Math 3272: Linear Programming[1] Mikhail Lavrov

Lecture 11: Worst cases for the simplex method

September 20, 2022 *Kennesaw State University*

1 The terrible trajectory

What is the worst case for the simplex method? How many pivoting steps do we need?

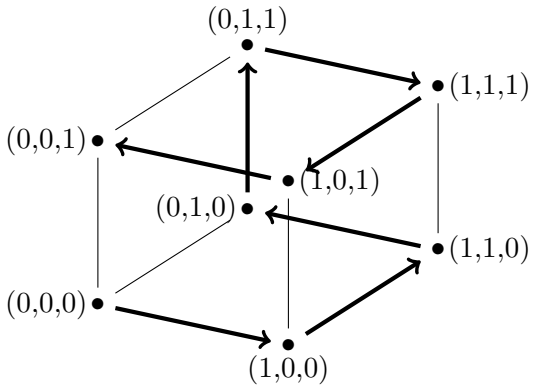
Today, we'll show that for all the pivoting rules we know, the worst case is pretty bad: we can cook up a linear program with only d variables and $2d$ constraints in which we take around 2^d steps. This is approximately as bad as it could possibly get, since there are at most $\binom{2d}{d} < 4^d$ possible basic solutions for such a linear program.

First, consider the following linear program (whose feasible region forms a d-dimensional hypercube):

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^d}{\text{maximize}} \quad & x_d \\ \text{subject to} \quad & 0 \le x_1 \le 1 \\ & 0 \le x_2 \le 1 \\ & \vdots \\ & 0 \le x_d \le 1 \end{aligned}$$

This is not actually the worst case for the simplex method, under any reasonable pivoting rule. The initial basic feasible solution is $\mathbf{x} = \mathbf{0}$, the only variable with positive reduced cost is x_d, and pivoting on x_d gets us to an optimal solution within one step. But tiny modifications will make it much much worse!

First of all, there are some really inefficient trajectories possible in theory: paths we can take going from $(0, 0, \ldots, 0, 0)$ to $(0, 0, \ldots, 0, 1)$ that visit every other vertex of the hypercube in between. Here is an illustration of such a path in the 3-dimensional case (when the feasible region is a cube):



[1]This document comes from an archive of the Math 3272 course webpage: http://misha.fish/archive/3272-fall-2022

We'll call this path the "terrible trajectory". (Despite the alliteration, this is not established terminology.) The terrible trajectory has a fairly simple recursive definition: to follow it in d-dimensions, first follow the $(d-1)$-dimensional terrible trajectory (keeping $x_d = 0$), then change x_d to 1, then follow the $(d-1)$-dimensional terrible trajectory again, but in reverse.

Second, note that this trajectory is actually kind of close to being reasonable for the linear program we want to solve. Every single step of the terrible trajectory is neutral with respect to the objective value (it does not change x_d), except for one step, which increases it. So it's possible that if we push the corners around a bit, then every single step of the terrible trajectory will increase the objective value. And at that point, we're close to tricking the simplex method into following it.

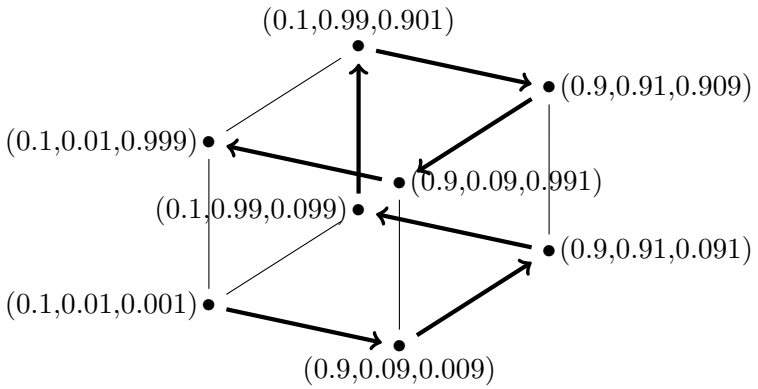
2 Tricking Bland's rule

"Easy mode" is tricking Bland's rule into following the terrible trajectory. To make this happen, we modify the linear program as follows:

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^d}{\text{maximize}} \quad & x_d \\ \text{subject to} \quad & 0.1 \le x_1 \le 1 - 0.1 \\ & 0.1x_1 \le x_2 \le 1 - 0.1x_1 \\ & 0.1x_2 \le x_3 \le 1 - 0.1x_2 \\ & \qquad\qquad \vdots \\ & 0.1x_{d-1} \le x_d \le 1 - 0.1x_{d-1} \end{aligned}$$

The value 0.1 could be replaced by any reasonably small constant. The smaller we make it, the closer we get to the original cube, and if we set it to 0, we just get back that cube.

Here's what the terrible trajectory looks like for this linear program, in 3 dimensions. (It's a bit of a lie, because with the modification, the feasible region is no longer a perfect cube.)



You can see that in this trajectory, the objective values steadily increase:

$$0.001 < 0.009 < 0.091 < 0.099 < 0.901 < 0.909 < 0.991 < 0.999.$$

It turns out that, with a natural choice of variable ordering, Bland's rule will end up picking this trajectory. Let's first add slack variables to the problem, rewriting $0.1x_{i-1} \le x_i \le 1 - 0.1x_{i-1}$

as

$$\begin{cases} 0.1x_{i-1} - x_i + w_i^0 = 0 \\ 0.1x_{i-1} + x_i + w_i^1 = 1 \end{cases}$$

Here, the superscript in w_i^0 and w_i^1 is not an exponent: it's an extra index, since we have $2d$ slack variables. To explain the naming convention: w_i^0 is the slack variable for the lower bound on x_i, and when $w_i^0 = 0$, x_i is close to 0. Meanwhile, w_i^1 is the slack variable for the upper bound on x_i, and when $w_i^1 = 0$, x_i is close to 1.

We have to use the two-phase simplex method for this problem, since $\mathbf{0}$ is not feasible, but let's skip ahead and suppose we arrive at the correct basic feasible solution we wanted: the corner point $(0.1, 0.01, \ldots, 0.1^d)$. Here, the variables $x_1, x_2, \ldots, x_d$ are all basic—and they'll stay basic forever, because none of them can be 0. In order to start out at this corner point, our nonbasic variables (corresponding to the tight constraints) must be $w_1^0, w_2^0, \ldots, w_d^0$; the slack variables $w_1^1, w_2^1, \ldots, w_d^1$ are basic.

In each of the basic feasible solutions we can encounter, exactly one of w_i^0 and w_i^1 is basic for each i When $x_i \approx 0$, w_i^0's constraint is tight, so w_i^1 is nonbasic. When $x_i \approx 1$, w_i^1's constraint is tight, so w_i^1 is nonbasic. Moving from one corner point to an adjacent one means pivoting so that w_i^0 enters the basis and w_i^1 leaves for som i, or vice versa.

If we put the slack variables in the order

$$w_1^0, w_1^1, w_2^0, w_2^1, \ldots, w_d^0, w_d^1$$

then Bland's rule will pivot on w_1^0 or w_1^1 whenever this improves the objective value, which is every other step. In between those, it will pivot on w_2^0 or w_2^1 as often as possible, and so on. In 3 dimensions, the sequence of entering variables will be

$$w_1^0, w_2^0, w_1^1, w_3^0, w_1^0, w_2^1, w_1^1$$

The pattern continues in higher dimensions: the subscripts will follow the sequence

$$1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, \ldots,$$

which traces out the terrible trajectory.

To prove that this behavior continues in a bit more detail, consider that for as long as w_d^0 remains nonbasic, x_d will be stuck at $0.1x_{d-1}$, and so maximizing x_d will be equivalent to maximizing x_{d-1}. This, together with the first $d-1$ inequalities in our linear program, looks exactly like the $(d-1)$-dimensional linear program.

As long as pivoting makes sense in the $(d-1)$-dimensional linear program, w_d^0 will not even be considered as an entering variable, because it has the lowest priority. So we'll follow the exact same sequence of pivots as in $d-1$ dimensions. After $2^{d-1} - 1$ steps, we'll end at the point where $w_1^0, w_2^0, \ldots, w_{d-2}^0$ and w_{d-1}^1 are nonbasic, and none of them are worth pivoting on.

That's the first, and only, step at which we'll pivot on w_d^0. After we do, w_d^1 becomes nonbasic. Now, instead of having $x_d = 0.1x_{d-1}$, we have $x_d = 1 - 0.1x_{d-1}$, so maximizing x_d is equivalent to *minimizing* x_{d-1}. So the simplex method will continue the first $2^{d-1} - 1$ pivot steps in reverse: for

every step that made sense to maximize x_{d-1}, undoing it now makes equal sense to minimize x_{d-1}, and has the same priority in our variable ordering.

After a total of $(2^{d-1}-1)+1+(2^{d-1}-1)$ steps, we'll finally reach the point where $w_1^0, w_2^0, \ldots, w_{d-1}^0$ and w_d^1 are nonbasic, which is our optimal solution.

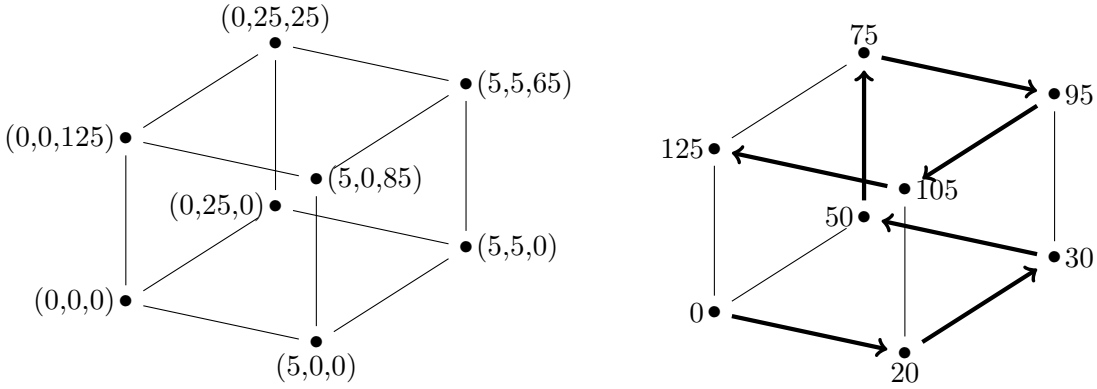
3 The Klee–Minty cube

You may think that this example simply exploits a flaw in Bland's rule (which, after all, does not try very hard to pick a good pivot). But a famous linear program called the Klee–Minty cube shows that the highest-cost pivoting rule is also vulnerable to the "terrible trajectory".

The Klee–Minty linear program in d dimensions is given below (I've written the inequalities backwards to make the pattern easier to see):

$$\begin{aligned}
\underset{\mathbf{x}\in\mathbb{R}^d}{\text{maximize}} \quad & 2^{d-1}x_1 + 2^{d-2}x_2 + \cdots + x_d \\
\text{subject to} \quad & 5 \geq x_1 \\
& 25 \geq 4x_1 + x_2 \\
& 125 \geq 8x_1 + 4x_2 + x_3 \\
& 625 \geq 16x_1 + 8x_2 + 4x_3 + x_4 \\
& \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
& 5^d \geq 2^d x_1 + 2^{d-1}x_2 + 2^{d-2}x_3 + \cdots + 8x_{d-2} + 4x_{d-1} + x_d \\
& x_1, x_2, \ldots, x_d \geq 0.
\end{aligned}$$

This is also shaped kind of like a hypercube in d dimensions (and a cube in 3 dimensions), but it is very distorted. Here is a picture of the "terrible trajectory" for the Klee–Minty cube, with coordinates given in the figure on the left, and their objective values on the right. (For this linear program, the shape of the cube is an incredible lie, but the adjacencies between the corners are the same.)



The best way to understand why this cube tricks the highest-cost rule is to try doing it, and see how the reduced costs change. But essentially, this construction exploits a weakess of the pivoting rule that we've already talked about: it's sensitive to changes in units. To get the highest-cost rule to

pick earlier variables over later ones, it's enough to set up the problem so that a very small change in x_1 or x_2 has the same effect as a very large change in x_{d-1} or x_d. However, the constraints are set up so that the distance that it's possible to go in the x_{d-1} or x_d direction is always much larger.

4 Closing remarks

4.1 Other pivoting rules

Other pivoting rules which we have not discussed exist. Some of them are slightly less vulnerable to this strategy: for example, there is the "best neighbor" pivoting rule, which considers all possible entering variables, and determines which one will lead to the greatest improvement in the objective function.

The best neighbor pivoting rule cannot be fooled by any of the examples we saw today. After all, in both examples, the optimal solution is within one pivot step of the starting solution. The best neighbor pivoting rule will notice this and solve the problem in one step.

However, this does not mean that the best neighbor pivoting rule is always guaranteed to be efficient. It's possible to cook up examples in which it, too, takes an exponentially long time. If the best neighbor pivoting rule is within one step of the optimal solution, it will work well; but if it's within two steps, and the first step does not look promising, the pivoting rule will remain happily oblivious.

There are complicated pivoting rules out there which have a better worst case: on a linear program with n inequalities and d variables, the number of pivot steps they take is bounded by functions such as $C^{\sqrt{d}\log n}$ for some constant C, which is better than exponential. But the upper bound we dream of is "a polynomial function in n and d", and it's an open problem whether any pivoting rule can achieve this.

4.2 The average case

Shouldn't we forget all about the worthless simplex method now that we know the truth about its worst-case behavior?

In fact, there *are* other algorithms for solving linear programs that are alternatives to the simplex method, and have better worst-case guarantees on running time. But linear programming is the one case where it turns out that the worst-case-exponential algorithm is better than the worst-case-polynomial algorithm.

It is very unusual to encounter examples like the Klee–Minty cube, where it makes sense to pivot on the same variable over and over and over and over again. One argument for this is that we don't encounter cases like this by generating linear programs at random, but this is not a particularly strong argument: most linear programs we want to solve aren't random!

Other studies have shown that the simplex method is efficient under **smoothed analysis**: when we take any linear program and introduce a bit of random noise in the coefficients, the average result is solved efficiently. (You can see how this would completely destroy our examples today, where the behavior is very sensitive to comparisons between numbers like 0.0001 and 0.00001.)