Master Notes

Professor Marcus

March 22, 2022

The date listed above is the last time it was edited. I will try to post the notes for each lecture before the lecture happens. It may happen that I will want to add to (or change) them based on a question in class.

If that happens, I will highlight the added/changed part like this.

It is also possible that I may not get to all of the material, in which case you should be sure to read it yourself.

If that happens, I will highlight the missed part like this.

It is also possible that there are errors in these notes that I learn about later.

If that happens, I will highlight the corrected part like this.

Note: if you have any issues seeing the difference between these colors, please let me know (I can easily change them).

Contents

1 Week 1: Linear Algebra Review and Intro to Optimization

1.1 Linear Algebra Review

In this class, when we say the word *vector*, we will mean a *column* vector:

$$\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

If I want to talk about an object that looks like

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

then I can write this as $\mathbf{v}^\intercal$ where $\intercal$ is the *transpose* operator (these are sometimes referred to as *covectors*). Note that some sources (like the Bertsimas/Tsitsiklis book) use $\mathbf{A}'$ to denote the transpose of $\mathbf{A}$.

Given a collection of vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subseteq \mathbb{R}^n$, the set of vectors $\mathbf{x} \in \mathbb{R}^n$ that can be written as

$$\mathbf{x} = \sum_i \alpha_i \mathbf{v}_i \tag{1}$$

for some values of $\alpha_i \in \mathbb{R}$ is called the *span* of $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$. Given vectors $\mathbf{v}_i \in \mathbb{R}^n$, the collection of vectors $\mathbf{x} \in \operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is called a *linear subspace of* $\mathbb{R}^n$.

In general, a linear subspace can be written as the span of vectors in a lot of different ways. In particular, if $\mathbf{x} \in \operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$, then

$$\operatorname{span}\{\mathbf{x}, \mathbf{v}_1, \ldots, \mathbf{v}_k\} = \operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}.$$

This means that some collections of vectors are suboptimal when it comes to describing a linear subspace — in fact, some subset of the vectors would have described the same linear subspace. This "suboptimality" is known as *linear dependence*. That is, we call a collection of vectors V *linearly dependent* if there exists a subset of $U \subset V$ $(U \neq V)$ for which $\operatorname{span}(V) = \operatorname{span}(U)$. If no such subset exists (the collection V is minimal in some sense), then we say that the vectors in V are *linearly independent* and refer to this collection as a *basis*.

One of the fundamental properties of linear subspaces of $\mathbb{R}^n$ is the fact that every basis has the same size. The number of vectors in a basis for a linear subspace is called its *dimension*. In other words,

Theorem 1. *The vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ form a basis if and only if $\operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ forms a k dimensional space.*

The reason a basis is better than a non-basis is that the description of a vector using a basis is *unique*. As a small example, let

$$\mathbf{x} = \mathbf{u} + 2\mathbf{v} - 3\mathbf{w} \quad \text{and} \quad \mathbf{y} = \mathbf{u} - 7\mathbf{v} + \pi\mathbf{w}$$

In the case that $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ form a basis, we can tell that $\mathbf{x} \neq \mathbf{y}$ just by looking at the coefficients and seeing they are different: $(1, 2, -3) \neq (1, -7, \pi)$. If $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ do not form a basis, then we don't know whether $\mathbf{x}$ and $\mathbf{y}$ are the same and this is annoying. If I solve a problem and get $\mathbf{x}$ and you solve the same problem and get $\mathbf{y}$, then we might thing one of us is wrong (when in fact we are both right).

One property of linear subspaces that is sometimes useful and sometimes annoying is that they must always contain the vector $\mathbf{0}$. So, for example, Let $S = \{(x, y) : x + y = 1\}$. S is *not* a linear subspace (it does not contain $(0, 0)$, even though the graph of S is line. Furthermore, any line going through $(0, 0)$ is a linear subspace, so the only difference between S and a legitimate linear subspace is that S has been shifted away from $(0, 0)$. Given a linear subspace X and a vector $\mathbf{b} \neq \mathbf{0}$, the collection of vectors

$$S = \{\mathbf{x} + \mathbf{b} : \mathbf{x} \in X\}$$

is called an *affine subspace*. The dimension of S is then said to be the same as the dimension of X (then linear subspace it was formed from).

1.1.1 Operations

First we define matrix–vector multiplication, which can be done in two ways. If we think of a matrix $\mathbf{M}$ as a collection of (column) vectors, then $\mathbf{Mv}$ forms a linear combination of these

vectors:

$$\begin{bmatrix} | & | & \dots & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n \\ | & | & \dots & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_i x_i \mathbf{v}_i.$$

On the other hand, if I think of a matrix as a collection of (rows) covectors, then $M\mathbf{v}$ forms a vector of dot products:

$$\begin{bmatrix} — & \mathbf{u}_1^\intercal & — \\ — & \mathbf{u}_2^\intercal & — \\ \vdots & \vdots & \vdots \\ — & \mathbf{u}_m^\intercal & — \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{u}_1 \cdot \mathbf{x} \\ \mathbf{u}_2 \cdot \mathbf{x} \\ \vdots \\ \mathbf{u}_m \cdot \mathbf{x} \end{bmatrix}$$

In particular, for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, we have

$$\mathbf{u}^\intercal \mathbf{v} = \mathbf{u} \cdot \mathbf{v}$$

and this way of writing the dot product will appear *a lot* in this class (so get used to it quickly). This is *not the same* as $\mathbf{u}\mathbf{v}^\intercal$ which would be an $n \times n$ matrix with entries $M_{ij} = \mathbf{u}_i \mathbf{v}_j$ (see (2) below). Fortunately, we will not see much of these in this class, so it should be easier to tell the difference (but this can sometimes cause you to forget that there is a difference).

Finally, we can think of matrix–matrix multiplication as a well-organized collection of matrix–vector multiplications:

$$\mathbf{M} \begin{bmatrix} | & | & \dots & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n \\ | & | & \dots & | \end{bmatrix} = \begin{bmatrix} | & | & \dots & | \\ \mathbf{M}\mathbf{v}_1 & \mathbf{M}\mathbf{v}_2 & \dots & \mathbf{M}\mathbf{v}_n \\ | & | & \dots & | \end{bmatrix}$$

or via the equivalent formula

$$[\mathbf{AB}]_{i,j} = \sum_k A_{ik} B_{kj}$$

Note that, in particular, for two matrices to be multiplied, the inner dimensions must match (the n in the equation below) and the outer dimensions become the dimensions of the new matrix (the m and the p):

$$\underbrace{\mathbf{A}}_{m \times n} \underbrace{\mathbf{B}}_{n \times p} = \underbrace{\mathbf{AB}}_{m \times p}$$

Or, repeating the examples above:

$$\underbrace{\mathbf{u}^\intercal}_{1 \times n} \underbrace{\mathbf{v}}_{n \times 1} = \underbrace{\mathbf{u} \cdot \mathbf{v}}_{1 \times 1} \quad \text{and} \quad \underbrace{\mathbf{u}}_{n \times 1} \underbrace{\mathbf{v}^\intercal}_{1 \times n} = \underbrace{\mathbf{u}\mathbf{v}^\intercal}_{n \times n} \tag{2}$$

1.1.2 Rank

Given an $m \times n$ matrix $\mathbf{M}$, the collection of vectors

$$\{\mathbf{M}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\}$$

is called the *image* of $\mathbf{M}$ and the collection of vectors

$$\{\mathbf{x} \in \mathbb{R}^n : \mathbf{M}\mathbf{x} = \mathbf{0}\}$$

is called the *kernel*. Both are linear subspaces. Using the terminology from the first section, the image of $\mathbf{A}$ is the span of the columns of $\mathbf{M}$.

$$\mathbf{M} = \begin{bmatrix} | & | & \dots & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n \\ | & | & \dots & | \end{bmatrix} \quad \Rightarrow \quad \mathrm{im}(\mathbf{M}) = \mathrm{span}\left\{\mathbf{v}_1, \dots, \mathbf{v}_n\right\}.$$

The dimension of the image of a matrix M is called its *rank*. One of the "magical" theorems of linear algebra is that every matrix (no matter what its dimensions) has the same rank as its transpose.

Theorem 2. *For all $n \times m$ matrices $\mathbf{A}$,*

$$\mathrm{rank}(\mathbf{A}) = \mathrm{rank}(\mathbf{A}^\intercal)$$

In particular, this means that, for an $m \times n$ matrix $\mathbf{A}$, $\mathrm{rank}(\mathbf{A}) \leq \min\{m, n\}$. When $\mathrm{rank}(\mathbf{A}) = \min\{m, n\}$, the matrix $\mathbf{A}$ is said to have *full rank*. Or, more specifically, it is said to have *full row rank* when $\mathrm{rank}(\mathbf{A}) = m$ and *full column rank* when $\mathrm{rank}(\mathbf{A}) = n$. One thing to remember about rank is that it is easy to lose it but impossible to get it back (once it is gone). In particular, if $\mathbf{A}$ and $\mathbf{B}$ are matrices for which $\mathbf{AB}$ makes sense, we have the inequality

$$\mathrm{rank}(\mathbf{AB}) \leq \min\{\mathrm{rank}(\mathbf{A}), \mathrm{rank}(\mathbf{B})\}$$

We will denote the $k \times k$ identity matrix as $\mathbf{I}_k$ (or simply $\mathbf{I}$ if it is clear what the dimensions are). A matrix $\mathbf{A}$ is called *invertible* if there exists a matrix $\mathbf{B}$ for which

$$\mathbf{AB} = \mathbf{I} = \mathbf{BA} \tag{3}$$

When such a $\mathbf{B}$ exists, we call it the *inverse* of $\mathbf{A}$ and use the notation $\mathbf{A}^{-1}$ to refer to it. Note that *both* equalities in (3) must hold for $\mathbf{B}$ to be a true inverse. It is possible to have

$$\mathbf{AB} = \mathbf{I} \quad \text{and} \quad \mathbf{BA} \neq \mathbf{I} \tag{4}$$

(for example) in which case $\mathbf{B}$ is not the inverse[1] of $\mathbf{A}$. One of the most important theorems of linear algebra is that one can tell whether a matrix is invertible simply by comparing its rank to its dimensions.

Theorem 3. *An $m \times n$ matrix $\mathbf{A}$ is invertible if and only if $m = n = \mathrm{rank}(\mathbf{A})$.*

In the case that $\mathbf{A}$ is invertible, we can solve systems of linear equations using the inverse:

$$\mathbf{Ax} = \mathbf{b} \quad \Rightarrow \quad \mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b} \quad \Rightarrow \quad \mathbf{Ix} = \mathbf{A}^{-1}\mathbf{b} \quad \Rightarrow \quad \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

In general, the set of solutions to a linear equation

$$\{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{b}\}$$

forms an affine subspace of $\mathbb{R}^n$ (a linear subspace if $\mathbf{b} = \mathbf{0}$) that has dimension $n - \mathrm{rank}(\mathbf{A})$. The case when $\mathbf{A}$ is invertible means $\mathrm{rank}(\mathbf{A}) = n$, so the space of solutions has dimension 0 (that is, it is a single point).

One of the important things to recall about inverses is that the product of two square matrices $\mathbf{A}$ and $\mathbf{B}$ is invertible if and only if $\mathbf{A}$ and $\mathbf{B}$ are both invertible. In this case, we have the formula

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$$

which follows directly from the definition of inverse.

[1] A matrix $\mathbf{B}$ that satisfies (4) is sometimes called the *right-inverse* or *pseudo-inverse* but we will not use these terms in this class.

1.1.3 Hyperplanes

An affine subspace $H \subseteq \mathbb{R}^n$ with $\dim(H) = k$ is said to have *codimension* $n - k$. Of particular interest in this course will be affine subspaces of dimension $n - 1$ (codimension 1), which we will refer to as *hyperplanes*. These are formed (as discussed above) by taking a linear subspace of dimension $n - 1$ (codimension 1) and shifting by a vector. As you perhaps recall, a linear subspace of codimension 1 can be described using its *normal vector*. That is, for a given vector $\mathbf{v} \in \mathbb{R}^n$ with $\mathbf{v} \neq \mathbf{0}$, the set

$$H = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \cdot \mathbf{v} = 0\}$$

is a linear subspace of codimension 1 and all linear subspaces of codimension 1 living inside of $\mathbb{R}^n$ can be written in this way.

A hyperplane is then a shift of one of these linear subspaces. That is, for two given vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ with $\mathbf{u}, \mathbf{v} \neq \mathbf{0}$, a hyperplane in $\mathbb{R}^n$ can be written in the form

$$H' = H + \mathbf{u} = \{\mathbf{x} + \mathbf{u} \in \mathbb{R}^n : \mathbf{x} \cdot \mathbf{v} = 0\}.$$

However, this is not the form you will usually see them. Instead, someone will substitute $\mathbf{y} = \mathbf{x} + \mathbf{u}$ (recall $\mathbf{u}$ is fixed) to get

$$\begin{aligned} H' &= \{\mathbf{x} + \mathbf{u} \in \mathbb{R}^n : \mathbf{x} \cdot \mathbf{v} = 0\} \\ &= \{\mathbf{y} \in \mathbb{R}^n : (\mathbf{y} - \mathbf{u}) \cdot \mathbf{v} = 0\} \\ &= \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y} \cdot \mathbf{v} = t\} \end{aligned}$$

where $t = \mathbf{u} \cdot \mathbf{v}$ is a fixed number (since $\mathbf{u}$ and $\mathbf{v}$ are fixed vectors).

That is, given a fixed vector $\mathbf{v}$ and fixed real number t, the set

$$H' = \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y} \cdot \mathbf{v} = t\}$$

forms a hyperplane in $\mathbb{R}^n$ and all hyperplanes in $\mathbb{R}^n$ can be written in this way. Geometrically, the $\mathbf{v}$ will again be the normal vector to H', with t now telling you how far H' has been shifted in the direction of $\mathbf{v}$.

1.1.4 References

1. Bertsimas and Tsitsiklis: Section 1.5
2. Your linear algebra class from your first semester at EPFL![2]

1.2 Intro to Optimization

At its heart, every optimization problem looks like the following:

$$\max\{f(x) : x \in X\} \tag{5}$$

where f is some function (called the *objective function*) and X is some set of things (called the *feasible region*) you are allowed to plug into f. The points in X are called *feasible points* and the goal is to find a feasible point $x_0 \in X$ for which

$$f(x_0) \geq f(x) \text{ for all } x \in X.$$

Some comments:

[2]The second semester (while super useful and interesting for other reasons) is less relevant to this class.

- Points that are not in X are called *infeasible points*.
- A "largest value" may not exist (this can happen for multiple reasons).
- In the form that it is written, the only possible way to solve this would be to plug in every possible value of $x \in X$ and see which one gives the biggest $f(x)$. In particular, if X is infinite, there is essentially no way to solve this problem (in general).

Fortunately, we are not in the business of trying to optimize "every possible function" over "every possible set X." Typically, we know something about f and X that can make optimization easier (in particular, not impossible). For example, we are sometimes told that f is *continuous* or *differentiable*, in which case we can perform a "hill-climbing algorithm" to try to find the optimal point (at a given point $\mathbf{x}$, take a step in the direction that causes the objective to go up as much as possible). Similarly, we might have a nice description of X like

$$X_1 = \left\{\mathbf{y} \in \mathbb{R}^3 : \|\mathbf{y}\| \leq 1\right\} \quad \text{or} \quad X_2 = \left\{\mathbf{y} \in \mathbb{Z}^3 : \|\mathbf{y}\| \leq 20\right\}.$$

If the feasible region is defined by a set of inequalities, those inequalities are called *constraints*. Despite the only difference between X_1 and X_2 being a change from $\mathbb{R}$ to $\mathbb{Z}$, the two domains are actually quite different, and optimizing over X_2 will be much harder than optimizing over X_1 (for reasons we will see).

It is here that I should mention that there is some strange terminology that is used when it comes to optimization. While we typically think of "solution" as being the answer to a problem, the word *solution* is used in this field to mean "anything that you can to try to plug into your objective function (whether it is feasible or not). Hence we will talk about *feasible solutions* to be those that satisfy the constraints and *infeasible solutions* to be those that do not. Since we are no longer using the word "solution" to talk about the answer to a problem, then we need a new word for this. In fact, depending on the optimization problem, the answer can take one of three forms:

1. There exists an *optimal solution*, which is a feasible solution $\mathbf{x}_*$ that satisfies $f(\mathbf{x}_*) \geq f(\mathbf{x})$ for all $\mathbf{x} \in S$.
2. The problem is *unbounded* — that is there exist feasible solutions for which $f(\mathbf{x})$ can be arbitrarily large.
3. The problem is *infeasible*, which happens when S is an empty set (there are no feasible solutions).

1.2.1 Convex Spaces

There are essentially two types of spaces when it comes to optimizing: convex and non-convex.[3]

A space X is called *convex* if you can get from point $A \in X$ to a point $B \in X$ by following a straight line that never leaves X. Mathematically, X is convex if for all pairs of points $\mathbf{x}, \mathbf{y} \in X$ and all $\lambda \in [0, 1]$ the point

$$(1 - \lambda)\mathbf{x} + \lambda\mathbf{y} \in X$$

Notice that when $\lambda = 0$, we are at $\mathbf{x}$ and when $\lambda = 1$, we are at $\mathbf{y}$, and for λ in between we get a line connecting them. So this is *exactly* how we described it. The reason convex spaces are nice is that the boundaries (if there are any) actually mean something.

[3]Note that we are using the word "convex" in a different meaning then when someone talks about a function f being "convex"). The "convexity" we will be considering here is a (geometric) property of sets (not functions). However, there is a link between the two: if a function f is convex (in the function sense) then the sets $\{\mathbf{x} : f(\mathbf{x}) \geq c\}$ are each convex (in the geometric sense) for every $c \in \mathbb{R}$.

For example, consider the space X_1 above. The point $\mathbf{y} = \begin{bmatrix}1\\0\\0\end{bmatrix}$ is a feasible point, but it is also on the boundary (so it is next to some infeasible points). In particular, if you were to try to move in the direction $\begin{bmatrix}0\\1\\0\end{bmatrix}$ a tiny bit, so

$$\mathbf{y}' = \begin{bmatrix}1\\0\\0\end{bmatrix} + 0.001 \begin{bmatrix}0\\1\\0\end{bmatrix}$$

you would notice $\mathbf{y}'$ is infeasible. From this, you can conclude that

$$\mathbf{y}_t = \begin{bmatrix}1\\0\\0\end{bmatrix} + t \begin{bmatrix}0\\1\\0\end{bmatrix}$$

is infeasible for all $t \geq 0.001$. That is, if you are walking on a line through a convex space, and you cross the boundary into infeasible territory, you will *never* get back into feasible territory again. However this is not the case in the space X_2 from above.

$$\mathbf{y}' = \begin{bmatrix}1\\0\\0\end{bmatrix} + 0.001 \begin{bmatrix}0\\1\\0\end{bmatrix}$$

is not in X_2, but

$$\mathbf{y}'' = \begin{bmatrix}1\\0\\0\end{bmatrix} + 17 \begin{bmatrix}0\\1\\0\end{bmatrix}$$

is in X_2.

1.2.2 Hill climbing intuition (and Lagrange multipliers)

In this class, a number of the most useful things we will discover will come from thinking about optimization geometrically. For example, consider the hill-climbing algorithm that was mentioned above. I like to pretend that I am just floating in the space X (and I am happy to just float). There is an objective function which assigns a value to each point in the space, but I don't see those values (or care). HOWEVER, there is an external "being" that sees those values and wants to push you in a direction that will increase the objective function value (like your parents, when they keep suggesting how nice it would be to have a doctor in the family). If f is the objective function (let's assume it is differentiable), then at a given point $\mathbf{x}$, the amount of force that f will apply is equal to $\nabla f(\mathbf{x})$. A force will be applied everywhere in the space (though it may be a different size or direction at different points) and typically this force will cause me to move (and as a result, the objective function will increase). However, if I am at a local maximum (that is, there are no feasible points around me where the objective function is higher than it is at my current spot), then I won't move (I can't move).

Physics has a very nice theory about the relationship between forces and movement — an object moves in the direction of the sum of the forces applied to it. In particular, there is only one way for an object to be at rest — the sum of the forces being applied to it have to equal $\mathbf{0}$. So that must be the case here (assuming I am not moving). It is possible that $\nabla f(\mathbf{x}) = \mathbf{0}$

all by itself (in which case, this would be a normal critical point). The only other possibility is that there are other forces acting on the object that are pushing back against the force that ∇f is applying. Those forces can only come from one place: the boundary of the space. As an example, consider the fact that you are currently under a constant force (gravity) and yet you are not falling towards the center of the earth. The reason gravity isn't moving you when you stand up is that floor is asserting a force on your feet that is equal strength (but opposite direction) to the gravity is. Depending on how you are standing, that "antiforce" could be pushing equally on both feet or it could be pushing different amounts.

So then you might ask, "What force could the boundary possibly be putting on me?" As it turns out, we can figure out the direction of such a force — a piece of boundary defined by the equation $g_i(\mathbf{x}) = 0$ applies its force at a point $\mathbf{x}$ in the direction of $\nabla g_i(\mathbf{x})$ — but we don't necessarily know the amount (so we give it a variable λ_i). In order for these boundary forces to cancel out the force the objective is putting on me, it must be the case that $\nabla f(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \nabla g_i(\mathbf{x})$, which should look familiar: it is what you likely learned as "the method of Lagrange multipliers". The theory of Lagrange multipliers says

- If I am at a point $\mathbf{x}$ and the force $\nabla f(\mathbf{x})$ is able to push me to a new point, then that new point will have a higher value of f (the gradient of a function points in the direction of maximum increase)
- Therefore, the contrapositive is true — if I am at a point $\mathbf{x}$ which maximizes f (over the set X), it must mean that (even though $\nabla f(\mathbf{x})$ is being applied to me) I am not moving.
- If I am not moving, it must be that the sum of the forces on me in $\mathbf{0}$. That leaves two possibilities: either $\nabla f(\mathbf{x}) = \mathbf{0}$ (meaning this is a normal critical point), or something (the boundaries) are pushing back with forces that cause the sum of forces to be $\mathbf{0}$.
- I know the direction that boundaries can push: $\nabla g_i(\mathbf{x})$, but I don't know the amount. So the best I can say is that the total force exerted by the boundary has the form $\lambda_i \nabla g_i(\mathbf{x})$ for some $\lambda_i \geq 0$.
- Therefore (in order to add up to $\mathbf{0}$), it must be that $\nabla f(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \nabla g_i(\mathbf{x})$ for some $\lambda_i \geq 0$.

However, there is one detail that, on the surface, seems so obvious that many math books leave it out. When it comes to Lagrange multipliers, however, it can be *super* useful.

- A piece of the boundary can only apply a force against me if I am touching it!

That is, if the boundary is defined by an inequality constraint $g_i(\mathbf{x}) \leq 0$, the only time λ_i is allowed to be nonzero is if $g_i(\mathbf{x}) = 0$!

Your math book might have avoided this part of the discussion by separating into cases of "equality constraints" and "inequality constraints" and then said something like "and then you just have to check all of the points in the boundary of the space defined by the inequality constraints." And while this is (technically) correct, it is hiding a LOT of potential complications in it. For example,

- It might not be easy to check all of the points of a boundary.
- It might not be easy to even find the boundary.

However things we *do* know about the boundary can make our lives simpler. Back to the gravity example, let's assume that I have constraints that describe a weird room with four walls

$(g_1, \dots g_4)$, a floor (g_5) and a slanted ceiling $(g6)$. Because the ceiling is slanted, $g5$ and $g6$ will intersect (at some point), but they will not intersect in the feasible region (there will be walls separating them). So while Lagrange multipliers will tells you to look for points where

$$\nabla f(\mathbf{x}) = \sum_{i=1}^{6} \lambda_i \nabla g_i(\mathbf{x})$$

it also tells you that you can ignore any points where $\lambda_5 \neq 0$ and $\lambda_6 \neq 0$ at the same time (there is no way for the floor and the ceiling to push on you at the same time.

This makes inequality constraints rather complicated — depending on how they intersect geometrically, some combinations may give false Lagrange multiplier solutions (see the problem set).

1.2.3 Inequality constraints are complicated

The main issue with inequality constraints is that their relevance will often depend on the point you referring to. Read the inequality $x \leq 7$ out loud and notice how you say "less than OR equal to." Some values of x satisfy $x \leq 7$ with equality and other values satisfy it without equality. Given a feasible point $\mathbf{x} \in X$, we say a constraint is *active* (or *tight*) at $\mathbf{x}$ if it holds with equality and *inactive* (or *not tight*) otherwise. For example, consider the constraints

$$(a)\ x + y \leq 12 \quad \text{and} \quad (b)\ x^2 + y^2 \leq 122$$

Then

- $\begin{bmatrix} 3 \\ 3 \end{bmatrix}$ is feasible with (a) and (b) inactive
- $\begin{bmatrix} 4 \\ 8 \end{bmatrix}$ is feasible, with (a) active and (b) inactive
- $\begin{bmatrix} 1 \\ 11 \end{bmatrix}$ is feasible, with (a) and (b) active
- $\begin{bmatrix} 4 \\ 9 \end{bmatrix}$ is infeasible

Note that an equality constraint is always considered to be active (at a feasible point). The reason that it is good to know whether a constraint is active at a given point goes back to the previous section — active constraints are the only ones that can apply a counteracting force. So in this new language, our conclusion in the previous section can be stated: if we are at a point $\mathbf{x}_*$ which maximizes $f(\mathbf{x})$ over some domain defined by the inequalities $g_i(\mathbf{x}) \geq b_i$, then it must be that $\nabla f(\mathbf{x}_*) = \sum_{i=1}^{n} \lambda_i \nabla g_i(\mathbf{x}_*)$ for some $\lambda_i \geq 0$ where the only $\lambda_i \neq 0$ are those for which g_i is active at $\mathbf{x}_*$ (so $g_i(\mathbf{x}_*) = b_i$).

This is why understanding the geometry of the space X is important. In theory if I had 200 inequality constraints, I could have 2^{200} possible combinations of active/inactive constraints. That is a lot of "checking the boundary." However some constraints can never be active together — if my feasible space was the room I am sitting, the constraints would be the floor, the walls, and the ceiling. But the floor does not touch the ceiling, so they can't be active at the same time. In fact, a rectangular room only has 8 corners, 12 edges, and 6 faces (a total of 26 feasible combinations, instead of $2^6 = 64$). Our goal, then, will be to pick a collection of spaces which are somewhat versatile (so we can potentially solve many different types of problems) but which are simple enough that we can understand the geometry (these will turn out to be *polytopes*).

1.2.4 References

1. Mathematica examples

1.3 Complexity

If there is one thing I can convince you of in this course, it is the fact that complexity is a complicated matter. The reason is that there is no standard way of saying "how good an algorithm is" so we have a bunch of different ways to try to characterize the goodness of a given algorithm. To be clear, I am only talking about *deterministic* algorithms — which is the only type of algorithm we will see in this class.[4] In this class, we will primarily use *big-O* notation to describe complexity relationships. Specifically, for two functions $f(n)$ and $g(n)$, we will say

$$f(n) = O(g(n))$$

to mean that there exists a constant c and number n_0 such that

$$f(n) \leq cg(n)$$

whenever $n \geq n_0$. In layman's turn, big-O notation is used to compare the "growth rates" of functions with the caveat that n could be quite large before the true behavior kicks in.

1.3.1 Worst case complexity

The "gold standard" in computer science is *worst case complexity*. This is also the most common type of way to talk about algorithms, so typically if someone just says "this algorithm runs in $O(n^{4/3})$ time", you should assume they are talking about worst case complexity. Worst case complexity is exactly how it sounds — if you have an algorithm, then I will run the algorithm on all possible inputs of a given size, and whichever one is the slowest is the worst case complexity. Of course it is impossible to run an algorithm on all possible inputs, so instead complexities are typically found using a proof[5] In other words, it is a guarantee — if you run this algorithm on an input of size n, then I can guarantee the algorithm will finish in fewer than $f(n)$ steps.

And this is why these are the "gold standard" — because they give you a guarantee. That said, the way it is stated (using big-O notation) is not a particularly great guarantee because we don't know the value of c. A guarantee that a given algorithm will finish in at most $2n^2$ steps is much different than a guarantee that it will finish in at most $10^{100}n^2$ steps. In fact, a guarantee of at most $2n^3$ steps is often much better than a guarantee of $10^{100}n^2$ (even though big-O notation makes the second one look better). But this is a whole separate issue — what is important (at the moment) is that when we say such things, we are talking about a worst case scenario.

And worst case complexity is only really relevant if you keep coming across problems that cause the algorithm to do poorly. It is possible for an algorithm to have very bad worst case complexity (exponential in the size of the input), but the only way to get such a running time is to construct a particular example that forces this particular algorithm to make bad decisions at all times. So the fact that your algorithm *could* take a really long time to finish is not really representative of its general usefulness.

[4] This is in contrast to *randomized* algorithms where you allow for decisions to be made using the random number generator.

[5] And these proofs often use the idea of an "adversary" (so if you hear someone talking about an adversary, they are likely talking about worst case complexity.

1.3.2 Real life complexity

And if we are talking about "real life" scenarios, there is another aspect that worst case complexity fails to capture and that is what I call the "upper bound phenomenon". The "upper bound phenomenon" is the fact that there is always an implicit upper bound on the amount of time an algorithm can run. So a lot of times you never even get to see the "worst case scenario" because you've given up at that point.

Imagine you were a farmer and you had an algorithm that took in all of the weather conditions and calculated the optimal amount of water to give your plants in the morning. Every night, you could plug in the weather forecast before you go to sleep and then (hopefully) do what the algorithm tells you to do the next morning. So you had to choose between an algorithm that always took 2 days to finish and an algorithm that finished in 4 hours 99% of the time and took 20 years the other 1% of the time, which do you think you should choose? Yes, in terms of worst case complexity, the first algorithm is *much* better. The problem is that, for your use case, 2 days and 20 years are both equally useless. So the first algorithm is useless 100% of the time and the second one is useless 1% of the time. But worst case complexity cannot capture this.

And you would be surprised how quickly things become useless. In some sense an $O(n^8)$ algorithm is a *huge* improvement on an $O(2^n)$ algorithm (one is in `P` and the other is in `NP`), but if the n I am interested in is anything bigger than 100, then it is quite possible that both of these algorithms are equally useless. The point is that having small worst case complexity is good, but that does not mean that having large worst case complexity is bad. It just means that it *could be* bad. And we are going to run across this theme multiple times in this class, so I wanted to plant the seed now.

1.3.3 References

1. Bertsimas and Tsitsiklis: Section 1.6

2 Week 2: Polyhedra and Linear Inequalities

We will now start talking about a specific optimization problem called a *linear program*. This is the situation where the objective function is linear — that is, $f(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x}$ for some vector $\mathbf{c}$ — and where the constraints are linear inequalities (so have the form $\mathbf{a} \cdot \mathbf{x} \geq b$ for some vector $\mathbf{a}$ and some real value b. As we saw in the previous section, knowing the geometry of the feasible region is an important factor for solving optimizations problems, so we start by trying to understand the types of sets that can be formed using these linear inequality constraints.

2.1 Geometry of linear programs (polyhedra)

Geometrically, a linear inequality corresponds to what is known as a *half space* — that is, it consists of an affine hyperplane and all of the points on one side of that hyperplane. The feasible region for a linear program is a finite intersection of half spaces, which is known as a *polyhedron*. As we discussed previously, a "nice" feasible region is one that is convex, so the first thing to do is to determine whether these polyhedra are convex (or not). On the problem set, we showed that the intersection of convex sets is convex, so in order to show that a polyhedron is convex it suffices to show that a half-space is convex. In class, we showed that this is true:

Theorem 4. *Half-spaces are convex (see the video[6] for the proof).*

One thing we did see in the proof was that things depended very heavily on the fact that certain numbers were positive. This is because inequalities (unlike equalities) change if you multiply by a negative number. So this means a lot of things that we are used to from linear algebra will become slightly different in the inequality context. For example, in linear algebra you had *linear combinations*

$$\sum_i a_i \mathbf{v}_i \quad \text{where } a_i \in \mathbb{R}$$

which will become *nonnegative linear combinations*

$$\sum_i a_i \mathbf{v}_i \quad \text{where } a_i \in \mathbb{R}, a_i \geq 0$$

or *convex combinations*

$$\sum_i a_i \mathbf{v}_i \quad \text{where } a_i \in \mathbb{R}, a_i \geq 0, \sum_i a_i = 1$$

This leads to different versions of *span*:

- all linear combinations $\Longrightarrow$ the span of the vectors
- all nonnegative linear combinations $\Longrightarrow$ the *cone* of the vectors
- all convex combinations $\Longrightarrow$ the *convex hull* of the vectors

The convex hull of a finite number of points is a *polytope* (a polytope is a polyhedron that is also bounded). And this brings up an interesting point — if I have a polytope that I want to talk about, there are two ways I can describe that polytope:

- as the convex hull of some collection of points
- as the intersection of half-spaces

[6] Class 2, Part 1, 26:00

The interesting thing is that the two descriptions are really *very* different. By that, I mean that if I am doing something that requires me to know the half-space description of a polytope and I only have the convex hull description, I could be in trouble. In general, getting one description from the other is an NP-hard problem, so there may be some approaches to solving linear programs that make sense geometrically, but just do not translate into math well because we don't have the right information available.

2.1.1 References

1. Bertsimas and Tsitsiklis: Section 2.1

2.2 Algebra of linear programs (linear inequalities)

The biggest difference between linear algebra and linear programming is the replacement of equalities with inequalities. For example, you used to solve things like

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{6}$$

whereas now we will try to solve things like

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}. \tag{7}$$

The word "solve" is perhaps not the best word to use... "find all possible solutions to" is more accurate. Recall that in (6), the types of solution sets we would get were pretty simple. Either

1. a solution does not exist, or
2. a unique solution exists, or
3. an infinite set of solutions exists, and these form an affine subspace (which we could describe using basis vectors)

In (7), we will find there are *many* more types of situations that arise. But before that, we should at least define what (7) even means and to do so we will start with (6).

Recall that one way to think about matrix multiplication (the "covector" perspective) was

$$\begin{bmatrix} — & \mathbf{u}_1^\intercal & — \\ — & \mathbf{u}_2^\intercal & — \\ \vdots & \vdots & \vdots \\ — & \mathbf{u}_m^\intercal & — \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{u}_1 \cdot \mathbf{x} \\ \mathbf{u}_2 \cdot \mathbf{x} \\ \vdots \\ \mathbf{u}_m \cdot \mathbf{x} \end{bmatrix}$$

Hence, for example, if

$$\mathbf{A} = \begin{bmatrix} — & \mathbf{u}_1^\intercal & — \\ — & \mathbf{u}_2^\intercal & — \\ — & \mathbf{u}_3^\intercal & — \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

then $\mathbf{A}\mathbf{x} = \mathbf{b}$ corresponds to the system of linear equations

$$\begin{aligned} \mathbf{u}_1 \cdot \mathbf{x} &= b_1 \\ \mathbf{u}_2 \cdot \mathbf{x} &= b_2 \\ \mathbf{u}_3 \cdot \mathbf{x} &= b_3. \end{aligned}$$

Algebraically, we can simply do the same thing:

$$\begin{bmatrix} — & \mathbf{u}_1^\intercal & — \\ — & \mathbf{u}_2^\intercal & — \\ — & \mathbf{u}_3^\intercal & — \end{bmatrix} \mathbf{x} \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

corresponds to the system of linear *inequalities*

$$\begin{aligned} \mathbf{u}_1 \cdot \mathbf{x} &\geq b_1 \\ \mathbf{u}_2 \cdot \mathbf{x} &\geq b_2 \\ \mathbf{u}_3 \cdot \mathbf{x} &\geq b_3. \end{aligned}$$

To see what is happening geometrically, recall that each equation $\mathbf{u}_i \cdot \mathbf{x} = b_i$ forms an affine hyperplane. To "solve for x" then means to find a point (or points) $\mathbf{x}$ which lie in the intersection of all of these hyperplanes. Inequalities like $\mathbf{u}_i \cdot \mathbf{x} \geq b_i$ form what are called *half spaces*. They consist of an affine hyperplane (the border) and then everything on one of the two sides of that hyperplane. So for example, the system

$$\begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x} \geq \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

forms a triangle in the plane (which was impossible when we just had equalities).

2.2.1 When inequalities get all over the floor

To be clear, the idea to "just replace the $=$ with a $\geq$" idea in the previous section is the mathematical equivalent of telling someone to "just feed the baby" — it is easy to accomplish when you have a really nice, quiet, well-behaved baby, but not all babies are nice, quiet, or well-behaved. Some of them throw their food on the floor and laugh at you. Some of them close their mouth right when you are about to put the spoon in, so the food falls onto their clothes (and then they laugh at you). A "generic baby" can often require some amount of creativity in order to get it to eat — I am told that pretending the spoon is an airplane that needs to come in for a landing is a good technique, but I have no experience in the matter. HOWEVER, I do have some experience in getting a "generic system of linear inequalities" to behave itself and that is what this section is about.

Firstly, what do I mean by a "generic system of linear inequalities"? We will consider 6 different "types" of linear inequalities (here $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{x} = [x_1, \ldots, x_n]^\intercal$ are variables):

(1) $\mathbf{v} \cdot \mathbf{x} \geq b$

(2) $\mathbf{v} \cdot \mathbf{x} \leq b$

(3) $\mathbf{v} \cdot \mathbf{x} = b$

(4) $x_i \geq 0$

(5) $x_i \leq 0$

(6) x_i is free

where a "generic system" can include any combination of linear inequalities of each type.

Note that $x_i = 0$ is not there because that would make life too easy (as soon as we know the value of x_i, we can just plug it in!). The meaning of type (6) is essentially the statement that neither (4) nor (5) need to hold (in the solution to this problem, x_i can be any real number, not just a positive/negative one). In that sense, (6) is more of a "lack of inequality" than an inequality itself, but nonetheless we call it a type. Finally, note also that (4) is a special subcase of (1) and (5) is a special subcase of (2). The reason we differentiate them is that (4) and (5) are in "diagonalized form" (which, as we know from linear algebra is a particularly nice form).

The first thing to notice is that 6 forms is a lot of forms, since presumably our algorithm[7] will have to treat each one in a different way. So our first goal is to show that we really don't need all 6 of these types — that any linear program can be transformed into an equivalent one that is easier to deal with. Optimization problems $\mathcal{P}$ and $\mathcal{P}'$ are said to be *equivalent* if for every feasible solution $\mathbf{x}$ for $\mathcal{P}$, there exists a feasible solution $\mathbf{x}'$ for $\mathcal{P}'$ such that

1. $\mathbf{x}$ and $\mathbf{x}'$ have the same cost (in the corresponding objective functions)
2. $\mathbf{x}$ can be easily constructed from $\mathbf{x}'$ (and the reverse is true as well)

Note that nothing here requires the mapping $\mathbf{x} \mapsto \mathbf{x}'$ to be $1-1$ or even symmetric — it may be that $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{P}$ have the same cost, and that both correspond to $\mathbf{x}' \in \mathcal{P}'$. Similarly, one does not need to be able to construct *all* of the reverse mappings — given $\mathbf{x}'$, one might be able to construct $\mathbf{x}_1$ but not $\mathbf{x}_2$ (that is OK, too). The whole goal here is to ensure that

- $\mathcal{P}$ and $\mathcal{P}'$ have the same optimal value
- Given an optimal solution to $\mathcal{P}$, we can easily construct an optimal solution to $\mathcal{P}'$ (and vice versa).

For reasons we will see later, there are actually two types of "standard forms" that we will aim to achieve.

1. Equality standard form: only type (3) and (4) allowed
2. Inequality standard form: only type (1) allowed

How can we "get rid of" inequalities of a given type? We can't — but we can transform them into other types in a variety of ways:

- turn a (2) into a (1)

  Multiply both sides of the inequality by -1 (this flips the sign). For example,

  $$2x + 3y - 7z \leq 2 \quad \Longrightarrow \quad -2x - 3y + 7z \geq -2$$

- turn a (3) into a (1) and (2)

  Use the fact that $x = y$ is equivalent to $x \leq y$ and $x \geq y$. For example,

  $$2x + 3y - 7z = 2 \quad \Longrightarrow \quad 2x + 3y - 7z \geq 2 \quad \text{and} \quad 2x + 3y - 7z \leq 2$$

  Notice that we now have more constraints than we did before (and that's OK)!

[7] While we haven't even started discussing an algorithm yet, it should always be in the back of our minds as the end product.

- turn a (4) into a (1) or a (5) into a (2)

  Since (4) and (5) are special cases of (1) and (2), we can do this by picking a vector that has a single 1 and the rest 0. So for example,

$$x_2 \geq 0 \quad \Rightarrow \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \cdot \mathbf{x} \geq 0$$

- turn a (6) into two (4)'s

  This is a trick that you should remember, because we will see it again. The idea is to decompose a variable into its positive and negative parts and then force those parts to be positive/negative. For example, if we have the constraints

$$2x + 7y = 4 \quad \text{and} \quad x \geq 0 \quad \text{and} \quad y \text{ free}$$

  then we can decompose y into y_+ and y_- and write $y = y_+ - y_-$ where y_+ and y_- must both be nonnegative. Hence we would get the (new) constraints

$$2x + 7y_+ - 7y_- = 4 \quad \text{and} \quad x \geq 0 \quad \text{and} \quad y_+ \geq 0 \quad \text{and} \quad y_- \geq 0.$$

  Two things to note here: firstly, we now have more (and different) variables than we did before (that's OK as long as the resulting inequalities are equivalent). There is still a small argument one needs to make to show that the corresponding LPs are equivalent (this is left for the reader).

- turn a (1) into a (3) and a (4)

  This is the other trick that you should remember because we will see it again. The idea is to introduce what is called a *slack variable*. You can think of a slack variable as a tool for measuring how close an inequality is to an equality. For example, the inequality $x \leq y$ is satisfied by the two points

$$(x, y) = (1, 2) \quad \text{and} \quad (x, y) = (4, 17,000)$$

  but the first solution is much closer to an equality than the second one (there is less slack).

  The nice thing about slack variables is that there is a super easy way to introduce them, which we show by example. Imagine you start with a type (1) equation

$$2x + 7y \geq 4$$

  and put all of the nonzero stuff on the same side

$$2x + 7y - 4 \geq 0.$$

  This looks like a (4) except for the whole "$2x + 7y - 4$" thing, so introduce a new variable w and set $2x + 7y - 4 = w$. So now you have

$$2x + 7y - 4 = w \quad \text{and} \quad w \geq 0$$

  which, we can then turn into a (3) and a (4) by rearranging:

$$2x + 7y - w = 4 \quad \text{and} \quad w \geq 0.$$

  Note that again we introduce a new variable (that's OK).

2.2.2 Success!

So what have we accomplished?

Theorem 5. *Any linear program with constraints of type (1)–(6) above can be turned into an equivalent linear program with constraints of type (1) only, which can then be written in the form*

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}$$

for some matrix $\mathbf{A}$*, some vector* $\mathbf{b}$ *and some vector of variables* $\mathbf{x}$*.*

Secondly,

Theorem 6. *Any linear program with constraints of type (1) – (6) above can be turned into an equivalent linear program with constraints of type (3) and (4) only, which can then be written in the form*

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

for some matrix $\mathbf{A}$*, some vector* $\mathbf{b}$ *and some vector of variables* $\mathbf{x}$*.*

Keep in mind that (in both cases), each of three of $\mathbf{A}, \mathbf{b}, \mathbf{x}$ that appears in the final representation could be quite different from what was in the original system of inequalities (that's OK, as long as the resulting linear programs are equivalent).

Note that (in theory) we have not made any progress whatsoever — we did a lot of work to take a problem (that we can't solve yet) and to rewrite it in a different form (that we can't solve yet). But (in practice) this is a success because each of these forms will be useful when it comes to the actual solving process. One obvious benefit of the first form (for example) is that it looks exactly like the "intersection of half spaces" geometric perspective that we had earlier. An obvious benefit of the second form, on the other hand, is that it is as close to a linear algebra problem as we could possibly hope to get (all of the "inequality-ness" has been pulled out of the matrix and turned into simple inequalities).

2.2.3 References

1. Bertsimas and Tsitsiklis: Section 1.1

3 Week 3: Vertices, Extreme Points, BFSs

3.1 Vertices, extreme points, and basic feasible solutions (Oh, my!)

We started class by noticing that linear programs had a tendency to end in the "corner" of a polyhedron (at least the ones created by dropping a ball inside a box). This was not *always* true — for example, if the "edge" of the box was parallel to the ground, it was possible to get it to stay in the middle of the "edge", but in that case it was tied with a "corner" when it came to the objective function. So we conjectured:

Conjecture 7. *For every vector* $\mathbf{c}$ *and every polyhedron* P*, there exists a "corner"* $\mathbf{y}$ *of* P *for which* $\mathbf{c} \cdot \mathbf{y} \geq \mathbf{c} \cdot \mathbf{x}$ *for all* $\mathbf{x} \in P$*.*

In particular, this allowed for the possibility that a non-"corner" vector $\mathbf{z}$ could have $\mathbf{c} \cdot \mathbf{y} = \mathbf{c} \cdot \mathbf{z}$ and this is what (we conjectured) is what happens when the ball gets stuck in the middle of the edge (the point it gets stuck at is $\mathbf{z}$).

Of course to prove something like this, we can't just wave our hands and say "you know, a *corner*" — we would need a definition that would mathematically characterize what we intuitively thought a "corner" of a polyhedron should be. In particular, it should be something that allows us to test any point in P and then tells us (definitively) whether that point should be considered a "corner". In fact, it will be useful to have 3 definitions of "corner":

- an "optimization" definition (which we call a *vertex*)
- a "geometric" candidate (which we call an *extremal point*)
- an "algebraic" candidate (which we call a *basic feasible solution (BFS)*)

At this point, you might be concerned that we would need to yell and scream and possibly fight over which definition is the best one for what we are trying to do, but (as it turns out), all three definitions are equivalent (making them all equally good definitions of a "corner"). That is,

Theorem 8. *Let* P *be a polyhedron defined by* $\mathbf{Ax} \geq \mathbf{b}$ *and* $\mathbf{x}$ *a point in* P*. Then*

$$\mathbf{x} \textit{ is a vertex} \iff \mathbf{x} \textit{ is an extremal point} \iff \mathbf{x} \textit{ is a BFS}$$

We will prove 2/3 of this statement below [8] (the final 1/3 — that "$\mathbf{x}$ is a vertex $\Rightarrow$ $\mathbf{x}$ is an extremal point" is part of Problem Set 3). It is worth noting that proving the conjecture would be noticable progress because it would effectively reduce a (potentially) infinite problem to a (potentially) finite one. The reasoning is that, since we are in $\mathbb{R}^n$, if a polytope was defined by m constraints, there would be at most $\binom{m}{n}$ possible BFS's and checking each one of those would take a finite (long, but finite) amount of time. Without this conjecture, we could potentially need to check every point in the polyhedron, something that would take a (much) longer time.

3.1.1 Proof: Equivalence of Vectors, Extreme Points, and Basic Feasible Solutions

Now is when we start to look more like a math class (with definition and theorems and proofs and stuff). First the definitions: given a polyhedron $P \subseteq \mathbb{R}^n$, a point $\mathbf{x} \in P$ is called

1. a *vertex* if there exists a linear program for which $\mathbf{x}$ is a unique solution. That is, there exists a vector $\mathbf{c}$ such that $\mathbf{c} \cdot \mathbf{x} > \mathbf{c} \cdot \mathbf{y}$ for all $\mathbf{y} \neq \mathbf{x} \in P$ (note that is equivalent to find a $\mathbf{w}$ for which $\mathbf{w} \cdot \mathbf{x} < \vec{e} \cdot \mathbf{y}$, since we can make $\mathbf{c} = -\mathbf{w}$.

[8] See also the video from Class 3, Part 1 at 34:20.

2. an *extreme point* if it is not the convex combination of two other points in P. That is, the equation $\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{z}$ has no solution satisfying $\mathbf{y}, \mathbf{x} \neq \mathbf{x}$ and $\lambda \in [0,1]$.

3. a *basic feasible solution (BFS)* if it is the unique point of intersection of a set of constraint hyperplanes. In other words, if there exist n linearly independent constraint vectors that are *active* at $\mathbf{x}$. Note that equality constraints must be active in order for the point to be feasible.

And now some proofs. For those coming from outside of the math department (that may not have as much experience proving things as the math students), I will try to make comments about how one would come up with these sorts of proofs on their own as I go.

Theorem 9. *If* $\mathbf{x}$ *is a BFS, then* $\mathbf{x}$ *is a vertex.*

Proof. Assume that P is defined by $\mathbf{Ax} \geq \mathbf{b}$ and that $\mathbf{x}$ is a BFS[9] This means that there exist n linearly independent rows of A which are active constraints. Without loss of generality (changing the order of the constraints does not change the P), we can assume that it is the first n rows.[10] Hence we have

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_0 \\ \mathbf{A}_1\end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix}\mathbf{b}_0 \\ \mathbf{b}_1\end{bmatrix}$$

where

1. $\mathbf{A}_0$ is $n \times n$ and full rank

2. $\mathbf{b}_0$ has length n

3. $\mathbf{x}$ satisfies
$$\mathbf{A}_0\mathbf{x} = \mathbf{b}_0{}^{11} \quad \text{and} \quad \mathbf{A}_1\mathbf{x} \geq \mathbf{b}_1{}^{12}$$

Now notice[13] that for any $\mathbf{y} \in P$, we have

$$\mathbf{A}_0\mathbf{y} \geq \mathbf{b}_0 = \mathbf{A}_0\mathbf{x}$$

which means $\mathbf{A}_0(\mathbf{y}-\mathbf{x}) \geq \mathbf{0}$ (it is a bunch of nonnegative numbers). On the other hand, $\mathbf{A}_0$ is invertible[14] so

$$\mathbf{A}_0(\mathbf{y}-\mathbf{x}) = \mathbf{0} \quad \Rightarrow \quad (\mathbf{y}-\mathbf{x}) = \mathbf{0} \quad \Rightarrow \quad \mathbf{y} = \mathbf{x}$$

That is, if $\mathbf{y}$ is in P then $\mathbf{A}_0(\mathbf{y}-\mathbf{x})$ is a nonnegative vector, and the only time it is zero is if $\mathbf{x} = \mathbf{y}$[15]. Now define

$$\mathbf{c}^\intercal = \mathbf{1}^\intercal\mathbf{A}_0 \quad \text{where } \mathbf{1} \text{ is the vector in } \mathbb{R}^n \text{ with a 1 as each entry}$$

then

$$\mathbf{c} \cdot (\mathbf{y}-\mathbf{x}) = \mathbf{1}\mathbf{A}_0(\mathbf{y}-\mathbf{x}).$$

[9] If your hypothesis includes something that has a definition, it is a good guess that the first thing you do is apply that definition.

[10] This is just to make notation easier — in theory these constraints could be anywhere, but since changing the order of the constraints of a polyhedron doesn't change the polyhedron, I can simply rearrange them so (regardless of where they were) they are now on top, which means I can use block matrix notation.

[11] These are the basic constraints we know to be active.

[12] These are all of the rest of the constraints (they may or may not be active, but they must be satisfied since $\mathbf{x}$ is feasible)

[13] The key observation is that $\mathbf{A}_0$ is still a matrix of constraints.

[14] The main characteristic of a BFS is the linear independence, so we should expect that to show up somewhere!

[15] This is starting to sound like the definition of vertex, but we are still going to need to produce a vector $\mathbf{c}$ somehow (and this is where a bit of cleverness is needed)

Since for a vector $\mathbf{v}$, $\vec{1}^\intercal \mathbf{v}$ is simply the sum of the elements of $\mathbf{v}$, we then have that $\mathbf{c} \cdot (\mathbf{y} - \mathbf{x})$ is simply the sum of the entries of $\mathbf{A}_0(\mathbf{y} - \mathbf{x})$. But those entries are nonnegative (and are only $\vec{0}$ when $\mathbf{x} = \mathbf{y}$), so that means

$$\mathbf{c} \cdot (\mathbf{y} - \mathbf{x}) \geq 0 \quad \text{with equality only when } \mathbf{x} = \mathbf{y}$$

which is equivalent to the definition of $\mathbf{x}$ being a vertex (so we are done).

□

Before giving the next proof, I want to point out that, were you told to prove this on your own, your first thought should be "I should try to prove the contrapositive". The reason is that the definition of extremal point is more like an anti-definition. So knowing that a given $\mathbf{x}$ is an extremal point is not super useful when proving things, because it doesn't give you something you can put your hands on (at least not until we prove these equivalences). Knowing that a given $\mathbf{x}$ is *not* an extremal point, however, is more useful because it means that $\mathbf{y}$ and $\mathbf{z}$ and λ all exist, and these are things you can then try to manipulate.

Theorem 10. *If* $\mathbf{x}$ *is an extremal point, then* $\mathbf{x}$ *is a BFS.*

Proof. We prove the contrapositive — that is, if $\mathbf{x}$ is not a BFS, then it is not an extremal point. As before, we can assume (without loss of generality) that the active constraints are in the top rows of A, only this time, we will take all of them. That is,

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \end{bmatrix}$$

where this time $\mathbf{A}_0$ is *not* full rank (since $\mathbf{x}$ is not a BFS) and $\mathbf{A}_1\mathbf{x} > \mathbf{b}_1$.[16] Since $\mathbf{A}_0$ is not full rank, there exists a vector $\mathbf{d} \neq \vec{0}$ in its kernel.[17] Now form the vectors[18]

$$\mathbf{y} = \mathbf{x} + \epsilon\mathbf{d} \quad \text{and} \quad \mathbf{z} = \mathbf{x} - \epsilon\mathbf{d}$$

It is easy to check that $\mathbf{x} = \frac{1}{2}(\mathbf{y} + \mathbf{z})$ so if we can show that $\mathbf{y}$ and $\mathbf{z}$ are both in P (for some small ϵ), this will show that $\mathbf{x}$ is not an extremal point. [19] So we compute[20]

$$\mathbf{A}\mathbf{y} = \mathbf{A}\mathbf{x} + \epsilon\mathbf{A}\mathbf{d} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{A}_1(\mathbf{x} + \epsilon\mathbf{d})] \end{bmatrix}$$

and similarly for $\mathbf{z}$. So to prove $\mathbf{y}, \mathbf{z} \in P$, we need to show that there exists an ϵ for which

$$\mathbf{A}_1(\mathbf{x} \pm \epsilon\mathbf{d}) \geq \mathbf{b}_1$$

or (equivalently)

$$\mathbf{A}_1\mathbf{x} - \mathbf{b}_1 \geq \pm\epsilon\mathbf{A}_1\mathbf{d} \tag{8}$$

[16] Again, these are the remaining constraints, but this time $\mathbf{A}_0$ contains all active constraints so we know none of these are active.

[17] Note that "not full rank" is another "lack of something" property. Fortunately, this time there is a theorem from linear algebra which gives us something we can put our hands on.

[18] To prove that something is not an extremal point, we need to two vectors and $\mathbf{d}$ is the only thing we have available

[19] Note that the best (some might say "only") way to show $\mathbf{y}$ is in P, is to show $A\mathbf{y} \geq \mathbf{b}$ (similarly $\mathbf{z}$).

[20] Keeping in mind all the properties $\mathbf{A}_0$ has: $\mathbf{A}_0\mathbf{x} = \mathbf{b}_0$ and $\mathbf{A}_0\mathbf{d} = \mathbf{0}$ means we can calculate $\mathbf{A}_0\mathbf{y}$.

However, we have seen that $\mathbf{A}_1\mathbf{x} - \mathbf{b}_1 > 0$[21] so if δ is the smallest element of $\mathbf{A}_1\mathbf{x} - \mathbf{b}_1$ and K is the largest (in absolute value) element of $\mathbf{A}_1\mathbf{d}$ then setting $\epsilon = \delta/K$. Gives

$$\pm\epsilon\mathbf{A}_1\mathbf{d} \leq \delta\vec{1} \leq \mathbf{A}_1\mathbf{x} - \mathbf{b}_1$$

which shows (8) and finishes the proof. □

One final comment — when it comes to proving the final 1/3 of this, it may be tempting to look at these proofs and try something similar. Don't do that! The reason these proofs are the way they are is because the theorems we wanted to prove involved BFS's and talking about BFS's requires matrices. The statements "$\mathbf{x}$ is a vertex" and "$\mathbf{x}$ is an extreme point" are both geometric statements, so you should try to find a relationship between those two that doesn't require matrices (and will end up being simpler than either of the proofs here).

3.1.2 References

1. Bertsimas and Tsitsiklis: Section 2.2

3.2 Optimal solutions to LPs can always be found at extreme points

Now that we have a characterization of corners, the goal will be to show that it suffices to look for optimal solutions in these corners:

Theorem 11. *Consider the problem of minimizing $\mathbf{c} \cdot \mathbf{x}$ for $\mathbf{x} \in P$ where P is a polyhedron (intersection of halfspaces). If*

1. *there exists an optimal solution in P, and*
2. *P contains at least one extreme point*

Then there exists an extreme point in P which gives the optimal value.

The proof from the lecture[22] uses an idea called *ray-tracing* that consists of two observations:

1. if $\mathbf{x}$ is not a BFS, it means there is a direction which is orthogonal to the normal vectors of the active constraints. If we move in that direction, the constraints that were active at $\mathbf{x}$ will remain active.
2. If $\mathbf{x} \in P$ and we move along a line $\mathbf{x} + \lambda\vec{d}$, one of two things must happen: either we leave the polyhedron at some point or we don't. The situation where we don't leave P we referred to as *P containing a line* situation where we do leave P at some point $\vec{y}$, then it means we had to cross a new constraint at $\vec{y}$ and that constraint must be linearly independent from the current ones (assuming we picked the direction $\vec{d}$ like we said in the first part).

This led to the theorem:

Theorem 12. *The following are equivalent:*

[21] If $\mathbf{v} > 0$, then it has a smallest element. The idea will be to pick ϵ small enough so that

$$\pm\epsilon\mathbf{A}_1\mathbf{d} \leq \delta\mathbf{1}$$

is even smaller than that smallest element.

[22] See video of Class 3, part 2.

1. *P contains at least one extreme point*

2. *P does not contain a line*

3. *There exist n constraints defining P which are linearly independent*

The equivalence of the first and last was Problem 4 in Problem Set 2, and the fact that they are equivalent to the second one uses a similar idea. The nice thing about "containing a line" is that it was geometrically intuitive. For example, let Q be a sub-polyhedron of P (that is, Q contains the same inequalities as P but also additional ones). Saying that P has an extreme point implies Q has an extreme point is not visually obvious, because everything depends on where the new constraints are. However, saying that P does not contain a line implies Q does not contain a line is much more obvious because it is a statement about subsets of subsets.

And this is the main idea of the proof of Theorem 11. Assume $\mathbf{c} \cdot \mathbf{x}$ achieves a minimum on P, and let that minimum value be t. Since P has an extreme point, it does not contain a line. Hence the sub-polyhedron

$$Q = \{\mathbf{x} \in P : \mathbf{c} \cdot \mathbf{x} = t\}$$

also does not contain a line (and therefore has an extreme point $\mathbf{x}_*$). The remainder of the proof is showing that $\mathbf{x}_*$ is also an extreme point of P.

The proof that $\mathbf{x}_*$ is an extreme point of P will go by contradiction [23]. Assume that $\mathbf{x}_*$ is not an extreme point in P — that is, we assume that there exist $\mathbf{y}, \mathbf{z}$ in P for which

$$\mathbf{x}_* = \lambda \mathbf{y} + (1 - \lambda)\mathbf{z} \tag{9}$$

for some $0 \leq \lambda \leq 1$. The goal will be to show that $\mathbf{y}$ and $\mathbf{z}$ must also be in Q — if we can do that, it would imply that $\mathbf{x}_*$ was *not actually* an extreme point in Q, and that would be a contradiction (because we picked $\mathbf{x}_*$ to be an extreme point in Q in the first place).

If you haven't seen this before, it may look weird at first, but it is something we will use a lot in this class, so make sure to take note. The idea is to show that the only way for certain inequalities to be true is if they are actually equalities. For example, consider the following set of inequalities:

$$a \leq c \quad \text{and} \quad b \leq d \quad \text{and} \quad a + b = c + d$$

The only way for all three to be true[24] is if $a = c$ and $b = d$.

For our purposes, we will use the fact that t is the minimum possible cost for any points in P:

$$\mathbf{c} \cdot \mathbf{y} \geq t \quad \text{and} \quad \mathbf{c} \cdot \mathbf{z} \geq t \tag{10}$$

and, on the other hand, we have by (9)

$$t = \mathbf{c} \cdot \mathbf{x}_* = \lambda(\mathbf{c} \cdot \mathbf{y}) + (1 - \lambda)(\mathbf{c} \cdot \mathbf{z}) \tag{11}$$

Together, (10) and (11) can only be true if

$$\mathbf{c} \cdot \mathbf{y} = t \quad \text{and} \quad \mathbf{c} \cdot \mathbf{z} = t.$$

But that would mean (by definition) that $\mathbf{y}, \mathbf{z} \in Q$, which is a contradiction.

As we noted, this is a success because it turns what seems like could be an infinite problem into a finite problem — all we have to do is check $\mathbf{c} \cdot \mathbf{x}$ at all extremal points and see which one is best. Well, not really — there are still some issues like what happens if there is no finite optimal

[23] See the proof of Theorem 10 if you want to know why this could be a good approach to try.

[24] This could be proved by another "proof by contradiction" argument, if you want to be rigorous.

solution or if there are no extreme points in P. And even if those things were not an issue, it was still not a *great* idea to use this "check all extremal points" algorithm because the only method we have for finding extremal points is to check combinations of n linearly independent constraints, and there are potentially A LOT of these combinations. Furthermore, many of them are infeasible, so it seems like a waste of effort to find them all. Particularly when we just discussed a method for finding basic feasible solutions that never left the polyhedron at all (ray-tracing!).

However, there is still a lot that we can add to this ray-tracing idea to make it better — for example, if we are trying to minimize $\mathbf{c} \cdot \mathbf{x}$ then we could limit our attention to those BFSs that have a smaller solution that the one we are currently at, which could cut out a lot of possible BFSs that would clearly be worse. So this will be the plan for next class — to formalize this idea of "ray-tracing in a good direction" into an algorithm.

3.2.1 References

1. Bertsimas and Tsitsiklis: Section 2.5, 2.6

4 Week 4: Simplex!

We start by separating the "Solve a linear program" problem into 2 steps:

1. Find a point in the feasible region
2. Find an optimal solution

We will ignore the first step (at least for now). The reason is that we already have an idea for how to find an optimal solution if we knew a point in the feasible region: Firstly, we could ray-trace to a BFS. Then we could ray-trace from BFS to BFS in directions that improved the objective function. The reason we might hope that we can move from BFS to BFS efficiently is that these correspond to edges in the polyhedron. So (at least geometrically) it is easy to imagine how this might go. Of course there are all kinds of "edge cases" (no pun intended) that we will need to deal with, but this will be the main idea of our algorithm, which is known as the *simplex algorithm*.

4.1 The basic idea

Since we understand the geometry of the inequality standard form, the obvious thing would be to try to accomplish this "moving along edges" procedure there. Assume we have a polyhedron $P \subseteq \mathbb{R}^n$ defined by the constraints

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}$$

where $\mathbf{A}$ is a tall skinny matrix (say $m \times n$ where $m > n$) and assume we are able to find a BFS $\mathbf{y}$. Then we could find the constraints (rows of $\mathbf{A}$) that are active at $\mathbf{y}$ — we call these rows a *basis* because they form a basis for $\mathbb{R}^n$[25] An edge is defined by $n-1$ active constraints, so to move along an edge, we will need to make one of our active constraints inactive (and thus remove that constraint from the basis). We then travel along that edge until we run into a new constraint, which when added to the basis gives us a new BFS. And then we iterate (not forever, of course — we'll discuss the stopping criteria in a moment).

So now let us see how this translates into equality standard form (which will have its own advantages, as we will see). A general equality standard form problem has a polyhedron defined by the constraints

$$\begin{aligned} \mathbf{U}\mathbf{y} &= \mathbf{v} \\ \mathbf{y} &\geq \mathbf{0} \end{aligned}$$

where $\mathbf{U}$ is a short and fat matrix (say $M \times N$ with $N > M$). Note that I am using different values from above $\mathbf{U}$ vs. $\mathbf{A}$ (for example) to make it clear that if we had turned our inequality standard form into an equality standard form, the matrices would be different. Specifically,

$$\mathbf{U} = \begin{bmatrix} \mathbf{A} & -\mathbf{I}_{m\times m} \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \mathbf{w} \end{bmatrix}. \tag{12}$$

HOWEVER, we are going to consider the case that we came upon an LP in equality standard form that we don't know where it came from; in particular, we will assume that $\mathbf{U}$ is just some arbitrary $M \times N$ matrix with $N > M$ (not necessarily in the form of (12)). Since $\mathbf{U}$ has N columns, the associated linear program will have N variables , and so a basic solution would correspond to having N linearly independent active constraints. But unlike before, some of

[25] It is possible that there are more than n active constraints, but we will ignore that possibility for the moment and come back to it later.

these constraints are going to always stay active (since we have equalities). In fact M of the constraints will remain active, which means we will need to find $N-M$ more active constraints in order to construct a basic solution. By process of elimination, those $N-M$ constraints must be nonnegativity constraints (which is what we call the $\mathbf{y} \geq \mathbf{0}$ constraints). Note that saying a nonnegativity constraint $y_i \geq 0$ is "active" is the same as saying $y_i = 0$, so this would amount to locking down $N-M$ of the variables at 0.

Of course we still need all of the active constraints to be linearly independent! Fortunately, some of the linear independence is obvious:

1. the nonnegativity constraints are linearly independent from each other because they come from a big identity matrix

2. the equality constraints are linearly independent from each other because the slack variables form an identity matrix

So we just need to check that the equality constraints are linearly independent from the nonnegativity constraints. For this, we can do Gaussian elimination — for each nonnegativity constraint, we can "zero-out" the column that has a single 1 in it — and what we are left with is the following lemma:

Lemma 13. *Let $S \subset \{1, \ldots, N\}$ with $|S| = N - M$. The set of constraints*

$$\{y_i = 0 : i \in S\}$$

is linearly independent from the equality constraints if and only if the $M \times M$ submatrix of $\mathbf{U}$ formed by removing the columns in S is invertible.

So for reasons I still cannot explain (and has led to many years of confusion for me) we defined a *basic column* to be a column that is indexed by a variable which is *not* an active constraint.[26] So, to be clear:

- When we are in inequality standard form, the "basis" is a "row basis" which consists of the rows that contain active constraints.

- When we are in equality standard form, the "basis" is a "column basis" which consists of columns associated to the variables that are *inactive* (that is, allowed to be nonzero).

I could spend the rest of this section apologizing for the absurdity of these two statements, but instead I will note that if we just think about the dimensions of things, this actually does make sense. The whole point of requiring linear independence is so that we can get an invertible matrix. Invertible matrices are square. So if $\mathbf{A}$ is an $m \times n$ matrix with $m > n$, an invertible submatrix is going to be formed from n linearly independent rows. On the other hand, $\mathbf{U}$ is an $M \times N$ matrix with $N > M$, so an invertible submatrix in this case is going to be formed from M linearly independent columns. To find those columns, we use the fact that a total of N linearly independent active constraints are needed, with M of them being equality constraints. This leaves $N - M$ nonnegativity constraints that must be active (that is $N - M$ variables that are locked down at 0. Since there are N columns total in $\mathbf{U}$, it makes sense that those M columns are the ones that are *not* associated to the $N - M$ rows.

In any case, we still have some translating to do and from what we just saw, we should expect much of it to be "flipped" in some way. In particular, the way to travel along an edge

[26] Note that the book refers to the columns by their variables, so column u_k corresponds to variable x_k. Furthermore, it often will take the analogy further and say the basis contains *the variables*, but that really means the columns corresponding to those variables.

in the row basis was to remove a constraint from the basis. The way to travel along an edge in the column basis will be to *add* a column to the basis. And while this may not be as intuitive as the row basis case, the fact that we know that what we are doing (geometrically) is moving along an edge turns out to be all we need in order to do our calculations (at least if we are clever about it). Geometrically, moving along an edge means we need to move from $\mathbf{y} \to \mathbf{y} + \theta\mathbf{d}$ for some vector $\mathbf{d}$ (which we will now have to figure out). Furthermore, $\mathbf{d}$ should (in some way) correspond to a column u_k entering the basis.

So let's simply list the things we know:

- At the start, u_k is not in the column basis, so $x_k = 0$. In order to get u_k into the basis, we need to make it[27] nonzero, so we need d_k to be nonzero. Since we are free to scale our direction vector however we want, we can just assume that $d_k = 1$.
- All other columns u_i that are not in the basis should stay out of the basis, so for these we want to keep $x_i = 0$ — the only way to do this to have $d_i = 0$ for each nonbasic column u_i where $i \neq k$.
- Lastly, we know that as long as we are traveling along an edge, we are going to remain feasible, which means that
$$\mathbf{U}(\mathbf{y} + \theta\mathbf{d}) = \mathbf{b}$$
for all θ. In particular, this tells us that we need to have $\mathbf{Ud} = \mathbf{0}$.

Now I claim $\mathbf{d}$ is completely defined. In the first two steps, we assigned values to $n - m$ of the entries of $\mathbf{d}$ (either 0 or 1). In the third step, we saw that $\mathbf{d}$ must satisfy $\mathbf{Ud} = \mathbf{0}$, which is a collection of m constraints. Filling in the values from the first two steps, this leaves m (linearly independent, since these are basis columns) constraints in m variables, which means there is a unique solution. So, in some sense, the equation $\mathbf{Ud} = \mathbf{0}$ has dictated what the other entries of $\mathbf{d}$ have to be.

4.2 An example

For example, take the 3×5 matrix $\mathbf{U}$ with columns

$$\mathbf{U} = \begin{bmatrix} | & | & | & | & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 & \mathbf{u}_4 & \mathbf{u}_5 \\ | & | & | & | & | \end{bmatrix}$$

and let's imagine that our current basis is $\beta = \{\mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_5\}$. The submatrix corresponding to these columns (called the *basis matrix*) is therefore

$$\mathbf{B} = \mathrm{col}_\beta(\mathbf{U}) = \begin{bmatrix} | & | & | \\ \mathbf{u}_2 & \mathbf{u}_3 & \mathbf{u}_5 \\ | & | & | \end{bmatrix}$$

and this is invertible (since a basis contains linearly independent vectors). The BFS $\mathbf{y}$ that corresponds to β would have all of the nonbasis entries 0 and the other entries determined by the basis matrix:

$$\mathbf{y} = \begin{bmatrix} 0 \\ y_2 \\ y_3 \\ 0 \\ y_5 \end{bmatrix} \quad \text{where} \quad \begin{bmatrix} y_2 \\ y_3 \\ y_5 \end{bmatrix} = \mathbf{B}^{-1}\mathbf{v}.$$

[27] At least, we need to allow it the opportunity — it may stay at 0 for reasons related to degeneracy.

The direction vector $\mathbf{d}$ that corresponds to adding $\mathbf{u}_1$ to the column basis would be

$$\mathbf{d} = \begin{bmatrix} 1 \\ d_2 \\ d_3 \\ 0 \\ d_5 \end{bmatrix} \quad \text{where} \quad \mathbf{d}_\beta = \begin{bmatrix} d_2 \\ d_3 \\ d_5 \end{bmatrix} = -\mathbf{B}^{-1}\mathbf{u}_1$$

If instead we wanted to add $\mathbf{u}_4$ into the basis, we would form the vector

$$\mathbf{d} = \begin{bmatrix} 0 \\ d_2 \\ d_3 \\ 1 \\ d_5 \end{bmatrix} \quad \text{where (this time)} \quad \mathbf{d}_\beta = \begin{bmatrix} d_2 \\ d_3 \\ d_5 \end{bmatrix} = -\mathbf{B}^{-1}\mathbf{u}_4$$

4.2.1 Picking the column to enter the basis

Now that we know which directions we can travel in, the obvious question is which direction to pick*. Note that my use of the term pick* is not a typo — the reason for this is that "picking things" is not a trivial task[28]. And while different picking routines can cause different global behaviors, it doesn't make sense to worry about global behavior now (when we are still figuring out the nuts and bolts of how simplex works). And at the same time, this whole "moving down an edge" thing that simplex is going to do will be (theoretically) the same regardless of which direction we do pick. So (at least for now) pick* will be a black box picking algorithm that we can worry about later.

However, there is one obvious thing that we should demand from pick* — it should pick something that improves the objective function! So given our basis β, we take each $u_k \notin \beta$ and form a direction vector $\mathbf{d}$ (which we saw above is)

- $d_k = 1$ (this forces x_k into the (new) basis)
- $d_i = 0$ for all other non basic variables (this keeps these other x_i out of the (new) basis)
- $\mathbf{d}_\beta = -\mathbf{B}^{-1}\,\mathrm{col}_k(\mathbf{U})$ (this keeps us feasible)

where $\mathbf{B} = \mathrm{col}_\beta(\mathbf{U})$. So now we should figure out if moving in that direction is going to help. To do this, we will calculate what is called the *reduced cost* for the basis β when adding the column u_k (which is nothing more than the derivative of the cost function in the direction $\mathbf{d}$):

$$\bar{c}_k = \mathbf{c} \cdot \mathbf{d} = \mathbf{c}_\beta \cdot \mathbf{d}_\beta + c_k = -\mathbf{c}_\beta^\intercal \mathbf{B}^{-1}\,\mathrm{col}_k(\mathbf{U}) + c_k.$$

So if our goal was to maximize $\mathbf{c} \cdot \mathbf{x}$, we would want to pick a column u_k (variable y_k) which has $\bar{c}_k > 0$ and if our goal was to minimize $\mathbf{c} \cdot \mathbf{x}$, we would want to pick a column/variable which has $\bar{c}_k < 0$. So for now we will assume that pick* gives us such a column/variable (assuming it exists).

4.2.2 Picking the column to leave the basis

For the moment, let's assume such a u_k exists (one that improves the objective function) and let $\mathbf{d}$ be the vector pointing in that direction. We still need to figure out how far to move

[28] This is why we need things like the Axiom of Choice.

(the value of θ in $\mathbf{y} + \theta\mathbf{d}$). One thing that is nice about $\mathbf{d}$ is (by the way we constructed it), $\mathbf{U}(\mathbf{y} + \theta\mathbf{d}) = \mathbf{b}$ for all θ, so the only constraints we need to worry about are the nonnegativity constraints $y_i \geq 0$. We claim there are two possibilities:

1. all elements of $\mathbf{d}$ are nonnegative — then adding $\theta\mathbf{d}$ is just going to make each y_i bigger, so we will never get a new $y_i = 0$. Then we can let $\theta \to \infty$ (knowing we are still feasible), which will drive the objective function as high (or low) as we want. In other words, this linear program has no optimal solution.

2. there exists at least one j with $d_j < 0$ — then $y_j + \theta d_j$ will eventually reach 0, and we can easily solve for when that happens:

$$\theta_j = \frac{-y_j}{d_j}.$$

   For θ smaller than this, the new value of y_j will be feasible and for θ larger than this, the new value will be infeasible. So in order to keep everything feasible, we need to pick*[29] the smallest such θ_j:

$$\theta_* = \min_{i:d_i<0} \frac{-y_i}{d_i}$$

   and this will ensure that all variables remain feasible in $\mathbf{x} + \theta_*\mathbf{d}$. Furthermore, whichever index resulted in the smallest value of θ (let's say it was y_t), will now have $y_t = 0$, which means we can take it out of the basis.

So, what have we accomplished? Well, it is possible that we found a direction which proved the linear program is unbounded, which would definitely be a success (it means we are done). In the other case, we managed to take a variable out of the basis and replace it with a new one. But, more importantly, we improved the objective function by $\theta_*\overline{c}_k$, which would be a smaller success (but a success nonetheless). I say "would be" because if (for some reason) $\theta_*\overline{c}_k = 0$, then we didn't actually improve the objective function at all. So we still have some issues to deal with — in particular:

1. What do we do if none of the $\overline{c}_k$ improve the objective?

2. What happens if an $\overline{c}_k$ does improve the objective function, but the resulting $\theta_* = 0$?

The first situation is easy to deal with because of the following theorem, which we will prove on Problem Set 4:

Theorem 14. *Let* $\mathbf{x}$ *be a basic feasible solution of a linear program* P *for which none of the reduced costs improve the objective function. That* $\mathbf{x}$ *is an optimal solution for* P*.*

Note that "not improve" includes the possibility that some of the $\overline{c}_j = 0$ (for example). The second situation, however, is a bit more touchy.

4.3 Degeneracies (and cycling)

Notice that in order for θ_* to be 0, it means that the variable we chose to take out (say y_t) had the value 0 in our current BFS $\mathbf{y}$. But remember that in equality standard form, having $y_t = 0$ means that this is an active constraint — in particular, we have more active constraints than we need at this BFS. A BFS $\mathbf{y} \in \mathbb{R}^n$ is called *degenerate* if there are more than n constraints

[29] If there happens to be a tie, then we need to find some way to decide which one to put in, but again we can deal with that later.

that are active at $\mathbf{y}$. In theory, this is not a problem — we could simply run simplex exactly how we would before, move a total distance of $\theta_* = 0$ in some direction (which is another way of saying we just stay at $\mathbf{y}$) and then swap out y_k for y_t in the basis. Even though we didn't move, we didn't "do nothing" because we now have a new basis and this will give us new edges we can try to use to improve our objective function.

In practice, however, this can be an issue. The reason is that, although we haven't discussed it yet, having an algorithm that improves the objective function at every step is really nice, because it guarantees you never return to the same point twice. [30] If we are allowed to do things that don't improve the objective function, it is possible that we could start going in circles.

For example, imagine you had a basis $\beta = \{y_2, y_3, y_5\}$ and then you do all of this hard work to determine that the best thing to do is to remove y_3 from the basis and replace it with y_1. Now you have a new basis $\beta' = \{y_1, y_2, y_5\}$ and you go and do all of this hard work again to determine that the best thing to do is to remove y_1 from the basis. Guess what might end up replacing it? Since you haven't moved, y_3 is still active (even though it is no longer in the basis) and so it is quite possible that y_1 gets replaced by y_3 which puts you back in the β basis.

4.3.1 Bland's Rule

This phenomenon is called *cycling* and it is something we very much would like to avoid. Now I know what you are thinking — "Hey, I'm a pretty smart person, so I can just look out for these things and make sure it never repeats." But here is the thing: regardless of what you end up doing in your life, there is likely only one time and place that you will need to *actually perform the physical act of "solving a linear program"* and that is in this class. After that, you will want very much to never solve a linear program by hand again and will instead get a computer to do it for you. And the issue with computers is that they are only as smart as the programs running on them, so it would be better if we could find a programmatic way to make sure that this doesn't happen. One popular method for accomplishing this is known as *Bland's rule* or *smallest index rule*:

1. If there are multiple variables that, when added to the basis, would improve the objective function, always choose to add the one with the smallest index.

2. If there are multiple variables that tie for the minimum value θ_*, always choose to remove the one with the smallest index.

It is known that Bland's rule avoids cycling[31] and is actually quite nice in a number of respects:

- It does not require you to keep track of any of the previous bases (like your "I'm a pretty smart person" method probably needed)

- It simplifies computations in the respect that you don't have to calculate reduced costs for every nonbasic variable — just do them in order until you find one that will improve the objective function, and then use that one.

4.3.2 Kick the polyhedron!

There is a second "clever" way to get rid of cycling, and that is to add a small amount of random noise to $\mathbf{b}$, which I like to think of this as kicking the polyhedron to cause all of these constraints which magically line up to no longer line up). The idea is that as long as noise is

[30] This would be like an M. C. Escher drawing where you go up 4 sets of stairs and end up back where you were (which, fortunately, only happens in M. C. Escher paintings).

[31] We will not prove this in class, but you can assume it is true for the purpose of this class.

small enough, then you shouldn't change which BFS is the optimal solution. We saw something like this before [32] when we were dealing with the situation of having a non-unique optimum solution — in that case, the "fix" was to add random noise to $\mathbf{c}$. The difference there was that changing $\mathbf{c}$ did not change the feasible region at all, so it was easy to translate solutions back and forth. This time, we are changing the polyhedron, so how are we going to calculate the new point?

Well, keep in mind that "points" aren't really the central object in our algorithm — *bases* are. And bases are much more "stable" than points are. To change a basis, you have to do something extreme — take something out and put something else in. To change a point, you just need to add a little noise. So the fact that everything happens in the "basis" language is actually quite useful in this scenario (and we will seen more advantages later). So the idea would be to solve the new linear program, get an *optimal basis*, and then move that basis back to the old linear program. Given a linear program

$$\min \{\mathbf{c} \cdot \mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

a feasible basis β is called an *optimal basis* if the matrix $\mathbf{B} = \text{col}_\beta(\mathbf{A})$ satisfies

- $\mathbf{B}^{-1}\mathbf{b} \geq \mathbf{0}$ (this is needed for feasibility)
- $\mathbf{c}^\intercal - \mathbf{c}_\beta^\intercal \mathbf{B}^{-1}\mathbf{A} \geq \mathbf{0}$ (see Theorem 14).

As long as you don't move things too much, BFS's should map to BFS's so in the end you can simply calculate the point in the old LP from the same optimal basis. This brings us to one final observation — we still can't solve linear programs. The algorithm we came up with here *could* solve linear programs, assuming we had a way to find a BFS to start it on. Fortunately, we will see a method for finding a BFS of a polyhedron on the problem set.

4.3.3 References

1. Bertsimas and Tsitsiklis: Section 2.4, 3.1–3.2

[32] Computer science is full of scenarios that are potentially problematic, but only if some very special conditions happen. Adding small (but random) noise is a useful technique to eliminate the need to deal with such scenarios.

5 Week 5: LP Duality

If you haven't read Section 3.1.1 (with excellent commentary!), you should read it. Specifically, read them for the commentary. The reason I say this is that the commentary points out what is a fundamental imbalance in the world of mathematics — the difference between something existing and something not existing.

5.1 Certificates

On the surface, existence vs. non-existence seems like an even match. Some things exist, some don't. Yes, or no. Yin and Yang (etc). The issue comes when you start trying to *prove* things. Let's go back to the optimization problem from the very first class:

$$\max f(x) \quad \text{such that} \quad x \in X$$

where $f(x)$ is the "height" function and X is the set of people in our class. As we noted, this is an annoying problem because (in general) the only way to solve it is to ask every single person in the class how tall they are (that is, calculate $f(x)$ for every feasible x) and then to find the maximum. But let's say I did all the work and went around and asked everyone and *found the actual maximum*: 224 cm.

So then when the Dean comes in and asks me the solution to my optimization problem, I tell him "224 cm." But how can he check to see that my answer is right? It would be nice to be able to prove to him (somehow) that my answer is correct, without having to force him to do the problem all over again. Such a thing is called a *certificate*. It is a way to show that an answer is correct without going through the effort of solving the entire problem[33] If you have any experience with computational complexity, the class of problems "NP" contains precisely those problems that have certificates (despite those certificates possibly being hard to find).

The truth is, there is no way to provide a certificate showing that 224 cm is the tallest person in the class. At least, not completely. In order to *prove* that 224 cm is the tallest height in the class, I really need to prove two things:

1. There is a person in the class that is 224 cm tall
2. Nobody in the class is taller than 224 cm

The first part *does* have a certificate — to prove that someone in the class is 224 cm tall, I simply point to Andre the Giant (who, in case you didn't know, is an honorary member of the class). Then the Dean can go over and measure Andre and see that he is 224 cm tall (at least according to his biography) and now he can be confident that the first statement is true (he has seen proof). This is where the difference between existence and non-existence becomes clear. If you can find something, proving that thing exists is easy (you just show everyone the thing that you found). On the other hand, how do you prove something *doesn't* exist? Well, one way would be to go and ask every single person in the class their height again (which is as hard as solving the original problem). Are there other ways?

[33] This should suggest that the process of finding a certificate is at least as hard as solving the problem in the first place.

5.1.1 Certificates for Linear Programs

In general, no,[34] but in some (very special) cases, yes, and the case of linear programming is one of those cases[35]. Let's say I have the linear program (in inequality standard form)

$$\begin{aligned} \min\ & x + y \\ \text{s.t. } & 2x + 3y \geq 4 \\ & x + 2y \geq 3 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$

And let's say I have taken this course already and know how to find the answer and the answer I get is 2. Now to show you my answer is right, I need to show you that (if we let s be the true solution)

1. $s \leq 2$, and
2. $s \geq 2$

Showing that $s \leq 2$ is easy — I just show you the point $(1, 1)$. Now you can check that this point satisfies the constraints (it does) and then you can calculate the value of the objective function at that point (it is 2), so the true minimum is definitely ≤ 2.

To show that $s \geq 2$, what I will try to do is to generate new inequalities from the old ones by combining them in different ways. For example, $x + 2y \geq 3$ and $y \geq 0$ when added together gives $x + 3y \geq 3$ (a new inequality!). Of course this is not a particularly helpful inequality for us, since we are interested in $x+y$ and not $x+3y$. So we will try to be a bit more clever and find a way to write the objective covector $\mathbf{c}^\intercal = [1, 1]$ as a combination of the constraint covectors

$$\text{row}_1(\mathbf{A}) = [2, 3] \quad \text{row}_2(\mathbf{A}) = [1, 2] \quad \text{row}_3(\mathbf{A}) = [1, 0] \quad \text{row}_4(\mathbf{A}) = [0, 1].$$

This is equivalent to multiplying $\mathbf{Ax} \geq \mathbf{b}$ on the left by a covector. For example, if I take $\mathbf{u}^\intercal = [1/3, 0, 1/3, 0]$ then

$$\mathbf{Ax} \geq \mathbf{b} \quad \Rightarrow \quad \mathbf{u}^\intercal\mathbf{Ax} \geq \mathbf{u}^\intercal\mathbf{b}$$

which, when I multiply it all out says $x + y \geq 4/3$, which is a nice lower bound, but not the one I was hoping for (to prove what I want, I would need to get $x + y \geq 2$). But I can fix this by instead using the covector $\mathbf{u}^\intercal = [2, -2, -1, -1]$, so that when I simplify $\mathbf{u}^\intercal\mathbf{Ax} \geq \mathbf{u}^\intercal\mathbf{b}$, I get $x + y \geq 2$, which then proves my answer is right.

Yes, yes, I am sure many of you are saying "but that's not how inequalities work" and the reason you might be saying that is because you are correct: that is not how inequalities work. This is one of the reasons that inequalities tend to be harder to work with than equalities — you can add/subtract/multiply/etc equalities without any issues, but inequalities are not so easy. When we multiply the second constraint by -2 we get

$$-2x - 4y \leq -6$$

which leaves me with a $\geq$ and $\leq$ and I can't add those together like I would a normal equation.

HOWEVER, this general idea would work as long as I add *nonnegative* multiples of the constraints together. In particular, anytime I can find a nonnegative linear combination of

[34] Although it is not clear what "in general" means here. This is related to an extremely important (as of yet unsolved) problem in computational complexity concerning whether NP is the same as co-NP.

[35] This is one of the reasons that linear programming is such a useful optimization tool.

constraints that add up to $[1, 1]$, that will give me a lower bound on the solution to the problem (like $\hat{u} = [1, 1, 0, 0]$ that we tried above). So, for example, I could try $\hat{u} = [1/5, 1/5, 2/5, 0]$ which will give me the new inequality

$$x + y \geq \frac{7}{5}$$

which is slightly better than $4/3$ but not good enough to prove what I want to prove: that $x + y \geq 2$. The goal then would be to look for a nonnegative covector $\hat{u}$ such that

$$\mathbf{u}^\intercal \mathbf{A} = [1, 1] \quad \text{and} \quad \mathbf{u}^\intercal \mathbf{b} = 2$$

Such a covector $\hat{u}$ would then provide a certificate that, in fact, $s \geq 2$ and so my answer would be correct. As it turns out, such a covector $\hat{u}$ doesn't exist, and this is not hard to see. When we combine constraints together correctly, points that used to be feasible should not suddenly become infeasible. So when I see that the point $(x, y) = (1, 1/2)$ is feasible in the initial linear program with value $3/2$, I should be very suspicious if I combined constraints in some way and got $x + y \geq 2$ because this new constraint would make $(1, 1/2)$ infeasible.

In fact, $(x, y) = (1, 1/2)$ is the true solution to this problem, and this time we *can* find a certificate that will allow us to prove it: $\mathbf{u}^\intercal = [0, 1/2, 1/2, 0]$ gives

$$\mathbf{u}^\intercal \mathbf{A} = [1, 1] \quad \text{and} \quad \mathbf{u}^\intercal \mathbf{b} = 3/2.$$

Hence I can use the certificate $(x, y) = (1, 1/2)$ to prove $s \leq 3/2$ and the certificate $\mathbf{u}^\intercal = [0, 1/2, 1/2, 0]$ to prove $s \geq 3/2$. Most importantly, however, both certificates can be checked with only basic linear algebra knowledge (no knowledge of how we actually found these certificates is needed).

This second certificate is called a *dual* certificate and while it may (at this point) look like we just came up with this idea for making dual certificates out of nowhere, it is actually nothing new at all[36]. But to actually prove that your solution to the problem is correct, you have to find one that matches the solution you got! Fortunately, I was able to find such a certificate for the problem above, but it is not clear whether a certificate this good always exists (though we will return to this question in a moment).

In conclusion, it is one thing to be able to solve an optimization problem, but it is a completely different thing to be able to prove to someone what the right answer is without having to re-solve the entire problem[37] In general, proving one side of an inequality is easy (you just find a feasible solution) whereas showing the other side of the inequality is difficult (because it is hard to prove the lack of the existence of something). Linear programming, however, comes with a "built in" way to get dual certificates and each such certificate will give a bound (of different quality) on the difficult side. And if we can find a certificate and a dual certificate that have matching values, this makes it easy to prove that a given solution is correct. So there are really three options at this point:

1. This certificate-finding method will always be able to produce an "optimal certificate" that matches the optimal value of the linear program.

2. All linear programs have an "optimal certificate" but this method will not always work to find one

[36] Let's put our detective hats on and see what is happening here: the dual certificate we are looking for is a way to write the objective vector as a nonnegative combination of constraint vectors — is this similar to something we have seen before (cough, cough, Lagrange Multipliers, cough, cough)?

[37] And, frankly, certificates are *better* than solving the whole problem again, because if you made a mistake the first time, chances are good that you will make the same mistake the second time.

3. There are some linear programs which do not have "optimal certificates"

Obviously the first option would be the best (for us), so let's see if it is true.

5.1.2 Good Certificates for Linear Programs!

We have seen that if we have a linear program

$$\begin{aligned} \mathcal{P} = \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \end{aligned}$$

then any $\mathbf{u} \geq \mathbf{0}$ for which $\mathbf{u}^\intercal \mathbf{A} = \mathbf{c}^\intercal$ will give you a certificate (which can be used to prove a lower bound on the optimal solution), and the quality of that certificate depends on the value of $\mathbf{u}^\intercal \mathbf{b}$. So if we are going to spend time looking for a certificate, we might as well try to find a good one. How good could we do? Well if we were to write down a characterization of the "best possible" certificate (of this type), it would be something like

$$\begin{aligned} \mathcal{D} = \max \quad & \mathbf{b} \cdot \boldsymbol{\lambda} \\ \text{s.t.} \quad & \boldsymbol{\lambda}^\intercal \mathbf{A} = \mathbf{c}^\intercal \\ & \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned} \tag{13}$$

which is a LINEAR PROGRAM! We call this new linear program the *dual* of the original one, and I claim that this is an extremely interesting phenomenon (which we will start to examine in the next section).

5.1.3 References

1. Bertsimas and Tsitsiklis: Section 4.1

5.2 Duality

The goal now is to formalize this notion the we saw in the previous section and generalize it to situations where we are not (necessarily) in inequality standard form. But before doing that, take another look at what (13) is trying to do... it is trying to write the objective function as a linear combination of the constraints, which is exactly how Lagrange multipliers work. Fortunately, we developed a geometric intuition of Lagrange multipliers (Section 1.2.2) and so we can expect this to come in handy when trying to understand this strange new "duality" concept.

5.2.1 Dual linear programs

There is one small point of language that we should clear up before moving on. Recall that we had the linear programs

$$\begin{aligned} \mathcal{P} = \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \end{aligned} \qquad\qquad \begin{aligned} \mathcal{D} = \max \quad & \mathbf{b} \cdot \boldsymbol{\lambda} \\ \text{s.t.} \quad & \mathbf{A}^\intercal \boldsymbol{\lambda} = \mathbf{c} \\ & \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned}$$

above and we said that $\mathcal{D}$ is called the *dual* of $\mathcal{P}$. In a moment, we will define a duality operator that can be applied to any linear program to form "the dual" and it is quite common in the literature to see the original linear program being called "the primal." However we will see that

this duality operator is an *involution* (that is, applying it twice gets you back to where you started, like the function $1/x$). So I find it a bit misleading to call $\mathcal{P}$ the "primal" because there was nothing special about $\mathcal{P}$ apart from the fact that it was the linear program I started with. Had I started with $\mathcal{D}$, then I could very easily be calling it the "primal" and $\mathcal{P}$ the "dual" and one can see how this would get confusing. So instead I will refer to $\mathcal{P}$ and $\mathcal{D}$ as a *primal/dual pair*.

The most important property of $\mathcal{P}$ and $\mathcal{D}$ that we will want to preserve in general primal/dual pairs is that for any $\mathbf{x} \in \text{feas}(\mathcal{P})$ and $\boldsymbol{\lambda} \in \text{feas}(\mathcal{D})$, we want to have

$$\mathbf{c} \cdot \mathbf{x} \geq \mathbf{b} \cdot \boldsymbol{\lambda}. \tag{14}$$

That way any feasible solution for $\mathcal{P}$ serves as a certificate for $\mathcal{D}$ and any feasible solution to $\mathcal{D}$ serves as a certificate for $\mathcal{P}$. So, in particular, if you could find a $\mathbf{x}^* \in \text{feas}(\mathcal{P})$ and $\boldsymbol{\lambda}^* \in \text{feas}(\mathcal{D})$ for which

$$\mathbf{c} \cdot \mathbf{x}^* = \mathbf{b} \cdot \boldsymbol{\lambda}^*$$

then $\mathbf{x}^*$ and $\boldsymbol{\lambda}^*$ would have to be optimal solutions to $\mathcal{P}$ and $\mathcal{D}$ (respectively).

To find the "dual" of an arbitrary linear program, we first turn it into a minimization problem and then use the map:

$$\begin{array}{rcl}
\min \ \mathbf{c} \cdot \mathbf{x} & \Longrightarrow & \max \ \boldsymbol{\lambda} \cdot \mathbf{b} \\
\text{row}_i(\mathbf{A}) \cdot \mathbf{x} \geq b_i & \Longrightarrow & \lambda_i \geq 0 \\
\text{row}_i(\mathbf{A}) \cdot \mathbf{x} \leq b_i & \Longrightarrow & \lambda_i \leq 0 \\
\text{row}_i(\mathbf{A}) \cdot \mathbf{x} = b_i & \Longrightarrow & \lambda_i \text{ free} \\
x_j \geq 0 & \Longrightarrow & \text{col}_j(\mathbf{A}) \cdot \boldsymbol{\lambda} \leq c_j \\
x_j \leq 0 & \Longrightarrow & \text{col}_j(\mathbf{A}) \cdot \boldsymbol{\lambda} \geq c_j \\
x_j \text{ free} & \Longrightarrow & \text{col}_j(\mathbf{A}) \cdot \boldsymbol{\lambda} = c_j
\end{array}$$

Figure 1: The duality map for general LP constraints.

The first property that one can check is the one mentioned above: that this map is an *involution*. That is, if we started with a minimization problem $\mathcal{P}$ and

- Found $\mathcal{D}$ (a maximization problem) using the duality map above
- Turned $\mathcal{D}$ into a minimization problem $\mathcal{D}'$ by multiplying the cost by -1
- Did the duality map above to $\mathcal{D}'$ to get a new maximization problem $\mathcal{Q}$
- Turned $\mathcal{Q}$ into a minimization problem $\mathcal{Q}'$ by multiplying the cost by -1

then we would find that $\mathcal{Q}' = \mathcal{P}$. Not that $\mathcal{Q}'$ and $\mathcal{P}$ are equivalent — they would be *exactly the same*. So this tells us that the arrows in the duality map are actually double arrows, and so we can move back and forth between a minimization problem and its dual maximization problem by going in either direction. So (as we suggested) neither one is really "the primal" or "the dual" — you get to say which is the primal and then the other one becomes the dual.

However if you look closely, going from left to right is DIFFERENT than going from right to left. So how do you know which one to do? That depends entirely on whether you are moving

from min to max (in which case you use $\Rightarrow$) or from max to min (in which case you use $\Leftarrow$). So for example,

$$\begin{array}{rl} \min & \mathbf{c}\cdot\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \qquad\qquad \begin{array}{rl} \max & \mathbf{b}\cdot\boldsymbol{\lambda} \\ \text{s.t.} & \mathbf{A}^\intercal\boldsymbol{\lambda} \leq \mathbf{c} \end{array}$$

and

$$\begin{array}{rrcrcrl} \min & x_1 & + & 2x_2 & + & 3x_3 & \\ \text{s.t.} & -x_1 & + & 3x_2 & & & = 5 \\ & 2x_1 & - & x_2 & + & 3x_3 & \geq 6 \\ & & & & & 2x_3 & \leq 7 \\ & x_1 & & & & & \geq 0 \\ & & & x_2 & & & \leq 0 \\ & & & & & x_3 & \text{free} \end{array} \qquad \begin{array}{rrcrcrl} \max & 5\lambda_1 & + & 6\lambda_2 & + & 7\lambda_3 & \\ \text{s.t.} & -\lambda_1 & + & 2\lambda_2 & & & \leq 1 \\ & 3\lambda_1 & - & \lambda_2 & & & \geq 2 \\ & & & 3\lambda_2 & + & 2\lambda_3 & = 3 \\ & \lambda_1 & & & & & \text{free} \\ & & & \lambda_2 & & & \geq 0 \\ & & & & & \lambda_3 & \leq 0 \end{array}$$

are both primal/dual pairs. The key property that the duality map enforces is that

$$\lambda_i(\text{row}_i(\mathbf{A})\cdot\mathbf{x} - b_i) \geq 0 \quad \text{and} \quad x_j(c_j - \text{col}_j(\mathbf{A})\cdot\boldsymbol{\lambda}) \geq 0$$

for all i, j because that is what ensures that we get the same inequality as in (14):

Theorem 15 (Weak Duality)**.** *Let $\mathcal{P}$ be a minimization problem and $\mathcal{D}$ a maximization problem related via the map in Figure 1. Then for all $\mathbf{x} \in \text{feas}(\mathcal{P})$ and all $\boldsymbol{\lambda} \in \text{feas}(\mathcal{D})$ we have*

$$\mathbf{b}\cdot\boldsymbol{\lambda} \leq \mathbf{c}\cdot\mathbf{x}$$

Or, in other words, a feasible solution to the dual acts as a certificate for the primal (and vice versa). So the big question now is: can I always find a certificate that matches the optimal value of the primal?

5.2.2 Strong Duality

To investigate this, we need to look at what the dual problem is doing — trying to find a nonnegative linear combination of the constraints that added up to the objective function. That sounds like... Lagrange Multipliers (see Section 1.2.2)! Lagrange multipliers tells us that at the optimal solution, the objective function needs to be a nonnegative linear combination of the active constraints. In the language of simplex, that becomes Theorem 14 which in turn gives the following theorem

Theorem 16 (Strong Duality)**.** *If a linear program has a finite optimal solution, then it's dual has a finite optimal solution and the two optimal values are equal.*

This is where we see that the idea of a "basis" actually works quite well with this whole duality thing. In particular, if I am doing simplex to an equality standard form problem and I have a basis β, then that basis gives me a solution in the original problem:

$$\mathbf{x}_\beta = \mathbf{B}^{-1}\mathbf{b} \qquad \text{and 0 elsewhere}$$

but it also gives me a solution in the dual:

$$\boldsymbol{\lambda}^\intercal = \mathbf{c}_\beta^\intercal\mathbf{B}^{-1}$$

and since the matrix goes from $\mathbf{A}$ to $\mathbf{A}^\intercal$ in the dual map, the column basis gets mapped to a row basis. As a result, an equivalent way of defining optimal basis would be "any basis for which both associated solutions are feasible."

Note that Theorem 16 only considers the case of finite optimal solutions. What happens when $\mathcal{P}$ (and/or $\mathcal{D}$) is infeasible or unbounded is still something to decide (see Problem Set 5).

On the other hand, if we think back to the physical meaning of Lagrange multipliers, I claim we can see something even stronger than the fact that the two linear programs have matching solutions[38]. It told us something about the structure of the solutions — that the only constraints that should have positive weight are the ones that are active. This is known as *complementary slackness*.

Theorem 17. *Let $\mathcal{P}$ be a minimization linear program with feasible solution $\mathbf{x}$, and let $\mathcal{D}$ be its dual (maximization) linear program with feasible solution $\boldsymbol{\lambda}$. Then $\mathbf{x}$ and $\boldsymbol{\lambda}$ are optimal solutions if and only if*

$$\begin{aligned} \lambda_i(\text{row}_i(\mathbf{A}) \cdot \mathbf{x} - b_i) &= 0 \qquad \textit{for all } i \\ x_j(\text{col}_j(\mathbf{A}) \cdot \boldsymbol{\lambda} - c_j) &= 0 \qquad \textit{for all } j \end{aligned}$$

In other words,

- $\lambda_i = 0$ whenever constraint i is non-active in $\mathcal{P}$
- $x_j = 0$ whenever constraint j is non-active in $\mathcal{D}$.

The proof of this uses the idea from the previous section that if you add up a bunch of nonnegative things and get 0, it means each of the individual entries must be 0.

5.2.3 References

1. Bertsimas and Tsitsiklis: Section 4.1–4.3

[38] See also Example 4.4 in the book.