Notes on (Occasionally Discrete) Linear Optimization

Aaron Pierce

Contents

1 Linear Programming

Linear programming is a method to optimize a function that is constrained by some other functions. The function you are optimizing (the objective function) and its constraints are both linear, hence *linear* programming.

Linear programming problems come in two parts, the objective function and the constraints. Those will be made up of a linear combination of the variables in the problem. We used the examples of foods, where each food has some amount of carbs, fat, protein, and sodium. Those would be your variables (called decision variables), denoted x_1, x_2, x_3, x_4 respectively, joined together in a column vector $x = (x_1, x_2, x_3, x_4)$. I would normally write the column vector out in, well, a column, but my instructor and textbook notate it like this so I may as well maintain consistency so as to not be confused when the exam arrives.

Our goal is to minimize some function of those variables, under some constraints maybe

$$\begin{aligned} \text{minimize} \quad & 2x_1 - x_2 + 4x_3 \\ \text{subject to} \quad & x_1 + x_2 - x_4 \leq 2 \\ & 3x_2 - x_3 = 5 \\ & x_3 - x_4 \geq 3 \\ & x_1 \geq 0 \\ & x_3 \leq 0 \end{aligned}$$

Unfortunately this example is a little nonsense when applied to the food. I'm supposed to eat negative protein? The idea is that the constraints are the ideal diet. You want your carbs plus your fat minus your sodium to be no more than 2. Whatever that means. And to make it worse, what does the objective function mean in this case? Maybe you want to save money, and eat as little as you can while still fulfilling all the requirements, and the coefficients are weights. Like it's 4 times as bad (expensive) to eat protein for some reason, because each unit of protein increases your objective function by 4 units, and you're shooting to minimize. Maybe that means cash in that case.

But don't get too bogged down with this example, it's super nonsense, but introduces the form. The diet problem in the textbook is more sane, I kinda

frankensteined that onto another problem and now I've typed too much to delete it all.

We can nicely write this system out with matricies. Everyone saw it coming. Linear is right there in the name!

However, when we're using matricies to represent systems of linear equations, it's easy, because they're **equ**ations. Everything equals something else! This is a system of linear inequalities, where each term has some different operator! Thankfully, we can represent them all in a standard form.

1.1 Standard/General Form

Representing our objective function is easy. It's linear (this is linear programming, after all), so all we need is a column vector of coefficients. Call this c. In the nonsense example above, $c = (2, -1m, 4, 0)$. We can easily create our objective function by multiplying c by x, the vector of decision variables from earlier ($x = (x_1, x_2, x_3, x_4)$). We take an inner product $c'x$ and get $2x_1 - x_2 + 4x_3$. Just like earlier! (That c' is c transpose. I had only ever seen transposes denoted with $c^{\mathbf{T}}$. By transposing, we multiply a 1xn matrix by an nx1 matrix, performing the inner product).

The constraints, naturally, will be collected in a matrix, where each column are the coefficients for each constraint. No problem, very familiar from linear algebra. But don't run off building columns just yet, we're getting ahead of ourselves. We first need to manipulate those equations to make them all have the same operator. We'll use $=$ because we've taken linear algebra and have good tools to solve those.

But hold on, we can't do that, right? How are we going to turn a $\geq$ constraint into an $=$ constraint without totally ruining the problem? It's super duper cool.

If we have some constraint that has to be at least as big as some constant b, that means that if you hand me a value of that constraint, I can subtract it by some number, and get b! Obvious, right, but super useful!

If we have a constraint $c'x \geq b$, we can transform this into $c'x - s = b$, where s is a new variable! So now we will be able to find solutions which will include some garbage value of s that we can discard, but will leave behind a valid satisfaction of the constraint! This new variable s is called a slack variable

(or surplus variable). One big problem, though. If we let s be anything, then we may not satisfy that the constraint is $\geq b$, because s could be negative, and we subtract the negative, to bring $c'x$ up to b. Very bad. If we had to bring $c'x$ up then it must have been less than b, so we'll always make sure our slack or surplus variables are ≥ 0. No funny business allowed with negative slack.

One final point about this standard form. Remember when I mentioned that it was weird that we had to eat negative protein? Standard form agrees that it's weird to eat negative protein, so it strictly enforces that $x_i \geq 0$, where that subscript i is some index into the vector of decision variables called x. This doesn't mean that you can't let protein give you negative nutrition, you just have to do that with a negative coefficient. Don't be fooled into thinking this changes the problem, all that negative protein still flies, but we have just represented it in a way that makes more sense. You don't eat negative food.

If the original LP problem has some variable not subject to any constraint, it will be able to go negative, which standard form won't allow. Not to worry, there's an easy fix. Split some x_j with no constraint into the sum of two new variables, $x_j^+ - x_j^-$, and now impose that each of those are ≥ 0. When $x_j^- > x_j^+$, then it will let you create a negative number with two positive variables. Very nice. Just like the slack variables, these are new variables that we will find values for in solutions.

All said, standard form is as follows:

$$\begin{aligned} \text{minimize} \quad & c'x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Where A is a matrix of coefficients to build the constraint functions. And while the standard form of a problem won't change the problem, it will still allow you to solve the original, it will probably have a different number of variables as you introduce slack variables.

There is a similar notion, the general form, that's a little more permissive.

$$\begin{aligned} \text{minimize} \quad & c'x \\ \text{subject to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

You can transform between these by using (or removing) slack variables, and can turn some random LP problem into general form by expressing $\leq$ constraints by negating everything and flipping the $\leq$, and equality can be represented by replacing it by an identical $\geq$ and $\leq$ constraint, and then transforming that $\leq$ constraint into a $\geq$. No problem at all.

Now we can represent the whole class of linear programming problems in standard or general form, allowing us to just have methods to solve problems in either of those forms, instead of something like differential equations where we have specific methods for much finer types of problems. Very convenient.

We'll extend this same charge in the next section. It's nice to be able to take some problem and transform it into an easier form to solve without modifying so much that the solutions are now meaningless. In the next section, we'll see a way to solve problems where the objective functions and constraints aren't necessarily linear

1.2 Piecewise Linear Convex Objective Functions

All of our techniques (which we haven't developed, by the way, but you can imagine some linear algebra magic can solve the standard form $Ax = b$ constraints) require linear objective functions and linear constraints. It's actually not so obvious to me why this is true. You can fall back on linear algebra and claim that we're only able to solve these because we have the mechanics to solve systems of linear equations and nothing else, so naturally we can only solve *linear* programming problems.

I need to review some linalg basics and understand why linearity is so crucial.

Anyways, linearity is crucial, and I'll stick with the linear algebra defence for now to keep myself from dwelling on it too long. However, we can get away with our functions only being kind of linear. That is, all the right linearity in all the right places.

When we're solving LP problems, we try to minimize some cost function. We know the calculus process to do that, find all the local mins and take the smallest of them. But some cost functions are just nicer to find minima for than others. For example, a convex function cannot have a local min that isn't a global min. I may be incorrect about this, but my understanding is that unless the convex function in question is linear, it has a single optima

that is a global min. That is super useful. It would be lovely if we could find some way to use convex cost functions in our linear programming problems.

Thankfully, we can! We can approximate any convex function by using a whole bunch of linear functions.

But before we approximate a convex function by using a piecewise linear function, we should formalize both of those things.

A function is convex when any two points on the curve can be connected by a line that is strictly above the curve segment between those two points. This essentially enforces that the function is always growing. If the function jumps above that line, it will have to go back down to reach the endpoint, so it can't always increase.

To formalize that, we need to show that $line \geq curve$. How do we get a line between those two points (call them x and y)?

$$\underbrace{x}_{\text{Start at the first point}} \quad \overbrace{+\lambda}^{\text{and move by a fraction from 0 to 1 of}} \quad \underbrace{(y-x)}_{\text{the vector that takes you to y}}$$

This is familiar from calc 3, but it's not how they format it in the textbook. Instead of $x+\lambda(y-x)$ they use $\lambda x+(1-\lambda)y$. This is just some easy algebra

$$\begin{aligned} x+\lambda(y-x) \\ x+\lambda y-\lambda x \\ (1-\lambda)x+\lambda y \end{aligned}$$

The variables are switched around but the form is the same. They say y is where you start, I called it x, no big deal, just be consistent. This has another neat interpretation, where $\lambda=0$ is entirely x, $\lambda=1$ is entirely y, and intermediate values kind of interpolate between x and y. Clearly the same thing, but a different way to intuiting about the same concept. Neat!

Okay so we have the line between the two points, $\lambda x+(1-\lambda)y$, and we said that it needs to be above the curve. To show that, we need to assert that every point on that line is higher than every point on the curve. We can get

points on the line from our parameterization earlier, so for every point on that line, the point on $f(x)$ is $f(\lambda x + (1-\lambda)y)$, but we need to be careful of what x and y really are. Before they were points on the curve, now they're points in the input space of the function. Still fine, just be mindful that we've kind of switched their meaning.

Now that we've downgraded x and y to being in the domain instead of the output space, we need to re-formulate points on the line. Now our starting point is $f(x)$ instead of just x, and our end point is $f(y)$, so the line overall is $\lambda f(x) + (1-\lambda)f(y)$. Basically the same.

Now that we've got both of those, we just plug it into the inequality we made earlier!

$$\overbrace{f(\lambda x + (1-\lambda)y)}^{\text{height of curve}} \leq \overbrace{\lambda f(x) + (1-\lambda)f(y)}^{\text{height of line}}$$

If this holds, your function is convex! (And if the opposite holds, that *curve* $\geq$ *line* everywhere, then it's concave!)

Why do we care? Take a peek at that inequality again. Doesn't it look a whole lot like saying that the function is at most linear? We're almost pretending that f is linear! We distribute it into $f(\lambda x) + f((1-\lambda)y)$ and pull out constants to get $\lambda f(x) + (1-\lambda)f(y)$. So if we pretend f is linear, then it won't make the function an bigger than the line.

Maybe you can also see why this may be vaugely useful. We're going to do this to a convex function with the intent to minimize it. By pretending it's linear, we won't increase it all that much, just up to the line. This will change our function a little, but we have a nice upper bound on that change, so we know our minimas won't increase so much as to spoil them (hopefully. This is all conjecture.)

But if we want to do this, converting our convex function into some lines, we would really like to do it more than once. It's definitely not okay to approximate the entire convex function by a single line, imagine trying to do that with a parabola. What line could possibly approximate all of that? But maybe with enough pieces we would do well enough. To prepare for this, let's thing about piecewise linearity.

If we're splitting this convex function into lines, it's not going to be smooth

anymore. It's going to have jagged edges. We're kinda crystallizing the function, making all of its geometry very rigid and straight. This won't give us a linear function, but it will give us a function made up by pieces of linear functions. It's a piecewise linear function!

But here's the kicker. A line goes on forever! If you hand me a pile of lines, I can't just stick them next to each other and pretend that it parameterizes the curve. It'll intersect itself and have lines shooting every which way. It won't look like the curve at all.

Instead, we should have some way to join those lines up, selecting pieces strategically to best approximate the curve. This is easier than you might guess!

If we take the max of each line at every point, it'll work!

Why? Because we know whatever we're approximating is convex, meaning that the line that connects two points is above the curve. Any lines below the curve definitely shouldn't be used, and any lines above the curve are (I could be wrong about this) just as good as any other, so we can arbitrarily choose one by just choosing the biggest. This may (more conjecture) have the nice side effect of choosing a single function for longer, which reduces the complexity of the approximation and makes it easier to solve.

In notation, if you have some list of vectors $c_1, \ldots, c_m$ that define lines, and some scalars $d_1, \ldots, d_m$ that represent y-intercepts, then the function

$$f(x) = \max_{i=1\ldots,m} (c_i' x + d_i)$$

is piecewise linear because you are selecting sections of purely linear functions, so each selection will be linear.

For the same reasoning, if you have a bunch of convex functions, then doing the same *max*ing at every point generates a convex function, because each piece always increases (it's convex), and you always choose the biggest of those pieces, so your new function will always increase, too.

I'm being quite liberal about my use of 'increase' here. x^2 doesn't always increase, the left half of the graph is definitely decreasing, but it maintains concavity by the second derivative being positive. It's a little harder to see why *max*ing also covers the decreasing half, but the same-ish reasoning holds.

If all of your pieces are decreasing, then you'll chose the one that decreases the least, but it will still decrease nonetheless, and if one piece starts to increase, then it will beat all the other ones that are decreasing, (forming a local min here) and can only be out-maxed by something bigger, so it won't decrease again to break concavity.

Okay, that was a bunch of preliminaries, now time for the big event. How do we work these piecewise linear convex functions into our LP problems?

Let's say its our objective function that's convex, and let's say we've already magically got some lines that will approximate it well. That means our new cost function is

$$\max_{i=1\ldots,m} c_i'x + d_i$$

But oh boy does that seem hard to minimize. It's not even differentiable! But boy oh boy do I have the trick for you. The maximum of some set of numbers is the smallest number greater than or equal to all the numbers. Weird way to say that, let me try to say it a different way.

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & z \geq c_i'x + d_i, \quad i = 1, \ldots, m \end{aligned}$$

Oh, that looks like a familiar form. Find the minimum number that's greater than or equal to all the other numbers! (Where those numbers just so happen to be the lines that approximate our convex cost function!)

Now let's slap on our constraints (in general form, why not)

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & z \geq c_i'x + d_i, \quad i = 1, \ldots, m \\ & Ax \geq b \end{aligned}$$

And we've got a whole stinkin' LP problem using our piecewise linear convex objective function! Isn't that neat? We've moved the problem into more comfortable grounds. We can easily take a derivative of z, and we can solve that new constraint just like any other. So much easier to handle.

We can use the same trick for piecewise linear constraints as well. A piecewise linear function expressed via the maxing of a bunch of linear functions can just turn into constraining each individual line. For example, if our constraint is $f(x) \leq h$, and $f(x)$ is piecewise linear as a result of maxing a bunch of functions, then you can just say that each $f_i'x + g_i$ must be $\leq h$. The actual value of $f(x)$ is the max of all of those functions, so it's no problem to say that all of the functions must be $\leq h$. If the biggest one (the actual value of $f(x)$) is less than h, than so are all the other ones (because it's the max, the rest are smaller.), and all we care about is the biggest one at any given point, so we're all good.

Spectacular. We've seen how to use piecewise linear convex functions in our LP problems. They can be represented as the max out of all the pieces at any given point, so we can turn each piecewise linear convex function into a bunch of constraints, one for each piece, and they become usable.

In notation, that general pattern follows that in LP problems, we can express

$$\max_{i=1,\ldots,m} c_i'x + d_i$$

as a bunch of constraints:

$$c_i'x + d_i \leq z$$

Where z is a new decision variable to be minimized in the case of an objective function, or some constant if it's a constraint.