \boldsymbol{\lambda})$, we obtain the *augmented Lagragian*

$$L_A(\mathbf{x}, \boldsymbol{\lambda}, \rho) = f(\mathbf{x}) + \boldsymbol{\lambda}^\intercal G(\mathbf{x}) + \frac{\rho}{2} G(\mathbf{x})^\intercal G(\mathbf{x})$$

where $\rho \geq 0$. The quadratic penalty term preserves the stationarity of $\mathbf{x}_*$ as

$$\nabla L_A(\mathbf{x}_*, \boldsymbol{\lambda}_*, \rho) = \nabla L(\mathbf{x}_*, \boldsymbol{\lambda}_*) + \rho \mathrm{J}G(\mathbf{x}_*)^\intercal G(\mathbf{x}_*) = 0$$

by the stationarity ($\nabla L(\mathbf{x}_*, \boldsymbol{\lambda}_*) = \mathbf{0}$) and primal feasibility ($G(\mathbf{x}_*) = \mathbf{0}$) KKT conditions, i.e., $(\mathbf{x}_*, \boldsymbol{\lambda}_*, \rho)$ is a stationary point of L_A for any $\rho \geq 0$. As

$$\nabla^2 L_A(\mathbf{x}, \boldsymbol{\lambda}, \rho) = \nabla^2 f(\mathbf{x}) + \sum_{i=1}^m (\lambda_i + \rho g_i(\mathbf{x})) \nabla^2 g_i(\mathbf{x}) + \rho \mathrm{J}G(\mathbf{x})^\intercal \mathrm{J}G(\mathbf{x}),$$

and $g_i(\mathbf{x}_*) = \mathbf{0}$ for all $i = 1, \ldots, m$, we get

$$\nabla^2 L_A(\mathbf{x}_*, \boldsymbol{\lambda}_*, \rho) = \nabla^2 f(\mathbf{x}_*) + \sum_{i=1}^m \lambda_i^* \nabla^2 g_i(\mathbf{x}_*) + \rho \mathrm{J}G(\mathbf{x}_*)^\intercal \mathrm{J}G(\mathbf{x}_*).$$

Note that $\mathrm{J}G(\mathbf{x}_*)^\intercal \mathrm{J}G(\mathbf{x}_*)$ is a positive semidefinite matrix. Under certain conditions, when ρ is large enough, $\nabla^2 L_A(\mathbf{x}_*, \boldsymbol{\lambda}_*, \rho)$ can be guaranteed to be positive definite and thus $\mathbf{x}_*$ becomes a minimizer of $L_A(\mathbf{x}, \boldsymbol{\lambda}_*, \rho)$.

Theorem 7.5. *Let*

$$(\mathbf{x}_*(\rho), \boldsymbol{\lambda}_*(\rho)) = \operatorname{argmin} L_A(\mathbf{x}, \boldsymbol{\lambda}, \rho) \quad \textit{and} \quad \mathbf{x}_* = \operatorname{argmin}\{f(\mathbf{x}) : G(\mathbf{x}) = \mathbf{0}\}.$$

Under mild conditions, there exists $\bar{\rho} > 0$ such that

$$\mathbf{x}_*(\rho) = \mathbf{x}_*$$

whenever $\rho > \bar{\rho}$.

Example 7.6. The univariate optimization problem

$$\begin{aligned} \text{minimize} \quad & x^3 \\ \text{subject to} \quad & x + 1 = 0, \end{aligned}$$

has obvious minimizer $x_* = -1$ and critical Lagrange multiplier $\lambda_* = -3$. The Lagrangian

$$L(x, \lambda_*) = x^3 - 3(x + 1)$$

has $x_* = -1$ as an unconstrained *maximizer* (not minimizer!). However the augmented Lagrangian

$$L_A(x, \lambda_*, \rho) = x^3 - 3(x + 1) + \frac{\rho}{2}(x + 1)^2$$

has local minimizer $x_* = -1$ if $\rho > 6 =: \bar{\rho}$.

For any of these methods that involve a penalty parameter ρ, the way you implement them is to pick an increasing sequence of parameters $\rho_1 < \rho_2 < \rho_3 < \cdots$, e.g., $\rho_k = 2^k$ or $\rho_k = 10^k$. We then use the kth minimizer $\mathbf{x}_*(\rho_k)$ as the starting point of the $(k + 1)$th iteration, i.e.

$$\mathbf{x}_0(\rho_{k+1}) = \mathbf{x}_*(\rho_k).$$

If we just pick an excessively large value of ρ or a sequence of ρ_k's that increases too fast, the method will fail. Choosing an appropriate sequence is an empirical process that depends on many factors — more an art that comes from experience than a science.

▶ Applications of equality constrained optimization

The most pertinent applications for equality constrained optimization problems are usually from economics. We will see why in the examples below.

Example 7.7 (Utility and production functions)**.** We write $\mathbb{R}_+ := [0, \infty)$ for the nonnegative reals below. There are two basic problems in economics:

- maximize *utilty* $f \colon \mathbb{R}^n_+ \to \mathbb{R}$ from consuming x_i units of commodity i at price $a_i \in \mathbb{R}_+$, $i = 1, \dots, n$ with constraint $a_1x_1 + \cdots + a_nx_n \le b$ where $a_1x_1 + \cdots + a_nx_n$ is the *expenditure* and $b \in \mathbb{R}_+$ is the *budget* or *income*. A common form of f is the *Cobb–Douglas utility function* where

  $$f(x_1, \dots, x_n) = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$$

  for some parameters $\alpha_1, \dots, \alpha_n \in \mathbb{R}_+$ that quantify *economies of scale*.

- minimize *cost* $a_1x_1 + \cdots + a_nx_n$ for producing $g(x_1, \dots, x_n)$ units of output given $x_i \ge 0$ units of input i at price $a_i \in \mathbb{R}_+$, $i = 1, \dots, n$ with constraint that we want an output level of at least $b \in \mathbb{R}$, i.e., $g(x_1, \dots, x_n) \ge b$. Here $g \colon \mathbb{R}^n \to \mathbb{R}$ is the *production function*, with a common form given by the *CES production function* where

  $$g(x_1, \dots, x_n) = c(\alpha_1 x_1^\rho + \cdots + \alpha_n x_n^\rho)^{1/\rho},$$

  with $\rho \in (-\infty, 0) \cup (0, 1)$ the *substitution parameter*, $\alpha_1, \dots, \alpha_n \in \mathbb{R}_+$ the *distribution parameters*, c the *total factor productivity* or *efficiency* parameter, and CES stands for Constant Elasticity of Substitution.

Note that the CES function is a weighted "p-norm" where $p < 1$, i.e., it is not really a norm, but we have discussed this in Lecture 3. As we can see they are both about optimizing various quantities subjected to various constraints. The Cobb–Douglas function is sometimes used as a *production function* with different interpretations for the variables and parameters; likewise the CES function is sometimes also used as a utility function. The two functions are intimately related, if we write g_ρ to emphsize the dependence of g on ρ, then

$$\lim_{\rho \to 0} g_\rho(x_1, \dots, x_n) = f(x_1, \dots, x_n),$$

see Remarks 7.21 for the $n = 2$ case. At first glance the optimization problems described above should be inequality constrained but they are in fact equivalent to equality constrained optimization problems with the "$\leq$" and "$\geq$" replaced by "$=$". Why? Because any solution must meet the income/production constraint with equality or else the utility/cost can be further increased/decreased — we will discuss it in the examples below. Hence utility maximization and cost minimization are equality constrained optimization problems:

$$\begin{aligned} \text{maximize} \quad & x_1^{\alpha_1} \cdots x_n^{\alpha_n} \\ \text{subject to} \quad & a_1 x_1 + \cdots + a_n x_n = b \end{aligned}$$

and

$$\begin{aligned} \text{minimize} \quad & a_1 x_1 + \cdots + a_n x_n \\ \text{subject to} \quad & c(\alpha_1 x_1^\rho + \cdots + \alpha_n x_n^\rho)^{1/\rho} = b. \end{aligned}$$

□

Example 7.8 (Utility maximization)**.** As our utility function, we take the Cobb–Douglas function with $n = 2$ and to reduce clutter of subscripts we write x, y for variables and α, β for scale parameters. Consider the utility maximization problem:

$$\begin{aligned} \text{maximize} \quad & x^\alpha y^\beta \\ \text{subject to} \quad & a_1 x + a_2 y \leq b. \end{aligned}$$

Let $(x_*, y_*) \in \mathbb{R}^2_+$ be a solution. If we just take $x = b/2a_1$, $y = b/2a_2$, then $f(x, y) > 0$; so we know that $x = 0$ or $y = 0$ cannot be solutions as we will have $f(x, y) = 0$. Therefore we know that the solution must lie in the region with $x > 0$ and $y > 0$. In this region

$$\frac{\partial f}{\partial x} = \alpha x^{\alpha-1} y^\beta > 0, \qquad \frac{\partial f}{\partial y} = \beta x^\alpha y^{\beta-1} > 0$$

so f is a strictly increasing function of x for any $y > 0$ and a strictly increasing function of y for any $x > 0$, i.e., the solution (x_*, y_*) we seek must satisfy the inequality constraint with equality since if $a_1 x + a_2 y < b$ we can further increase $f(x, y)$ by increasing x or y or both until we have $a_1 x + a_2 y = b$. Hence we might as well consider

$$\begin{aligned} \text{maximize} \quad & x^\alpha y^\beta \\ \text{subject to} \quad & a_1 x + a_2 y = b, \end{aligned}$$

which may be solved using the methods we have covered. While we may just solve this problem as is (try it to see the difference) but there is a trick to simplify calculations. Note

that the problem has the same solution if we take log of the objective function since log is a strictly increasing function and we are only interested in the region with $x > 0$ and $y > 0$:

$$\begin{aligned} \text{maximize} \quad & \alpha \log x + \beta \log y \\ \text{subject to} \quad & a_1 x + a_2 y = b. \end{aligned}$$

The Lagrangian is

$$L(x, y, \lambda) = \alpha \log x + \beta \log y + \lambda(b - a_1 x - a_2 y).$$

First order condition and the equality constraint give us

$$\nabla L(x_*, y_*, \lambda_*) = \begin{bmatrix} \alpha/x_* - \lambda_* a_1 \\ \beta/y_* - \lambda_* a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad a_1 x_* + a_2 y_* = b.$$

Adding the first two equations $\alpha = \lambda_* a_1 x_*$ and $\beta = \lambda_* a_2 y_*$ and substituting the third $a_1 x_* + a_2 y_* = b$, we get $\lambda_* = (\alpha + \beta)/b$ and hence

$$x_* = \frac{\alpha b}{(\alpha + \beta) a_1}, \qquad y_* = \frac{\beta b}{(\alpha + \beta) a_2}, \qquad \lambda_* = \frac{\alpha + \beta}{b} \qquad \square$$

Example 7.9 (Cost minimization)**.** To simplify notation further, we set $\alpha = \beta = 1$ in the Cobb–Douglas function in the previous example and use $g(x, y) = xy$ as a production function. Consider the cost minimization problem:

$$\begin{aligned} \text{minimize} \quad & a_1 x + a_2 y \\ \text{subject to} \quad & xy \geq b. \end{aligned}$$

Since $b > 0$ the solution clearly lies in the region with $x > 0$ and $y > 0$. Since $a_1 > 0$ and $a_2 > 0$ (or otherwise a solution cannot exist), if $xy > b$ we can further reduce $a_1 x + a_2 y$ by reducing x or y or both until we have $xy = b$. Hence we obtain an equality constrained minimization problem:

$$\begin{aligned} \text{minimize} \quad & a_1 x + a_2 y \\ \text{subject to} \quad & xy = b. \end{aligned}$$

The Lagrangian is

$$L(x, y, \lambda) = a_1 x + a_2 y + \lambda(b - xy).$$

First order condition and the equality constraint give us

$$\nabla L(x_*, y_*, \lambda_*) = \begin{bmatrix} a_1 - \lambda_* y_* \\ a_2 - \lambda_* x_* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad x_* y_* = b.$$

As $\lambda_* \neq 0$ by the first equation, we get

$$x_* = \left(\frac{a_2 b}{a_1}\right)^{1/2}, \qquad y_* = \left(\frac{a_1 b}{a_2}\right)^{1/2}, \qquad \lambda_* = \left(\frac{a_1 a_2}{b}\right)^{1/2}. \tag{7.10}$$

We will take this opportunity to illustrate the second order sufficient condition in Theorem 6.17 to show that this is indeed a local minimizer. Since

$$\nabla^2 L(x_*, y_*, \lambda_*) = \begin{bmatrix} 0 & -\lambda_* \\ -\lambda_* & 0 \end{bmatrix}$$

and since $\mathrm{J}g(x_*, y_*) = \begin{bmatrix}1 & 1\end{bmatrix} \in \mathbb{R}^{1\times 2}$,

$$\begin{aligned}\ker\big(\mathrm{J}g(x_*, y_*)\big) &= \left\{\begin{bmatrix}z_1\\z_2\end{bmatrix} \in \mathbb{R}^2 : \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}z_1\\z_2\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}\right\}\\ &= \left\{\begin{bmatrix}z_1\\z_2\end{bmatrix} \in \mathbb{R}^2 : z_1 + z_2 = 0\right\} = \left\{\begin{bmatrix}t\\-t\end{bmatrix} \in \mathbb{R}^2 : t \in \mathbb{R}\right\},\end{aligned}$$

we just need to check that for all $t \neq 0$,

$$\begin{bmatrix}t\\-t\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}0 & -\lambda_*\\-\lambda_* & 0\end{bmatrix}\begin{bmatrix}t\\-t\end{bmatrix} = 2t^2\lambda_* = 2t^2(a_1a_2/b)^{1/2} > 0,$$

and therefore the solution in (7.10) is indeed a local minimizer. In fact with a bit more work, we can show that it is actually a global minimizer. □

Remark 7.11. The reason we were able to reduce the inequality constrained optimization problems to equality constrained ones in Examples 7.8 and 7.9 is really convexity. The sets

$$\{(x, y) \in \mathbb{R}^2 : a_1x + a_2y \leq b\} \quad \text{and} \quad \{(x, y) \in \mathbb{R}^2 : xy \geq b\}$$

are both convex sets. Whenever we maximize a convex function or minimize a concave function over a convex set, the solution is always attained at the *extreme points* in

$$\{(x, y) \in \mathbb{R}^2 : a_1x + a_2y = b\} \quad \text{and} \quad \{(x, y) \in \mathbb{R}^2 : xy = b\}.$$

This follows from the inequality $f(t\mathbf{x}_1+(1-t)\mathbf{x}_2) \leq tf(\mathbf{x}_1)+(1-t)f(\mathbf{x}_2) \leq \max\{f(\mathbf{x}_1), f(\mathbf{x}_2)\}$ for any convex function f and any $t \in [0, 1]$. □

Example 7.12 (Marshallian demand)**.** Consider the utility maximization problem

$$\begin{aligned}\text{maximize} \quad & f(x_1, \dots, x_n)\\ \text{subject to} \quad & a_1x_1 + \cdots + a_nx_n = b.\end{aligned} \tag{7.13}$$

A *demand function* gives the amount of a good a consumer will purchase as a function of commodity prices $a_1, \dots, a_n$ and consumer income b,

$$\mathbf{x}_*(\mathbf{a}, b) = \begin{bmatrix}x_1^*(a_1, \dots, a_n, b)\\ \vdots\\ x_n^*(a_1, \dots, a_n, b)\end{bmatrix} = \operatorname*{argmax}_{a_1x_1+\cdots+a_nx_n=b} f(x_1, \dots, x_n),$$

i.e., it is the solution of (7.13) regarded as a function of prices $a_1, \dots, a_n$ and budget b. For the Cobb–Douglas function, we get

$$x_i^*(a_1, \dots, a_n, b) = \frac{\alpha_i b}{(\alpha_1 + \cdots + \alpha_n)a_i}, \qquad i = 1, \dots, n.$$

Note that we did the $n = 2$ case in Example 7.8. For the CES function, we get

$$x_i^*(a_1, \dots, a_n, b) = \frac{(\alpha_i/a_i)^\sigma b}{\alpha_1^\sigma a_1^{1-\sigma} + \cdots + \alpha_n^\sigma a_n^{1-\sigma}}, \qquad i = 1, \dots, n,$$

where $\sigma := 1/(1 - \rho)$ is called the *elasticity of substitution*. Observe that when $\sigma = 1$ we get the Cobb–Douglas demand function above. You may wish to derive these expressions as an exercise.

- By plugging the demand function $\mathbf{x}_*(\mathbf{a}, b)$ back into the utility function, we obtain the maximum value of (7.13)

$$f_*(a_1, \dots, a_n, b) \coloneqq f\big(x_1^*(a_1, \dots, a_n, b), \dots, x_n^*(a_1, \dots, a_n, b)\big) \tag{7.14}$$

  as a function of prices $a_1, \dots, a_n$ and budget b. This is called the *indirect utility function*.

- By plugging the demand function $\mathbf{x}_*(\mathbf{a}, b)$ back into the left hand side of the constraint, we obtain the *optimal expenditure function*

$$e_*(a_1, \dots, a_n, b) \coloneqq a_1 x_1^*(a_1, \dots, a_n, b) + \cdots + a_n x_n^*(a_1, \dots, a_n, b) \tag{7.15}$$

  as a function of prices $a_1, \dots, a_n$ and budget b.

The first order condition applied to (7.13) gives us

$$\frac{\partial f}{\partial x_i}(x_1^*, \dots, x_n^*) = \lambda_* a_i, \qquad i = 1, \dots, n. \tag{7.16}$$

By the equality constraint, we always have

$$a_1 x_1^*(a_1, \dots, a_n, b) + \cdots + a_n x_n^*(a_1, \dots, a_n, b) = b$$

and taking partial derivative with respect to a_i gives

$$x_i^*(a_1, \dots, a_n, b) + \sum_{i=1}^n a_i \frac{\partial x_i^*}{\partial a_i}(a_1, \dots, a_n, b) = 0, \qquad i = 1, \dots, n, \tag{7.17}$$

and taking partial derivative with respect to b gives

$$\sum_{i=1}^n a_i \frac{\partial x_i^*}{\partial b}(a_1, \dots, a_n, b) = 1. \tag{7.18}$$

Taking partial derivative of (7.14) with respect to a_i, we get

$$\begin{aligned}
\frac{\partial f_*}{\partial a_i}(a_1, \dots, a_n, b) &= \sum_{i=1}^n \frac{\partial f}{\partial x_i}(x_1^*, \dots, x_n^*) \frac{\partial x_i^*}{\partial a_i}(a_1, \dots, a_n, b) && \text{by chain rule} \\
&= \sum_{i=1}^n \lambda_* a_i \frac{\partial x_i^*}{\partial a_i}(a_1, \dots, a_n, b) && \text{by (7.16)} \\
&= -\lambda_* x_i^*(a_1, \dots, a_n, b) && \text{by (7.17)},
\end{aligned}$$

for $i = 1, \dots, n$. Taking partial derivative of (7.14) with respect to b, we get

$$\begin{aligned}
\frac{\partial f_*}{\partial b}(a_1, \dots, a_n, b) &= \sum_{i=1}^n \frac{\partial f}{\partial x_i}(x_1^*, \dots, x_n^*) \frac{\partial x_i^*}{\partial b}(a_1, \dots, a_n, b) && \text{by chain rule} \\
&= \sum_{i=1}^n \lambda_* a_i \frac{\partial x_i^*}{\partial b}(a_1, \dots, a_n, b) && \text{by (7.16)} \\
&= \lambda_* && \text{by (7.18)}.
\end{aligned}$$

Eliminating λ_*, we obtain an expression for the demand function

$$x_i^*(a_1, \dots, a_n, b) = -\frac{\partial f_* / \partial a_i}{\partial f_* / \partial b}, \qquad i = 1, \dots, n. \qquad \square$$

Example 7.19 (Shadow prices)**.** These examples allow us to inutitively understand the Lagrange multipliers in an Economics setting. It is clearest when we are talking about cost minimization. If we are allowed to choose our $\mathbf{x}$ freely provided we pay a *fine* λ_i whenever $g_i(\mathbf{x})$ is nonzero, i.e., the constraint is violated, then the cost of choosing $\mathbf{x}$ is

$$f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \cdots + \lambda_m g_m(\mathbf{x}).$$

Here the amount of fine we pay is proportional to how much the constraint is violated, i.e., $\lambda_i g_i(\mathbf{x})$, and a negative fine means a reward. In Economics, λ_i^* is called a *shadow price*, i.e., the maximum price the producer is prepared to pay for violating the constraint $g_i(\mathbf{x}) = 0$ — we will discuss this again in the next lecture. The Economics lesson drawn is that by introducing the right amouts of fines and rewards, the producer will automatically be led to satisfy the constraints by his own desire to minimize cost, there is no need to enforce constraints by law.

Utility maximization provides another perspective. We use the simple problem (7.13) in the previous example because it only involves a single Lagrange multiplier. Taking partial derivative of (7.15) with respect to b, we get

$$\frac{\partial e_*}{\partial b}(a_1, \dots, a_n, b) = \sum_{i=1}^{n} a_i \frac{\partial x_i^*}{\partial b}(a_1, \dots, a_n, b). \tag{7.20}$$

Hence we obtain an expression for the shadow price:

$$\lambda_* = \frac{\partial f_*/\partial b}{\partial e_*/\partial b} = \frac{\text{change in optimal utility}}{\text{change in optimal expenditure}},$$

recalling that the indirect utility function f_* is the optimal utility and e_* is the optimal expenditure. In other words, every unit of optimal expenditure leads to λ_* units of optimal utility. □

Remark 7.21. We have been using notations consistent with those we have been using in this course. But in Economics, standard notations for these quantities and parameters are very different from the ones used here. These are extremely important functions in real life — for instance, the Congressional Budget Office (CBO) assumes that the US economy's production function is Cobb–Douglas and uses it to forecase potential output and analyze policy changes — so we should at least look at the conventional notations and say a few words about their meaning.

For $n = 2$, the two-factor Cobb–Douglas production function is written as

$$Y = AK^\alpha L^\beta$$

with efficiency parameter $A > 0$, and $0 < \alpha, \beta < 1$ parameters describing the output elasticity of capital and labor respectively; the two-factor CES production function is written as

$$Y = A[\alpha K^\rho + (1-\alpha)L^\rho]^{1/\rho}$$

with efficiency parameter $A > 0$, distribution parameter $0 < \alpha < 1$, and substitution parameter $\rho \in (-\infty, 0) \cup (0, 1)$. In either case the independent variables K and L represent

the quantities of capital and labor respectively and the dependent variable Y represents the quantity of output.

If we scale the input K and L in Cobb–Douglas by a factor t, then

$$A(tK)^\alpha (tL)^\beta = t^{\alpha+\beta} A K^\alpha L^\beta,$$

i.e., the output get scaled by a factor of $t^{\alpha+\beta}$. The cases $\alpha+\beta = 1$, $\alpha+\beta < 1$, and $\alpha+\beta > 1$ are thus referred to as constant, decreasing, and increasing return to scale respectively. But if we scale the input K and L in CES by a constant factor t, then

$$A[\alpha(tK)^\rho + (1-\alpha)(tL)^\rho]^{1/\rho} = tA[\alpha K^\rho + (1-\alpha)L^\rho]^{1/\rho}$$

i.e., the output of CES is scaled by t. Since CES always has constant return to scale, it cannot capture returns to scale like Cobb–Douglas. On the other hand, CES is good for modeling inputs that can be substitutes for one another. Indeed the parameter ρ, which has no counter part in Cobb–Douglas, measures the ease to which one can switch between K and L: as $\rho \to 1$, they become perfectly substitutable for each other; as $\rho \to -\infty$,

$$\lim_{\rho\to-\infty} [\alpha K^\rho + (1-\alpha)L^\rho]^{1/\rho} = \min\{\alpha K, (1-\alpha)L\},$$

they become perfectly complementary to each other; as $\rho \to 0$,

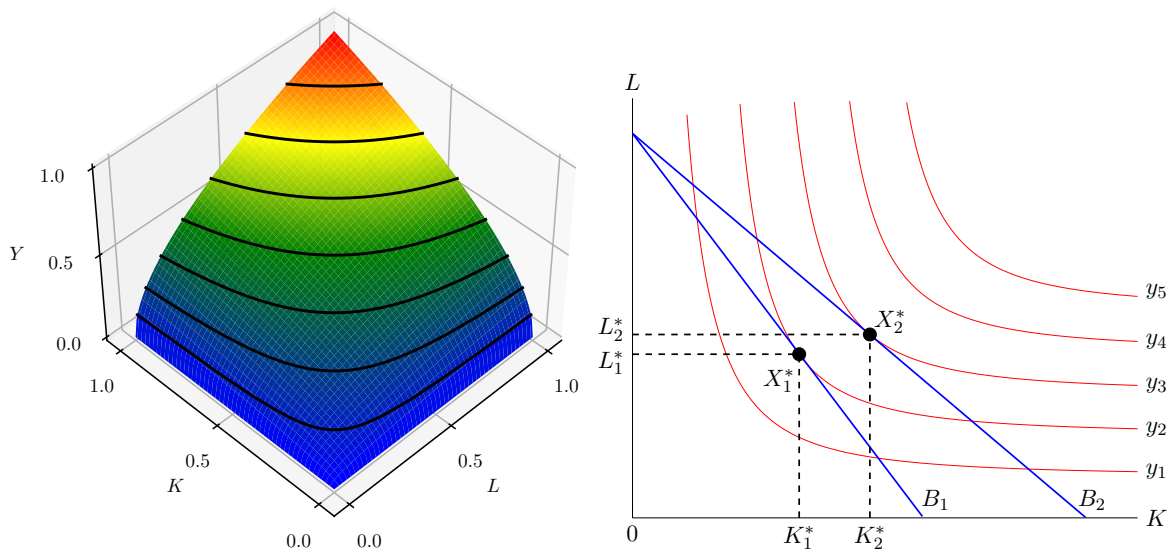
$$\begin{aligned}
\lim_{\rho\to 0} \log[\alpha K^\rho + (1-\alpha)L^\rho]^{1/\rho} &= \lim_{\rho\to 0} \frac{\log[\alpha K^\rho + (1-\alpha)L^\rho]}{\rho} \\
&= \lim_{\rho\to 0} \frac{\alpha K^\rho \log K + (1-\alpha)L^\rho \log L}{\alpha K^\rho + (1-\alpha)L^\rho} \qquad \text{(by l'Hôpital)} \\
&= \alpha \log K + (1-\alpha)\log L \\
&= \log(K^\alpha L^{1-\alpha}),
\end{aligned}$$

so CES reduces to Cobb–Douglas and loses the ability to capture substitutability. This also shows that the parameter α in both CES and Cobb–Douglas captures the relative share of the factors K and L.

The budget constraint is usually written

$$P_K K + P_L L = B$$

where P_K and P_L are respectively the *price of capital* and *price of labor* (sometimes called cost of capital and cost of labor) and B is the budget of the producer. For $n = 3$, a common three-factor model would include a third independent variable M that represents the quantity of raw materials and supplies with a corresponding modification to the production function and constraint.



The picture on the left shows a two-factor Cobb–Douglas production function. The picture on the right show five of its level curves or *isoquants* in red, i.e., where the quantity Y has constant value y_1, y_2, y_3, y_4, y_5 respectively. In Economics, people study the effects of changing prices and budgets, i.e., the same optimization problem under different equality constraints:

$$\begin{array}{ll} \text{maximize} & AK^\alpha L^\beta \\ \text{subject to} & P_K K + P_L L = B, \end{array} \quad \longrightarrow \quad \begin{array}{ll} \text{maximize} & AK^\alpha L^\beta \\ \text{subject to} & P'_K K + P'_L L = B'. \end{array}$$

Suppose we want to see what happens when the price of capital P_K increases to P'_K while the price of labor P_L and budget B stay the same. The blue line with K-intercept $B_1 = B/P'_K$ is the constraint $P'_K K + P_L L = B$; the blue line with K-intercept $B_2 = B/P_K$ is the constraint $P_K K + P_L L = B$. As discussed in the proof of Theorem 6.11, the maximizers $X_1^* = (K_1^*, L_1^*)$ and $X_2^* = (K_2^*, L_2^*)$ are exactly the points where the respective constraints are tangential to the level curves. That the demand for capital decreases from K_2^* to K_1^* when the price of capital increases from P_K to P'_K illustrates the famous maxim "demand slopes downward," i.e., $K_*(P_K, P_L, B)$ has $\partial K_*/\partial P_K < 0$.

The discussion above is all from the perspective of a producer. But it essentially applies to a consumer with some slight change in terminologies — utility function instead of production function, income instead of budget, various goods instead of capital/labor/materials, etc.

Optimization	*Consumer Theory*	*Producer Theory*
objective	utility	production
equality constraints	income/budget constraints	cost/expenditure constraints
primal variables	quantity of goods and services	quantity of production factors
dual variables	rewards and fines	rewards and fines
optimal primal variables	Marshallian demand	factor demand
optimal dual variables	shadow prices	shadow prices

The bottom line is that a lot of Economics is nothing more than giving economically meaningful interpretation to various quantities in an equality constrained optimization problem. □

Example 7.22 (Flow prediction)**.** Suppose we observe the flow of a commodity into and out of n ports $i = 1, \ldots, n$. These are entirely abstract — the commodities could be anything

from actual cargos shipped over sea routes to data packets transmitted over the internet; the ports could be anything from an actual container terminals to cloud servers or data centers. Let x_{ij} be the amount of commodity sent from port i to port j — we do not know what x_{ij} is because we could only measure what the total amount that arrives at each port $a_i \in \mathbb{R}$, $i = 1, \dots, n$, and the total amount that leaves each port $b_i \in \mathbb{R}$, $i = 1, \dots, n$. Since the sum total of everything arriving at every port must equal the sum total of everything entering every port, we must have

$$a_1 + \cdots + a_n = b_1 + \cdots + b_n = c \tag{7.23}$$

where $c \in \mathbb{R}$ denotes the common value of these sums, i.e., c is the total amount of commodities being traded. In other words, we have the constraints

$$\sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1, \dots, n, \qquad \sum_{i=1}^{n} x_{ij} = b_j, \quad j = 1, \dots, n, \tag{7.24}$$

where we know the values $a_1, \dots, a_n, b_1, \dots, b_n \in \mathbb{R}$ and need to find the values of the unknown x_{ij}'s. The *method of maximal entropy* (which we would not go into) gives us the minimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \log x_{ij} - x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1, \dots, n, \\ & \sum_{i=1}^{n} x_{ij} = b_j, \quad j = 1, \dots, n, \end{aligned}$$

where there is an implicit constraint $x_{ij} > 0$, $i, j = 1, \dots, n$. The Lagrangian is

$$\begin{aligned} & L(x_{11}, \dots, x_{nn}, \lambda_1, \dots, \lambda_n, \lambda'_1, \dots, \lambda'_n) \\ & \qquad = \sum_{i=1}^{n}\sum_{j=1}^{n} (x_{ij} \log x_{ij} - x_{ij}) + \sum_{i=1}^{n} \lambda_i \left[a_i - \sum_{j=1}^{n} x_{ij} \right] + \sum_{j=1}^{n} \lambda'_j \left[b_j - \sum_{i=1}^{n} -x_{ij} \right] \end{aligned}$$

and the first order condition gives

$$0 = \frac{\partial}{\partial x_{ij}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\lambda}') = \log x_{ij} - \lambda_i - \lambda'_j, \qquad i, j = 1, \dots, n,$$

and so

$$x_{ij} = e^{\lambda_i} e^{\lambda'_j}, \quad i, j = 1, \dots, n.$$

Plugging this back into (7.24), we get

$$\begin{aligned} e^{\lambda_i} &= \frac{a_i}{e^{\lambda'_1} + \cdots + e^{\lambda'_n}}, \qquad i = 1, \dots, n, \\ e^{\lambda'_j} &= \frac{b_j}{e^{\lambda_1} + \cdots + e^{\lambda_n}}, \qquad j = 1, \dots, n; \end{aligned}$$

and plugging any of them into (7.23), we get

$$(e^{\lambda_1} + \cdots + e^{\lambda_n})(e^{\lambda'_1} + \cdots + e^{\lambda'_n}) = c.$$

Therefore we obtain our solution

$$x_{ij} = \frac{a_i b_j}{c}, \qquad i, j = 1, \ldots, n.$$

In more realistic situations things are of course not so simple. We may have incomplete information; we may not know all values of a_i's and b_j's and may only know aggregate information, e.g., the amount of commodities flowing into the Eurozone instead each individual port. We may have cargo ships losing containers due to weather or piracy or have packet loss due to damaged submarine cables, i.e., some of the equality constraints would have to be replaced by inequality constraints. □

Lecture 8 (Tuesday, November 14)

We now move on to the last topic:

$$
\begin{tikzcd}
\text{univariate} \arrow[r] & \text{multivariate unconstrained} \arrow[d] & \\
& \text{multivariate equality constrained} \arrow[d] & \\
\text{convex} & \boxed{\text{multivariate inequality constrained}} \arrow[l] \arrow[r] \arrow[d] & \text{nonsmooth} \\
& \begin{array}{c}\text{multiobjective, combinatorial,}\\ \text{dynamic, stochastic, integer, etc}\end{array} &
\end{tikzcd}
$$

While it is impossible to discuss the advanced topics within the scope of a one-quarter introductory course, we have incorporated various examples of these throughout the quarter in earlier lectures (e.g., nonsmooth), this one (convex, multiobjective, stochastic), and the problem sets (integer, combinatorial, dynamic), emphasizing an important strategy for solving them, namely, that they may all be reduced to the kind of continuous optimization problems we discussed over these eight lectures.

► Inequality constrained optimization

We will write an inequality constrained optimization problem in the form

$$
\begin{aligned}
\text{minimize} \quad & f(\mathbf{x}) \\
\text{subject to} \quad & H(\mathbf{x}) \leq \mathbf{0},
\end{aligned}
$$

where $H : \mathbb{R}^n \to \mathbb{R}^p$ is the function $H(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_p(\mathbf{x}))$ and where we have assumed that equality constraints, if any, have been expressed as equivalent inequality constraints.

Slack variables

As we saw earlier that every equality constrained optimization problem can be formulated as an inequality constrained optimization problem by introducing *more constraints*. It turns out that every inequality constrained optimization problem can also be formulated as an equality constrained optimization problem by introducing *more variables*. By simply observing that

$$
h_i(\mathbf{x}) \leq 0 \qquad \Leftrightarrow \qquad h_i(\mathbf{x}) + s_i = 0,\ s_i \geq 0.
$$

The variable s_i is called a *slack variable*. By introducing one slack variable for each inequality constraint $s_1, \ldots, s_p$, we may write an inequality constrained optimization problem as an equality constrained optimization problem with nonnegative variables (6.9):

$$
\begin{aligned}
\text{minimize} \quad & f(\mathbf{x}) \\
\text{subject to} \quad & h_i(\mathbf{x}) + s_i = 0, \quad i = 1, \ldots, p, \\
& \mathbf{s} \geq \mathbf{0}.
\end{aligned}
\tag{8.1}
$$

As we mentioned earlier, this is as good as an equality constrained optimization problem, which means all previously discussed methods for equality constrained optimization apply verbatim. But note that while there were n optimization variables $x_1, \dots, x_n$ in the original problem, there are now $n+p$ optimization variables $x_1, \dots, x_n, s_1, \dots, s_p$ in the new problem.

However, if one insists on a purely equality constrained optimization problem, then we may write our slack variable in the form s_i^2, noting that this is always nonnegative and so we do not need to impose nonnegativity constraint, giving us

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h_i(\mathbf{x}) + s_i^2 = 0, \qquad i = 1, \dots, p. \end{aligned}$$

Nevertheless the form in (8.1) is more common and preferred. For example, a common use of slack variables is to transform linear inequality constraints in linear programming from one form to another:

$$A\mathbf{x} \le \mathbf{b} \quad \longrightarrow \quad \begin{cases} A\mathbf{x} + \mathbf{s} = \mathbf{b}, \\ \qquad\quad \mathbf{s} \ge \mathbf{0}. \end{cases}$$

In order to get a linear programming problem after the transformation, we have to keep all variables linear so we cannot introduce quadratic terms like s_i^2.

Penalty methods

We will first introduce some standard terminologies. For the inequality constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & H(\mathbf{x}) \le \mathbf{0}, \end{aligned} \qquad \text{or} \qquad \begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h_i(\mathbf{x}) \le 0, \quad i = 1, \dots, p, \end{aligned}$$

we have the following:

Definition 8.2. with respect to a constraint $h_i(\mathbf{x}) \le 0$ and a point $\mathbf{x} \in \mathbb{R}^n$, the point $\mathbf{x}$ is said to be

- *feasible* if $h_i(\mathbf{x}) \le 0$,
- *infeasible* if $h_i(\mathbf{x}) > 0$;

and the constraint $h_i(\mathbf{x}) \le 0$ is said to be

- *active* if $h_i(\mathbf{x}) = 0$,
- *inactive* if $h_i(\mathbf{x}) < 0$,
- *violated* if $h_i(\mathbf{x}) > 0$.

The *feasible region* is the set $\{\mathbf{x} : H(\mathbf{x}) \le \mathbf{0}\}$ and its complement is called the *infeasible region*. The term *binding* is used synonymously as 'active' for a constraint.

Let $\mathrm{ReLU}(t) \coloneqq \max(t, 0)$ and extend it to vector variables in a coordinatewise sense: for any $\mathbf{x} \in \mathbb{R}^n$,

$$\mathrm{ReLU}(\mathbf{x}) \coloneqq \begin{bmatrix} \mathrm{ReLU}(x_1) \\ \vdots \\ \mathrm{ReLU}(x_n) \end{bmatrix}.$$

Observe that $\mathrm{ReLU}(H(\mathbf{x}))$ keeps track of the violated constraints. With this, it is straightforward to modify the penalty function method for inequality constraints — just replace G with $\mathrm{ReLU}(H)$. The quadratic and l^1-penalty functions are given by

$$P_2(\mathbf{x}, \rho) = f(\mathbf{x}) + \frac{\rho}{2}\|\mathrm{ReLU}(H(\mathbf{x}))\|_2^2, \qquad P_1(\mathbf{x}, \rho) = f(\mathbf{x}) + \rho\|\mathrm{ReLU}(H(\mathbf{x}))\|_1.$$

Note that minimizing these for sufficiently large ρ have the effect of forcing $\mathbf{x}$ to move into the feasible region so that there are no violated constraint and the penalty term vanishes. We may also state analogues of the respective theorems for inequality constrained optimization. In addition to

$$\lim_{\rho \to +\infty} \mathbf{x}_*(\rho) = \mathbf{x}_*,$$

one may also show

$$\lim_{\rho \to +\infty} \rho h_i(\mathbf{x}_*(\rho)) = -\lambda_i^*.$$

In particular it follows (why?) that $\lambda_i^* \geq 0$ — in the next lecture we will see that this is actually the dual feasibility condition in the so-called KKT conditions.

Unlike equality constrained optimization, we have to keep track of which constraints are violated and which are not. Ideally we want a method that avoids this. This consideration leads us to more sophisticated methods based on the optimality conditions for inequality constrained optimization, the famous *KKT conditions*.

▶ Optimality conditions for inequality constrained optimization

The corresponding result for general constrained optimality with both equality and inequality constraints is more complicated and is known as the Karush–Kuhn–Tucker or KKT conditions, by far the most important conditions in optimization theorem.

Theorem 8.3 (Karush–Kuhn–Tucker conditions)**.** *Let $\mathbf{x}_*$ be a local minimizer of*

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) = 0, \quad i = 1, \dots, m, && (8.4) \\ & h_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, p. && (8.5) \end{aligned}$$

There exist $\lambda_1^*, \dots, \lambda_m^*, \nu_1^*, \dots, \nu_p^* \in \mathbb{R}$ such that

1. *stationarity:*

$$\nabla f(\mathbf{x}_*) + \sum_{i=1}^m \lambda_i^* \nabla g_i(\mathbf{x}_*) + \sum_{i=1}^p \nu_i^* \nabla h_i(\mathbf{x}_*) = \mathbf{0},$$

2. *primal feasibility:*

$$\begin{aligned} g_i(\mathbf{x}_*) &= 0, \quad i = 1, \dots, m, \\ h_i(\mathbf{x}_*) &\leq 0, \quad i = 1, \dots, p, \end{aligned}$$

3. *dual feasibility:*

$$\nu_i^* \geq 0, \quad i = 1, \dots, p,$$

4. complementary slackness:

$$\nu_i^* h_i(\mathbf{x}_*) = 0, \quad i = 1, \ldots, p.$$

There are two important University of Chicago connections here: William Karush first established the theorem as part of his M.S. in Mathematics thesis here. Harold Kuhn and Albert Tucker discovered the conditions independently of Karush almost twelve years later but received name recognition because they were already famous. Moreover, Tucker was a postdoc here as well.

Definition 8.6. The function defined by

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^\mathsf{T} G(\mathbf{x}) + \boldsymbol{\nu}^\mathsf{T} H(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \sum_{i=1}^{p} \nu_i h_i(\mathbf{x})$$

is called the *Lagrangian* of the optimization problem (8.4). The variables:

$$\mathbf{x} = (x_1, \ldots, x_n)$$

are called *primal variables* and the variables:

$$\begin{aligned} \boldsymbol{\lambda} &= (\lambda_1, \ldots, \lambda_n) \\ \boldsymbol{\nu} &= (\nu_1, \ldots, \nu_p) \end{aligned}$$

are called *dual variables*.

Note that the stationarity condition in the KKT conditions is nothing more than

$$\nabla L(\mathbf{x}_*, \boldsymbol{\lambda}_*, \boldsymbol{\nu}_*) = \mathbf{0}.$$

Primal-dual methods

One strategy to solve inequality constrained optimization problems is to take the KKT conditions and convert the inequality constraints $h_i(\mathbf{x}_*) \leq 0$, $i = 1, \ldots, p$, into equality constraints with slack variables, i.e., $h_i(\mathbf{x}_*) + s_i = 0$, $s_i \geq 0$, $i = 1, \ldots, p$. Now we apply Newton method for nonlinear equations to the resulting system of nonlinear equations. This is the idea behind so called primal-dual method. The actual method is actually a bit more subtle and requires that we use something called perturbed KKT conditions instead of the standard KKT conditions.

Example 8.7 (Linear complementarity problem)**.** The special case of the quadratic programming problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^\mathsf{T} A\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x} + c \\ \text{subject to} \quad & A\mathbf{x} + \mathbf{b} \geq \mathbf{0}, \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

gives us the following KKT conditions

- $\mathbf{x} \geq \mathbf{0}$,
- $\boldsymbol{\nu} \geq \mathbf{0}$,
- $\boldsymbol{\nu}^\mathsf{T}\mathbf{x} = 0$,
- $\boldsymbol{\nu} = A\mathbf{x} + \mathbf{b}$.

Note that in general complementary slackness cannot be expressed as an inner product $\boldsymbol{\nu}^\mathsf{T}\mathbf{x} = 0$ but in this case it can because of the nonnegativity constraints $\mathbf{x}$. This set of four conditions is so common they are given its own name: *linear complementarity problem* or LCP. It arises as the *Nash equilibrium* for so-called bimatrix games (very important in game theory and economics); in contact dynamics for modeling wheels and brakes (very important in designing motorized vehicles); in impulse-constraint simulations in game engines (very important in video games and movie CGI). When $A \succ 0$, applying Newton method for nonlinear equations to this system allows us to compute a solution in polynomial time. However the problem is NP-hard in general when $A \not\succ 0$.

Barrier methods

While penalty function methods work by coercing $\mathbf{x}$ to move from infeasible region into feasible region, *barrier function* methods work by trapping $\mathbf{x}$ within the feasible region so that it never move into infeasible region. Because of this, barrier function method requires that we start from an initial point $\mathbf{x}_0$ that is feasible, i.e., lies within the feasible region.

The two most common types of barrier functions are the *log barrier*:

$$b(\mathbf{x}, \mu) = f(\mathbf{x}) - \mu \sum_{i=1}^{p} \log(-h_i(\mathbf{x})),$$

and the *inverse barrier*:

$$b(\mathbf{x}, \mu) = f(\mathbf{x}) - \mu \sum_{i=1}^{p} \frac{1}{h_i(\mathbf{x})},$$

where $\mu \geq 0$. Note that whenever $\mathbf{x}$ moves close to the boundary, i.e., where $h_i(\mathbf{x}) = 0$ for some $i \in \{1, \dots, p\}$, the barrier term blows up. This produces the effect of never allowing $\mathbf{x}$ to escape into the infeasible region but the trouble is that if the minimizer $\mathbf{x}_*$ is near the boundary, $\mathbf{x}$ would kept away from it. The trick is to gradually decrease μ to zero.

Theorem 8.8. *Let*

$$\mathbf{x}_*(\mu) = \operatorname{argmin} b(\mathbf{x}, \mu) \quad \textit{and} \quad \mathbf{x}_* = \operatorname{argmin}\{f(\mathbf{x}) : H(\mathbf{x}) \leq \mathbf{0}\}.$$

Under mind conditions,

$$\lim_{\mu \to 0^+} \mathbf{x}_*(\mu) = \mathbf{x}_*.$$

In fact, we may also show

$$\lim_{\mu \to 0^+} \frac{\mu}{h_i(\mathbf{x}_*(\mu))} = -\lambda_i^*.$$

Note that in particular it follows (why?) that $\lambda_i^* \geq 0$, i.e., the dual feasibility condition in the KKT conditions is satisfied.

Example 8.9. The univariate optimization problem

$$\begin{aligned} \text{minimize} \quad & x^2 \\ \text{subject to} \quad & 1 - x \leq 0, \end{aligned}$$

has obvious minimizer $x_* = 1$ (exercise). The log barrier function

$$b(x, \mu) = x^2 - \mu \log(x - 1)$$

has unconstrained minimizer (exercise)

$$x_*(\mu) = \frac{1}{2} + \frac{1}{2}\sqrt{1 + 2\mu}.$$

Clearly $x_*(\mu) \to 1 = x_*$ and $\mu / h(x_*(\mu)) \to 0 = -\lambda_*$ as $\mu \to 0^+$.

Example 8.10. Consider the linearly constrained quadratic programming (LCQP) problem

$$\begin{aligned} \text{minimize} \quad & x^2 - 2xy + 4y^2 \\ \text{subject to} \quad & 1 - x/10 + y \leq 0. \end{aligned}$$

The log barrier function is

$$b(x, y, \mu) = x^2 - 2xy + 4y^2 - \mu \log(x/10 - y - 1).$$

Figure 17 shows the contour plots of the barrier function as μ decreases from 100 to 0.01. Note that:

- the contours of $b(x, y, \mu)$ converge to those $f(x, y) := x^2 - 2xy + 4y^2$ as $\mu \to 0^+$;
- the feasible region is everything below the red line $h(x, y) := 1 - x/10 + y = 0$;
- all contours of $b(x, y, \mu)$ are confined to the feasible region.

▶ Convex optimization

Barrier functions are extremely important in convex optimization under the name *interior point method*, which is essentially Newton method for optimization applied to the barrier function, or, alternatively, Newton method for nonlinear equations applied to the KKT conditions. Convex optimization problem is essentially one where the objective function f, the equality constraints $g_1, \ldots, g_m$, and the inequality constraints $h_1, \ldots, h_p$ in (8.4) are all convex functions. Its most attractive feature is that one can do *global optimization* with convex optimization because any stationary point is necessarily a global minimizer.

Another attractive feature, and one of the biggest breakthrough in convex optimization in the 20th Century, is the demonstration that if the feasible set has a barrier that satisfies a property called *self-concordance*, then (8.4) can be solved using Newton method to any ε-accuracy in polynomial time (under some mild assumptions). Self-concordant barriers exist for many types of convex optimization problems and that's why they may all be solved efficiently. We will briefly review the most common types of polynomial-time solvable convex optimization problems below:

Figure 17: Contours of $b(x, y, \mu)$ in Example 8.10 as the barrier parameter μ decreases through $10^2, 10^{1.9}, 10^{1.8}, \ldots, 10^{-1.9}, 10^{-2.0}$. The static contours are those of $f(x, y)$ and the red line is $h(x, y) = 0$. You need a PDF viewer capable of displaying animation (e.g., Adobe Acrobat) to see this figure.

Linear Programming (LP). This is the most common type of optimization problems where f, g_i, h_i in (8.4) are all linear functions:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, \end{aligned} \tag{8.11}$$

for $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$. Using the log barrier function, we may convert this constrained optimization problem into an unconstrained optimization problem

$$\text{minimize} \quad \mathbf{c}^\mathsf{T}\mathbf{x} - \mu \sum\nolimits_{i=1}^{m} \log(\mathbf{b} - \mathbf{a}_i^\mathsf{T}\mathbf{x})$$

where $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{R}^n$ are the columns of A. Does this look familiar? If we write everything in terms of their entries, we get

$$\text{minimize} \quad \sum\nolimits_{j=1}^{n} c_j x_j - \mu \sum\nolimits_{i=1}^{m} \log\left(b_i - \sum\nolimits_{j=1}^{n} a_{ij} x_j\right)$$

in Homework 2, Problem 4 with $n = 100$ variables and $m = 500$ linear constraints. Of course in the homework, we did it for $\mu = 1$ but it is trivial to modify whatever code you have to add a barrier parameter $\mu \geq 0$ and thereby obtain a sequence of solutions $\mathbf{x}_*(\mu)$ that converges to the solution $\mathbf{x}_*$ of (8.11) as we decrease $\mu \to 0$.

Quadratic Programming (QP). This is if we allow f, g_i, h_i in (8.4) to be quadratic functions:

$$\begin{aligned} \text{minimize} \quad & \mathbf{x}^\mathsf{T} A_0 \mathbf{x} + \mathbf{b}_0^\mathsf{T} \mathbf{x} \\ \text{subject to} \quad & \mathbf{x}^\mathsf{T} A_i \mathbf{x} + \mathbf{b}_i^\mathsf{T} \mathbf{x} \leq c_i, \quad i = 1, \dots, p \end{aligned} \tag{8.12}$$

with $A_0, A_1, \dots, A_p \in \mathbb{R}^{n \times n}$, $\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_p \in \mathbb{R}^n$, and $c_1, \dots, c_p \in \mathbb{R}$. Using the log barrier function, we may convert this constrained optimization problem into an unconstrained optimization problem

$$\text{minimize} \quad \mathbf{x}^\mathsf{T} A_0 \mathbf{x} + \mathbf{b}_0^\mathsf{T} \mathbf{x} - \mu \sum\nolimits_{i=1}^{p} \log(c_i - \mathbf{b}_i^\mathsf{T} \mathbf{x} - \mathbf{x}^\mathsf{T} A_i \mathbf{x}).$$

In general optimization of QPs are NP-hard problems. However, if all $A_0, A_1, \dots, A_p \in \mathbb{R}^{n \times n}$ are symmetric positive definite matrices, then we obtain a *convex QP*, which is solvable in polynomial time to ε-accuracy using this barrier function, which can be shown to be self-concordant. The special case where the constraints are linear:

$$\begin{aligned} \text{minimize} \quad & \mathbf{x}^\mathsf{T} A_0 \mathbf{x} + \mathbf{b}_0^\mathsf{T} \mathbf{x} \\ \text{subject to} \quad & B\mathbf{x} \leq \mathbf{c} \end{aligned} \tag{8.13}$$

is called a linearly constrained quadratic program (LCQP). The more general case in (8.12) is sometimes called a quadratically constrained quadratic program (QCQP).

Second-order cone program (SOCP). This refers to a powerful generalization of convex QP that takes the form:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}_0^\mathsf{T} \mathbf{x} \\ \text{subject to} \quad & \|A_i \mathbf{x} + \mathbf{b}_i\| \leq \mathbf{c}_i^\mathsf{T} \mathbf{x} + d_i, \quad i = 1, \dots, p \end{aligned} \tag{8.14}$$

with $A_1, \dots, A_p \in \mathbb{R}^{m \times n}$, $\mathbf{b}_1, \dots, \mathbf{b}_p \in \mathbb{R}^m$, $\mathbf{c}_0, \mathbf{c}_1, \dots, \mathbf{c}_p \in \mathbb{R}^n$, and $d_1, \dots, d_p \in \mathbb{R}$. Using the log barrier function, we may convert this constrained optimization problem into an unconstrained optimization problem

$$\text{minimize} \quad \mathbf{c}_0^\mathsf{T} \mathbf{x} - \mu \sum\nolimits_{i=1}^{p} \log[(\mathbf{c}_i^\mathsf{T} \mathbf{x} + d_i)^2 - \|A_i \mathbf{x} + \mathbf{b}_i\|^2].$$

The constraints are squared for computational convenience (avoid taking square roots) but can be shown to be equivalent to the original problem.

Semidefinite Programming (SDP). If we let the columns of the matrix A in (8.11) be $A = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n]$, then since $A\mathbf{x} = x_1 \mathbf{a}_1 + x_2 \mathbf{a}_2 + \dots x_n \mathbf{a}_n$, the linear programming problem (8.11) may be written as

$$\begin{aligned} \text{minimize} \quad & c_1 x_1 + \cdots + c_n x_n \\ \text{subject to} \quad & x_1 \mathbf{a}_1 + x_2 \mathbf{a}_2 + \dots x_n \mathbf{a}_n \leq \mathbf{b}. \end{aligned}$$

An SDP problem is just one where we replace the vectors $\mathbf{a}_1, \dots, \mathbf{a}_n, \mathbf{b} \in \mathbb{R}^m$ by symmetric matrices $A_1, \dots, A_n, B \in \mathbb{R}^{m \times m}$,

$$\begin{aligned} \text{minimize} \quad & c_1 x_1 + \cdots + c_n x_n \\ \text{subject to} \quad & x_1 A_1 + x_2 A_2 + \dots x_n A_n \preceq B \end{aligned} \tag{8.15}$$

where we write $X \succeq Y$ if and only if $X - Y$ is positive definite. To solve this problem using barrier function, we solve the unconstrained problem

$$\text{minimize} \quad \sum\nolimits_{j=1}^{n} c_j x_j - \mu \log \det(B - x_1 A_1 - x_2 A_2 - \cdots - x_n A_n),$$

for $\mu \geq 0$ and obtain a sequence of solutions $X_*(\mu)$ that converges to the solution X_* of (8.15) as we decrease $\mu \to 0$.

Geometric Programming (GP). It is the same story if we replace the linear functions in the objective and constraints of (8.11) by log-sum-exp functions we discussed earlier. The result is

$$\begin{aligned} \text{minimize} \quad & \textsc{lse}(A_0 \mathbf{x} + \mathbf{b}_0) \\ \text{subject to} \quad & \textsc{lse}(A_i \mathbf{x} + \mathbf{b}_i) \leq 0, \quad i = 1, \dots, p \end{aligned}$$

with $A_0, A_1, \dots, A_p \in \mathbb{R}^{m \times n}$, $\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_p \in \mathbb{R}^m$. Using the log barrier function, we may convert this constrained optimization problem into an unconstrained optimization problem

$$\text{minimize} \quad \textsc{lse}(A_0 \mathbf{x} + \mathbf{b}_0) - \mu \sum\nolimits_{i=1}^{p} \log(-\textsc{lse}(A_i \mathbf{x} + \mathbf{b}_i)).$$

If we write out the LSE functions in full glory, we see that barrier function involves nested logs:

$$\log\Big(\sum\nolimits_{j=1}^{m_0} \exp(\mathbf{a}_{0j}^{\mathsf{T}} \mathbf{x} + b_{0j})\Big) - \sum\nolimits_{i=1}^{p} \log\Big[-\log\Big(\sum\nolimits_{j=1}^{m_i} \exp(\mathbf{a}_{ij}^{\mathsf{T}} \mathbf{x} + b_{ij})\Big)\Big]$$

but there is nothing to fear — this is self-concordant and always convex. Therefore a GP is solvable in polynomial time to ε-accuracy.

Basically, LP, (convex) QP, SOCP, SDP, and GP are all examples of convex optimization problems that are solvable in polynomial time to ε-accuracy because they all have self-concordant barrier functions. In particular, the first four fit into a hierarchy

$$\text{LP} \longrightarrow \text{LCQP} \longrightarrow \text{QCQP} \longrightarrow \text{SOCP} \longrightarrow \text{SDP}$$

where each is a special case of the succeeding one but it gets increasingly more difficult to solve problems further on the right. For example, we can solve LPs (with no special structures like sparsity) with millions of variables and millions of constraints within an hour on a personal computer but for SDPs (with no special structures like sparsity), it would just be in the hundreds of thousands (see `https://plato.asu.edu/bench.html` for the latest numbers).

▶ Applications of inequality constrained optimization

We start with an example that uses only unconstrained optimization. The simple example conveys an important lesson that explains some of our later examples, namely, it is not necessarily a good idea to just maximize profits. A commonly held belef is that a company must maximize its profit since if not it would be driven from the market by comapnies that do.

Example 8.16 (Profit versus revenue maximization)**.** Suppose market demand for a certain product is proportional to its price

$$f(x, y) = 1 - x - y$$

where x and y are the production output of Companies X and Y respectively. Suppose each company has the same constant cost $c \in \mathbb{R}_+$. Company X just wants to maximize profit:

$$\underset{x \in \mathbb{R}_+}{\text{maximize}} \quad (1 - x - y - c)x.$$

Company Y on the other hand wants to maximize profits but give some weight $\alpha \in \mathbb{R}_+$ to total revenue:

$$\underset{y \in \mathbb{R}_+}{\text{maximize}} \quad (1 - x - y - c)y + \alpha(1 - x - y)y.$$

Company X only gets to choose x and its first order condition gives

$$0 = \frac{\partial}{\partial x}(1 - x - y - c)x = 1 - 2x - y - c.$$

Company Y only gets to choose y and its first order condition gives

$$0 = \frac{\partial}{\partial y}(1 - x - y - c)y + \alpha(1 - x - y)y = 1 - x - 2y - c + \alpha - \alpha x - 2\alpha y.$$

Solving the last two equations, we get

$$x_* = \frac{2}{3}(1 - c) - \frac{1}{3}g(\alpha), \qquad y_* = \frac{2}{3}g(\alpha) - \frac{1}{3}(1 - c)$$

where $g(\alpha) := (1 + \alpha - c)/(1 + \alpha)$. So the optimal profit of Company X is

$$(1 - x_* - y_* - c)x_* = \Big(\frac{2}{3}(1 + c) - \frac{1}{3}g(\alpha)\Big)\Big(\frac{2}{3}(1 - c) - \frac{1}{3}g(\alpha)\Big)$$

and the optimal profit of Company Y is

$$(1 - x_* - y_* - c)y_* = \Big(\frac{2}{3}(1 + c) - \frac{1}{3}g(\alpha)\Big)\Big(\frac{2}{3}g(\alpha) - \frac{1}{3}(1 - c)\Big).$$

Note that $g(\alpha)$ is an increasing function of α. Exercise: Verify that the profit of Company X is decreasing in α and the profit of Company Y is increasing in α as long as

$$g(\alpha) < \min\Big(\frac{3c + 5}{4c}, 2\Big).$$

In other words, Company X, which set out to maximize its own profit, ultimately made lower profit than Company Y, which didn't. The bottom line is that any market invariably has more than one player. The type of optimization above where each company optimizes its own production output, taking the opponent's output level as given is called *Nash equilibirum*.

Example 8.17 (Portfolio optimization as LP)**.** We have assets $1, \dots, n$ with current values $c_1, \dots, c_n$, expected future values $f_1, \dots, f_n$, expected dividends $d_1, \dots, d_n$, and our current holdings are $h_1, \dots, h_n$. We desire an investment strategy with the following:

(i) no extra investment, i.e., portfolio value must remain the same after change in holdings;

(ii) future portfolio value to grow at least $g \times 100\%$ from current portfolio value;

(iii) balanced portfolio with each asset value at least $b \times 100\%$ of current portfolio value;

(iv) no short positions.

Let $x_1, \dots, x_n$ be change in holdings of assets $1, \dots, n$. There are various LPs we may formulate:

Maximize portfolio value subject to (i) and (iv):

$$\begin{aligned} \text{maximize} \quad & c_1(x_1 + h_1) + \cdots + c_n(x_n + h_n) \\ \text{subject to} \quad & c_1(x_1 + h_1) + \cdots + c_n(x_n + h_n) = c_1 h_1 + \cdots + c_n h_n, \\ & x_i + h_i \geq 0, \quad i = 1, \dots, n. \end{aligned}$$

This has a trivial and obvious solution — if we order returns so that $c_1 \geq c_2 \geq \cdots \geq c_n$, then the solution is given by

$$x_1 = (c_1 h_1 + \cdots + c_n h_n - h_1)/c_1, \; x_2 = -h_2, \dots, x_n = -h_n.$$

If the future values are fixed and known, we are going to put our entire budget in the asset with the highest return.

Maximize dividends subject to all (i), (ii), (iii), (iv):

$$\begin{aligned} \text{maximize} \quad & d_1(x_1 + h_1) + \cdots + d_n(x_n + h_n) \\ \text{subject to} \quad & c_1(x_1 + h_1) + \cdots + c_n(x_n + h_n) = c_1 h_1 + \cdots + c_n h_n, \\ & f_1(x_1 + h_1) + \cdots + f_n(x_n + h_n) \geq (c_1 h_1 + \cdots + c_n h_n)(1 + g), \\ & [c_1(x_1 + h_1) + \cdots + c_n(x_n + h_n)]b \leq c_i(x_i + h_i), \quad i = 1, \dots, n, \\ & x_i + h_i \geq 0, \quad i = 1, \dots, n. \end{aligned}$$

Note that as soon as we include dividends, the problem becomes not entirely trivial (cannot be solved by inspection) even if assuming knowledge of future values.

Of course knowledge of future values and dividends is an unrealistic assumption. These should all be random variables. Even if we could estimate their expected values, once we introduce randomness, we need to worry about risk. □

Example 8.18 (Basic portfolio terminologies)**.** The previous example assumes that we hold n assets over a single period. More generally, portfolio optimization assumes "n assets held over m periods." We define some basic terms and notations in this context. A portfolio comprising n assets have an overall return given by

$$r_1 x_1 + \cdots + r_n x_n = \mathbf{r}^{\mathsf{T}} \mathbf{x}$$

where $r_1, \dots, r_n$ are *random variables* representing rate of return of assets $j = 1, \dots, n$, and $x_1, \dots, x_n$ are variables representing holdings of these assets. We say that $x_j > 0$ is a *long* position and $x_j < 0$ a *short* position on asset j. Usually we hold the portfolio over m successive periods of time indexed by $i = 1, \dots, m$. We define

$$\text{rate of return at time } i := \frac{\text{value at the end of } i\text{th period} - \text{value at beginning of } i\text{th period}}{\text{value at beginning of } i\text{th period}}.$$

We observe the random vector $\mathbf{r}$ over a period of time so that we obtain $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_m \in \mathbb{R}^n$ at m different points in time. The mean vector and covariance matrix of $\mathbf{r}$ are estimated by

$$\boldsymbol{\mu} := \frac{1}{m}\sum_{i=1}^{m} \mathbf{r}_i \in \mathbb{R}^n, \qquad \Sigma := \frac{1}{m}\sum_{i=1}^{m} (\mathbf{r}_i - \boldsymbol{\mu})(\mathbf{r}_i - \boldsymbol{\mu})^{\mathsf{T}} \in \mathbb{R}^{n\times n}$$

respectively. The *profit* and *risk* are then given by mean of returns and variance of returns:

$$\mathbb{E}(\mathbf{r}^{\mathsf{T}}\mathbf{x}) = \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x}, \qquad \operatorname{Var}(\mathbf{r}^{\mathsf{T}}\mathbf{x}) = \mathbf{x}^{\mathsf{T}}\Sigma\mathbf{x}$$

respectively. The word *loss* is sometimes used to refer to the negative of the profit, i.e., $-\boldsymbol{\mu}^{\mathsf{T}}\mathbf{x}$, particularly when the profit is negative. We also write $\mathbf{1} = (1, \dots, 1)$ for a vector of all ones so that $\mathbf{1}^{\mathsf{T}}\mathbf{x} = x_1 + \cdots + x_n$, note that this is different from the 1-norm $\|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n|$. $\square$

Example 8.19 (Markowitz portfolio optimization)**.** The most basic portfolio optimization problem that incorporates risk is a QP that represents return maximization:

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} \\ \text{subject to} \quad & \mathbf{x}^{\mathsf{T}}\Sigma\mathbf{x} \le \sigma^2, \\ & \mathbf{1}^{\mathsf{T}}\mathbf{x} = b, \end{aligned} \tag{8.20}$$

or a QP that represents risk minimization:

$$\begin{aligned} \text{minimize} \quad & \mathbf{x}^{\mathsf{T}}\Sigma\mathbf{x} \\ \text{subject to} \quad & \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} \ge \mu, \\ & \mathbf{1}^{\mathsf{T}}\mathbf{x} = b, \end{aligned} \tag{8.21}$$

or a QP that represents minimization of risk-sensitive return:

$$\begin{aligned} \text{minimize} \quad & -\boldsymbol{\mu}^{\mathsf{T}}\mathbf{x} + \gamma\mathbf{x}^{\mathsf{T}}\Sigma\mathbf{x} \\ \text{subject to} \quad & \mathbf{1}^{\mathsf{T}}\mathbf{x} = b. \end{aligned} \tag{8.22}$$

Here $\boldsymbol{\mu} \in \mathbb{R}^n$ is the portfolio's expected rate of return, $\Sigma \in \mathbb{R}^{n\times n}$ is the portfolio variance, commonly interpreted as *risk* or *volatility*, σ^2 and μ are respectively the maximum allowed risk and minimum acceptable return, $\gamma \in \mathbb{R}_+$ the *risk-aversion* parameter, and b the initial capital or budget. So here (8.21) and (8.22) are LCQPs. If $\gamma \to 0$, we get a maximum return without regard for risk whereas if $\gamma \to \infty$ we get a minimum risk return without regard for return. Except for Examples 8.17 and 8.33, all portfolio optimization models discussed here are some version of such risk–return trade-off.

The fact that we are trying to optimize multiple different functions on the same variables $\min_{\mathbf{x}}(f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))$, or equivalently optimizing a vector-valued objective function F :

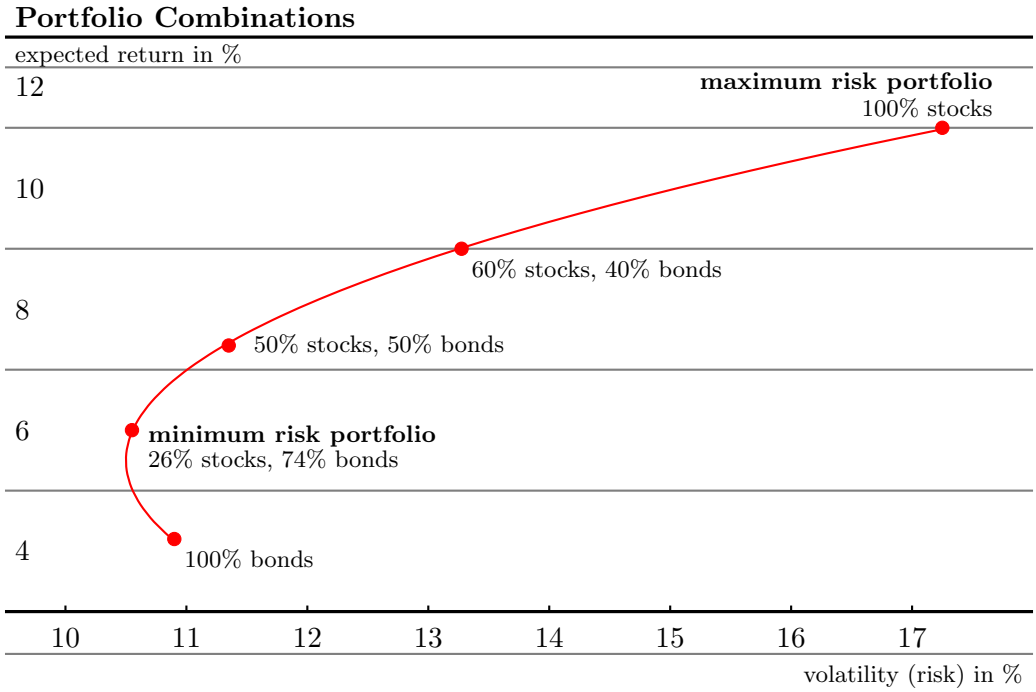


Figure 18: Pareto frontier of the vector optimization problem $\min(-\boldsymbol{\mu}^\mathsf{T}\mathbf{x}, \mathbf{x}^\mathsf{T}\Sigma\mathbf{x})$.

$\mathbb{R}^n \to \mathbb{R}^m$, gives us what's called the *Pareto frontier* or *efficient frontier* shown in the red curve in Figure 18, i.e., these are the values $F(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_m(\mathbf{x})$ for which it is impossible to decrease one $f_i(\mathbf{x})$ without increasing some other $f_j(\mathbf{x})$'s, and are called *Pareto optima*. This is the topic of *multiobjective optimization* or *vector optimization*.

The optimization problems (8.20), (8.21), (8.22) all finds Pareto optimum solutions for $\min(-\boldsymbol{\mu}^\mathsf{T}\mathbf{x}, \mathbf{x}^\mathsf{T}\Sigma\mathbf{x})$, i.e., the optimum values of all these problems are points on the red curve; one that doesn't lie on this curve is called inefficient.

Figure 18 illustrates a key insight of modern portfolio theory: the least risky possible portfolio is not one that is purely bonds as one might naively believe — one can actually decrease risk even further by throwing in some risky assets like stocks. An important precondition here is *low correlation*: different types of assets with low correlations like stocks and bonds come together to form a portfolio that is greater than the sum of its parts, in this case an overall portfolio with lower risk than bonds alone. This would not have been possible if we had chosen two assets that move in tandem. □

There is a University of Chicago connection here: Harry Markowitz was a triple alumnus, receiving his bachelors, masters, and doctoral degrees here. The guy below, William Sharpe, was Markowitz's de facto PhD student.

Example 8.23 (Sharpe ratio optimization)**.** The famous Sharpe ratio is defined by

$$\operatorname{Sharpe}(\mathbf{x}) := \frac{\boldsymbol{\mu}^\mathsf{T}\mathbf{x} - \mu_0}{\sqrt{\mathbf{x}^\mathsf{T}\Sigma\mathbf{x}}}$$

where μ_0 is the risk-free return (e.g., interests from FDIC-insured bank deposits). It is a measure of reward-to-risk as the amount of expected return in excess of risk-free rate per unit risk. A Shrape-optimal portfolio would therefore be one that

$$\begin{aligned} \text{maximize} \quad & \frac{\boldsymbol{\mu}^\mathsf{T}\mathbf{x} - \mu_0}{\sqrt{\mathbf{x}^\mathsf{T}\Sigma\mathbf{x}}} \\ \text{subject to} \quad & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} \geq \mu_0, \\ & \mathbf{1}^\mathsf{T}\mathbf{x} = b, \end{aligned}$$

which is equivalent to

$$\begin{aligned} \text{minimize} \quad & \frac{\sqrt{\mathbf{x}^\mathsf{T}\Sigma\mathbf{x}}}{\boldsymbol{\mu}^\mathsf{T}\mathbf{x} - \mu_0} \\ \text{subject to} \quad & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} \geq \mu_0, \\ & \mathbf{1}^\mathsf{T}\mathbf{x} = b, \end{aligned}$$

which is in turn equivalent to

$$\begin{aligned} \text{minimize} \quad & \sqrt{\mathbf{x}^\mathsf{T}\Sigma\mathbf{x}} \\ \text{subject to} \quad & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} - \mu_0 = 1, \\ & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} \geq \mu_0, \\ & \mathbf{1}^\mathsf{T}\mathbf{x} = b, \end{aligned}$$

making the inequality constraint superfluous. Recall from Lecture 4 that a positive semidefinite matrix Σ must be a covariance matrix, i.e., $\Sigma = R^\mathsf{T}R$ for some $R \in \mathbb{R}^{n\times n}$, so $\sqrt{\mathbf{x}^\mathsf{T}\Sigma\mathbf{x}} = \sqrt{(R\mathbf{x})^\mathsf{T}(R\mathbf{x})} = \|R\mathbf{x}\|$. Introducing a new variable t, we transformed the previous problem into an SOCP:

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & \|R\mathbf{x}\| \leq t, \\ & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} = 1 + \mu_0, \\ & \mathbf{1}^\mathsf{T}\mathbf{x} = b. \end{aligned}$$

This simple exercise illustrates one of the key tenets of optimization practice: It is not enough to formulate a real-world problem as an optimization problem. We want to transform it into an optimization problem for which we can do *global optimization* (e.g., convex optimization). Ideally we want to formulate it as an optimization problem for which we have a wealth of efficient software (e.g., LP, QP, SOCP, SDP, GP). □

Example 8.24 (Value-at-Risk optimization)**.** Famously concocted by quants at J. P. Morgan at the demand of its then-CEO Dennis Weatherstone for a single number that captures the firm-wide risk, available within 15 minutes of market close, the Value-at-Risk is defined as

$$\mathrm{VaR}_\beta(\mathbf{x}) \coloneqq \min_{\gamma\in\mathbb{R}}\Big[\mathbb{P}(-\mathbf{r}^\mathsf{T}\mathbf{x} \geq \gamma) \leq \beta\Big] = -\max_{\alpha\in\mathbb{R}}\Big[\mathbb{P}(\mathbf{r}^\mathsf{T}\mathbf{x} \leq \alpha) \leq \beta\Big]$$

for any $\beta \in (0, \frac{1}{2})$, recalling that $-\mathbf{r}^\mathsf{T}\mathbf{x}$ is the *loss*. Basically VaR is intended to capture the risk of a large loss ($\gamma \gg 0$ or equivalently $\alpha \ll 0$) caused by a black swan event ($\beta \approx 0$) that

threatens the survival of the company. For example if the VaR on a portfolio return is 80% at $\beta = 0.05$, it means that there is at most a 5% chance that the portfolio value will drop by more than 80% over the given period. The variable $\mathbf{r}$ here is a *random vector*, i.e., its coordinates $r_1, \ldots, r_n$ are all random variables. Indeed we will assume that $\mathbf{r} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. In general, an optimization problem that involves random variables is a *stochastic optimization* problem and at first glance Value-at-Risk optimization belongs to stochastic optimization. But we will see how we can formulate it in a way that avoids random variables. Let

$$\Phi(z) := \frac{1}{2\pi} \int_{-\infty}^{z} e^{-t^2/2}\, dt$$

denote the cumulative distribution function of a standard normal random variable $z \sim \mathcal{N}(0, 1)$. Under normality assumption

$$\mathbb{P}(x \le \alpha) = \mathbb{P}\Big(\frac{x-\mu}{\sigma} \le \frac{\alpha-\mu}{\sigma}\Big) = \mathbb{P}\Big(z \le \frac{\alpha-\mu}{\sigma}\Big) = \Phi\Big(\frac{\alpha-\mu}{\sigma}\Big).$$

Hence

$$\mathbb{P}(x \le \alpha) \le \beta \quad \Longleftrightarrow \quad \frac{\alpha-\mu}{\sigma} \le \Phi^{-1}(\beta) \quad \Longleftrightarrow \quad \mu + \Phi^{-1}(\beta)\sigma \ge \alpha.$$

Setting $\mu = \boldsymbol{\mu}^\mathsf{T}\mathbf{x}$ and $\sigma = \sqrt{\mathbf{x}^\mathsf{T}\Sigma\mathbf{x}} = \|R\mathbf{x}\|$ where $\Sigma = R^\mathsf{T}R$, we see that maximizing expected return subject to a bound on the VaR is an SOCP in $\mathbf{x}$:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} \\ \text{subject to} & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} + \Phi^{-1}(\beta)\|R\mathbf{x}\| \ge \alpha, \\ & \mathbf{1}^\mathsf{T}\mathbf{x} = b. \end{array}$$

The condition $\beta \in (0, \frac{1}{2})$ ensures that $\Phi^{-1}(\beta) \le 0$ or otherwise we would not have an SOCP. There are many variations of this simple formulation. There is no need to assume normality — we may replace Φ by the cumulative distribution function of whatever distribution we use. Another straightforward extension is to allow different risk β_i for different loss α_i

$$\mathbb{P}(x \le \alpha_1) \le \beta_1, \ldots, \mathbb{P}(x \le \alpha_k) \le \beta_k,$$

say at most a 5% chance of an 80% loss, say, at most a 10% chance of an 60% loss, say, at most a 15% chance of an 40% loss, etc. Yet another variant is to directly minimize an upper bound t on the VaR: Since (Exercise)

$$\mathrm{VaR}_\beta(\mathbf{x}) \le t \quad \Longleftrightarrow \quad \boldsymbol{\mu}^\mathsf{T}\mathbf{x} + t \ge \Phi^{-1}(1-\beta)\|R\mathbf{x}\|,$$

we obtain another SOCP in variables $\mathbf{x}$ and t:

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} + t \ge \Phi^{-1}(1-\beta)\|R\mathbf{x}\|, \\ & \boldsymbol{\mu}^\mathsf{T}\mathbf{x} \ge \mu, \\ & \mathbf{1}^\mathsf{T}\mathbf{x} = b. \end{array}$$

where μ is our desired return μ on investment and $\beta \in (0, \frac{1}{2})$ is our acceptable risk level. $\square$

Example 8.25 (Portfolio variance optimization)**.** Unlike the previous examples, here we do not want to ascertain an optimal portfolio holding $\mathbf{x} \in \mathbb{R}^n$ but the portfolio variance $\Sigma \in \mathbb{R}^{n\times n}$ given prior knowledge of the risks $s_k \in \mathbb{R}$ of various portfolios $\mathbf{x}_k \in \mathbb{R}^n$, $k = 1, \dots, m$. In other words, we know that $\mathbf{x}_1^\mathsf{T}\Sigma\mathbf{x}_1 = s_1, \dots, \mathbf{x}_m^\mathsf{T}\Sigma\mathbf{x}_m = s_m$ and we wish to find out the worst risk that a given portfolio $\mathbf{x}_0 \in \mathbb{R}^n$ can take. So we want to optimize over a matrix variable Σ in a way that

$$\begin{aligned} \text{maximize} \quad & \mathbf{x}_0^\mathsf{T}\Sigma\mathbf{x}_0 \\ \text{subject to} \quad & \mathbf{x}_k^\mathsf{T}\Sigma\mathbf{x}_k = s_k, \quad k = 1, \dots, m, \\ & \Sigma \succeq 0. \end{aligned}$$

Note that we need $\Sigma \succeq 0$ since Σ is a covariance matrix and necessarily positive semidefinite. So this is an SDP. A variant of this is when we are do not know $\mathbf{x}_k^\mathsf{T}\Sigma\mathbf{x}_k = s_k$ with certainty but only upper and lower bounds $\overline{s}_k$ and $\underline{s}_k$. The we have

$$\begin{aligned} \text{maximize} \quad & \mathbf{x}_0^\mathsf{T}\Sigma\mathbf{x}_0 \\ \text{subject to} \quad & \underline{s}_k \le \mathbf{x}_k^\mathsf{T}\Sigma\mathbf{x}_k \le \overline{s}_k, \quad k = 1, \dots, m, \\ & \Sigma \succeq 0, \end{aligned} \tag{8.26}$$

which is still an SDP. Yet another variation is when we have estimates (upper and lower bounds) for the covariances σ_{ij} of specific pairs of assets i and j, e.g., we know that asset i always increases in value when asset j does. Suppose $S \subset \{1, \dots, n\}^2$ are those pairs of assets. Then we have the following

$$\begin{aligned} \text{maximize} \quad & \mathbf{x}_0^\mathsf{T}\Sigma\mathbf{x}_0 \\ \text{subject to} \quad & \underline{s}_{ij} \le \sigma_{ij} \le \overline{s}_{ij}, \quad (i,j) \in S, \\ & \Sigma \succeq 0, \end{aligned} \tag{8.27}$$

Observe that $\sigma_{ij} = \mathbf{e}_i^\mathsf{T}\Sigma\mathbf{e}_j$ so (8.27) actually has the same form as (8.26). Yet another variant is when we only have information on the correlation $\rho_{ij} = \sigma_{ij}/\sqrt{\sigma_{ii}\sigma_{jj}}$ but we know the volatility σ_{ii} of each asset i exactly. This gives us

$$\begin{aligned} \text{maximize} \quad & \mathbf{x}_0^\mathsf{T}\Sigma\mathbf{x}_0 \\ \text{subject to} \quad & \underline{s}_{ij} \le \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} \le \overline{s}_{ij}, i, j = 1, \dots, n, \\ & \Sigma \succeq 0. \end{aligned}$$

More sophisticated variants would incorporate *factor models* into the structure of the covariance matrix like $\Sigma = F\Sigma_{\text{factor}}F^\mathsf{T} + D$. □

Example 8.28 (Bells and whistles)**.** One may incorporate varioxus additional features into the previous examples.

Transaction costs: A more realistic formulation of the budget constraint $\mathbf{1}^\mathsf{T}\mathbf{x} = b$ would have been $\mathbf{1}^\mathsf{T}\mathbf{x} + \tau(\mathbf{x}) = b$ where the function τ gives the *transaction costs* and is usually chosen to be linear or convex quadratic to keep the optimization problems in their required form (e.g., LCQP or QCQP). A simple version is given by

$$\tau(\mathbf{x}) = c\|\mathbf{x}\|_1 = c(|x_1| + \cdots + |x_n|)$$

assuming the same unit cost c for all transactions. Recall from earlier lectures that a 1-norm constraint can be expressed as linear inequality constraints.

Differential transaction costs: The simple version above does not differentiate between transaction costs for buying and selling. The way to do this is to write

$$\mathbf{x} = \mathbf{x}_{\text{ini}} + \mathbf{u}_{\text{buy}} - \mathbf{u}_{\text{sell}}, \quad \mathbf{u}_{\text{buy}} \geq \mathbf{0}, \quad \mathbf{u}_{\text{sell}} \geq \mathbf{0} \tag{8.29}$$

with $\mathbf{x}_{\text{ini}}$ the initial holding and $\mathbf{u}_{\text{buy}}$, $\mathbf{u}_{\text{sell}}$ the amounts we buy and sell at the end of the holding period. The budget constraint $\mathbf{1}^\top \mathbf{x}$ would have to be replaced by

$$(1 - c_{\text{sell}})\mathbf{1}^\top \mathbf{u}_{\text{sell}} = (1 + c_{\text{buy}})\mathbf{1}^\top \mathbf{u}_{\text{buy}} \tag{8.30}$$

where c_{sell} and c_{buy} are the unit transaction costs for selling and buying respectively. Note that with this modification, whatever optimization problem in $\mathbf{x}$ that we began with is now an optimization problem in $\mathbf{x}, \mathbf{u}_{\text{buy}}, \mathbf{u}_{\text{sell}}$ satisfying the additional linear constraints (8.29) and (8.30).

Limiting short selling: So far in all our previous examples we do not prohibit short selling. Since short selling has the potential for unlimited losses, some investors may want to avoid or limit its use. Avoiding short selling entirely can be simply accomplished by adding the constraint $\mathbf{x} \geq \mathbf{0}$. However this can be an overkill. A way to limit short selling is to introduce a linear constraint instead. We write

$$\mathbf{x} = \mathbf{x}_{\text{long}} - \mathbf{x}_{\text{short}}$$

with new variables

$$\mathbf{x}_{\text{long}} \geq \mathbf{0}, \quad \mathbf{x}_{\text{short}} \geq \mathbf{0}$$

representing long and short positions. Then the linear constraint

$$\mathbf{1}^\top \mathbf{x}_{\text{short}} \leq \eta \mathbf{1}^\top \mathbf{x}_{\text{long}} \tag{8.31}$$

where $\eta \in [0, 1]$ limits the total short positions as a fraction η of the long positions. Note that with this modification, whatever optimization problem in $\mathbf{x}$ that we began with is now an optimization problem in $\mathbf{x}, \mathbf{x}_{\text{short}}, \mathbf{x}_{\text{long}}$ satisfying the additional linear constraint (8.31).

Sector bounds: Say assets are divided into different sectors and the first assets $1, \ldots, k$ are in some sector. Then the linear constraint

$$\sum_{i=1}^{k} x_i \leq \eta \mathbf{1}^\top \mathbf{x}$$

where $\eta \in [0, 1]$ limits exposure to not more than a fraction η of total assets. This extends verbatim to other sectors, say, the next assets $k + 1, \ldots, l$ are from another sector and assets $l + 1, \ldots, m$ from yet another section, then the linear constraints

$$\sum_{i=k+1}^{l} x_i \leq \eta \mathbf{1}^\top \mathbf{x}, \qquad \sum_{i=l+1}^{m} x_i \leq \eta \mathbf{1}^\top \mathbf{x}$$

would impose the same limit on these sectors.

Diversification: Say you want to impose the condition that no group of k assets contains more than a fraction η of the total amount. This can be accomplished with the constraint

$$\sum_{i=1}^{k} x_{[i]} \le \eta \mathbf{1}^{\mathsf{T}}\mathbf{x} \tag{8.32}$$

where $\eta \in [0,1]$ and the ordered variable $x_{[i]} = i$th largest coordinate of $\mathbf{x}$. For instance if we set $k = \lfloor n/10 \rfloor$ and $\eta = 0.8$, then (8.32) says that no more than 80% of the total budget can be invested in any 10% of the assets. This nonstandard constraint (8.32) can be converted into standard linear constraints by introducing new variables s_0 and $\mathbf{s} = (s_1, \dots, s_n)$:

$$ks_0 + \mathbf{1}^{\mathsf{T}}\mathbf{s} \le \eta \mathbf{1}^{\mathsf{T}}\mathbf{x}, \qquad \mathbf{s} \ge \mathbf{0}, \qquad \mathbf{s} \ge \mathbf{x} - s_0\mathbf{1}.$$

Median risk: Recall that

$$\Sigma \coloneqq \frac{1}{m}\sum_{i=1}^{m}(\mathbf{r}_i - \boldsymbol{\mu})(\mathbf{r}_i - \boldsymbol{\mu})^{\mathsf{T}} \in \mathbb{R}^{n\times n}$$

and so far we have used average risk

$$\mathbf{x}^{\mathsf{T}}\Sigma\mathbf{x} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{r}_i^{\mathsf{T}}\mathbf{x} - \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x})^2$$

as our notion of risk. An alternative is to measure risk by

$$\frac{1}{m}\sum_{i=1}^{m}|\mathbf{r}_i^{\mathsf{T}}\mathbf{x} - \boldsymbol{\mu}^{\mathsf{T}}\mathbf{x}|$$

to reduce sensitivity to outliers. □

Example 8.33 (Kelly criterion)**.** Here is an example to show that the Markowitz risk–return trade-off philosophy is not the only game in town for portfolio optimization. With the same setup: n assets held over m periods, and we want to determine the best possible holdings $\mathbf{x} = (x_1, \dots, x_n)$ subjected to budget constraint $\mathbf{1}^{\mathsf{T}}\mathbf{x} = b$ without short positions $\mathbf{x} \ge \mathbf{0}$. At the beginning of each holding period i, we reinvest a fraction x_j of our total wealth accumulated in the previous holding period, w_{i-1}, in asset j, i.e., we invest $x_j w_{i-1}$ in asset j for $i = 1, \dots, m$, $j = 1, \dots, n$. Let $\lambda_i \coloneqq w_i/w_{i-1}$, the return during period i. So by the end of the m periods, our initial wealth $w_0 = b$ would have been multiplied by $\prod_{i=1}^{m}\lambda_i$. We want an $\mathbf{x}$ that maximizes our ultimate growth w_m/w_0, which is equivalent to maximizing

$$\frac{1}{m}\log\Big(\frac{w_m}{w_0}\Big) = \frac{1}{m}\log\Big(\frac{w_m}{w_{m-1}}\cdot\frac{w_{m-1}}{w_{m-2}}\cdots\frac{w_1}{w_0}\Big) = \frac{1}{m}\sum_{i=1}^{m}\log\lambda_i.$$

We will impose the simplifying assumption that the random variable λ_i can only take p possible values given by $\mathbf{r}_1^{\mathsf{T}}\mathbf{x}, \dots, \mathbf{r}_p^{\mathsf{T}}\mathbf{x}$ and that $\pi_k \coloneqq \mathbb{P}(\lambda_i = \mathbf{r}_k^{\mathsf{T}}\mathbf{x})$ is the same for all time

period $i = 1, \ldots, m$, depending only on $k = 1, \ldots, p$. We also assume that $\lambda_1, \ldots, \lambda_m$ are independent and thus I.I.D., thus allowing us to use the law of large numbers:

$$\lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} \log \lambda_i = \mathbb{E}(\log \lambda_i) = \sum_{k=1}^{p} \pi_k \log(\mathbf{r}_k^\top \mathbf{x}).$$

The log-optimal investment strategy is therefore for $\mathbf{x}$ to

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{k=1}^{p} \pi_k \log(\mathbf{r}_k^\top \mathbf{x}) \\ \text{subject to} & \mathbf{1}^\top \mathbf{x} = b, \\ & \mathbf{x} \geq \mathbf{0}. \end{array}$$

If we could estimate the values of $\pi_1, \ldots, \pi_p$ from historic data, then this is a convex optimization problem although not one of the standard ones (LP, QP, SOCP, SDP, GP). Note that there is one difference from previous examples: here our asset return rate $\mathbf{r}_k$ is not indexed by the time period j but by the index of the possible outcomes. We will see a variant of this in Homework 3.

This idea of choosing the bet size to maximize the expected value of the log of wealth is due to John L. Kelly Jr., and call *Kelly criterion*. It is purportedly used by investors like Warren Buffet, Bill Gross, and Jim Simons. Indeed Elwyn Berlekamp, Kelly's assistant at Bell Labs in the 1960s, had relied on this method in his work at Axcom, which evolved into Medallion Fund, Renaissance Technologies' flagship hedge fund. □