Math 340 Lecture 19

The revised simplex method. Our next big topic is the revised simplex method. Actually this is a strange name for it, since it ends up being the opposite of what you might expect: we aren’t revising the procedure of the simplex method, but instead our implementation and interpretation of it. What we’ll be doing will look very different - it will be working with matrix notation and not dictionaries at all - but in the background the exact same computation is going on.

Reminder of matrix notation. It’s been a few weeks, so let’s review how we set things up in matrix notation. Our primal LP in matrix form is

\[ \text{Maximize } z = \vec{c} \cdot \vec{x}, \text{ subject to } A\vec{x} \leq \vec{b} \text{ and } \vec{x}_D \geq 0 \]

where \( \vec{x} = \vec{x}_D \) is the vector of decision variables here. If \( \vec{x}_S \) is the vector of slack variables, we can augment them together and form

\[ \vec{x}_{\text{aug}} = \begin{bmatrix} \vec{x} \\ \vec{x}_S \end{bmatrix}. \]

(I’m slightly changing my notation here from last time and leaving \( \vec{x} \) for the vector of decision variables, which can also be denoted \( \vec{x}_D \), and writing \( \vec{x}_{\text{aug}} \) rather than \( \vec{x} \) for the combined vector). Using the slack variables we could rewrite the constraint \( A\vec{x} \leq \vec{b} \) as \( A\vec{x} + \vec{x}_S = \vec{b} \) and then \( A_{\text{aug}}\vec{x}_{\text{aug}} = \vec{b} \) for \( A_{\text{aug}} = \begin{bmatrix} A & I \end{bmatrix} \).

Similarly we extend \( \vec{c} \) to \( \vec{c}_{\text{aug}} \) by adding a bunch of 0’s and can then rewrite our problem as

\[ \text{Maximize } z = \vec{c}_{\text{aug}} \cdot \vec{x}_{\text{aug}}, \text{ subject to } A_{\text{aug}}\vec{x}_{\text{aug}} = \vec{b} \text{ and } \vec{x}_{\text{aug}} \geq 0. \]

Then, the main thing we talked about was how to read off a dictionary in matrix form. Any dictionary involves splitting up the variables \( \vec{x}_{\text{aug}} \) into basic and nonbasic variables; we let \( \vec{x}_B \) be a vector of the basic variables and \( \vec{x}_N \) a vector of nonbasic variables. The dictionary them consists of (1) a system of equations writing \( \vec{x}_B \) in terms of \( \vec{x}_N \) and (2) an equation writing \( z \) in terms of \( \vec{x}_N \). I won’t go through the details again but we worked this out to be

\[
\vec{x}_B = \begin{bmatrix} A_B^{-1} \vec{b} \\ -A_B^{-1}A_N\vec{x}_N \end{bmatrix}, \quad z = \vec{c}_{\text{aug}} \cdot \vec{x}_{\text{aug}} = (\vec{c}_B^\top - \vec{c}_B^\top A_B^{-1}A_N)\vec{x}_N.
\]

Here \( \vec{c}_B, \vec{c}_N \) came from splitting up \( \vec{c}_{\text{aug}} \) in the same way we split up \( \vec{x}_{\text{aug}} \), and \( A_B \) and \( A_N \) came from similarly splitting up the columns of \( A_{\text{aug}} \). An important fact here was that if \( \vec{x}_B \) arose as basic variables from running the simplex method, then \( A_B \) was invertible!

Basic idea of the revised simplex method. One iteration of the simplex method involves starting with a dictionary (determined by choosing \( \vec{x}_B \) and \( \vec{x}_N \) as we talked about above) and pivoting to a new dictionary (corresponding to a new choice of vectors \( \vec{x}_B' \) and \( \vec{x}_N' \), where what we did was picked an entering variable in \( \vec{x}_N \) and an exiting variable in \( \vec{x}_B \) and swapped them). So what I want to do now is think about exactly what computations are involved in choosing this pivot, and how those computations can be expressed in matrix form. Roughly the idea is that we know our dictionary looks something like

\[
\begin{array}{cccccc}
  x_1 & = & ? & -?x_3 & -?x_4 & -?x_6 \\
  x_2 & = & ? & -?x_3 & -?x_4 & -?x_6 \\
  x_5 & = & ? & -?x_3 & -?x_4 & -?x_6 \\
  z & = & ? & +?x_3 & +?x_4 & +?x_6
\end{array}
\]

and we want to figure out how we can do a step of the simplex method while doing the least possible work in computing and filling in only as many '?'s as we need.

So what do we do in the simplex method? Well, first we need to decide what variable to enter, and we follow the standard rule: we need to pick the variable in \( \vec{x}_N \) that has the largest coefficient in our expression
for $z$. Following our matrix notation these coefficients are given by $(\vec{c}_N^\top - \vec{c}_B^\top A_B^{-1} A_N)$. So we just compute that (we have all of the data involved from the initial setup of the problem, plus our choice of $\vec{x}_B$ and $\vec{x}_N$) and pick out our entering variable.

Next we need to pick our exiting variable. Thinking back to the standard rule, this required looking at both the constants in our dictionary and the coefficients of the entering variable. From the matrix form for the dictionary we know the constants are $A_B^{-1} \vec{b}$ and the coefficients are one column from $-A_B^{-1} A_N$ - we can pick this column out as $A_B^{-1} \vec{A}_j$, where $\vec{A}_j$ is the single column from $A_N$ (or just from $A_{aug}$) corresponding to the variable $x_j$.

The idea is then that we want to think of increasing $x_j$ until we hit a point where one of the constraints becomes zero. So we can think of putting a parameter $t$ in for $x_j$ (and keeping the other nonbasic variables as 0), and then the values of our basic variables $\vec{x}_B$ are given by

$$(A_B^{-1} \vec{b}) - t(A_B^{-1} \vec{A}_j).$$

Once we have this we can pick our exiting variable: for each entry in this vector (corresponding to a basic variable) we find the value of $t$ that makes it zero, if there is one, and pick the smallest such $t$ (the tightest constraint).

And then... we’re done our iteration of the simplex method! We’ve found our entering variable and our exiting variable, and can just swap those two between $\vec{x}_B$ and $\vec{x}_N$ to get our description of the next dictionary. So we can summarize our (extremely preliminary version) of doing one step of the revised simplex method as follows:

1. Compute $\vec{c}_N^\top - \vec{c}_B^\top A_B^{-1} A_N$.

2. The largest coefficient in the row vector we just computed corresponds to a variable $x_j$ in $\vec{x}_N$ that we choose as our entering variable. (As usual in the event of a tie we pick the $x_j$ with a smaller index).

3. Compute $A_B^{-1} \vec{b}$ and $A_B^{-1} \vec{A}_j$.

4. For each entry of $(A_B^{-1} \vec{b}) - t(A_B^{-1} \vec{A}_j)$ find what value of $t$ makes it zero (i.e. just divide the entry from the first vector by the corresponding entry from the second).

5. Take the smallest nonnegative value from the list above, and take the exiting variable to correspond to the row it was from. (Again pick the smaller index in the event of a tie).

So that’s our “lazy” method of doing a pivot of the simplex method while filling in as few of the $?’s in our dictionary as were needed. That seems great, because doing less work computing things we didn’t actually need is always nice! But on second thought, this might not actually be less work - if you actually try to do this, you’ll soon run into realizing that computing $A_B^{-1}$ itself is a lot of work, and that shows up repeatedly in these formulas. So if you just go and plod through this procedure naively for each step, you’re likely to be a lot slower than just doing the simplex method like normal.

The actual revised simplex method, version 1. So how can we make the revised simplex method actually work reasonably efficiently, i.e. not spend a massive amount of time computing $A_B^{-1}$ at each step?

- Approach 1: Notice that we’re not actually ever using $A_B^{-1}$ by itself, just $A_B^{-1}$ times vectors. So use linear algebra to find these products more efficiently.

- Approach 2: Reduce computations by keeping track of $A_B^{-1}$ from step to step - if we have $A_B^{-1}$ at the beginning of one iteration of the simplex method, we can try to use it to get the new $A_B^{-1}$ for the next iteration without recomputing the whole thing.

We’ll actually give two different ways to do the simplex method, one based primarily on each approach.